Fingerprinting Bitcoin entities using money flow representation learning

Natkamon Tovanich, Rémy Cazabet

Tovanich and Cazabet
Applied Network Science (2023) 8:63
https://doi.org/10.1007/s41109-023-00591-2

Applied Network Science

RESEARCH Open Access

Fingerprinting Bitcoin entities using money flow representation learning

Natkamon Tovanich[1*] and Rémy Cazabet[2]

*Correspondence:
natkamon.
tovanich@polytechnique.edu

[1] Blockchain@X Research Center, CREST, École Polytechnique, Institut Polytechnique de Paris, Palaiseau, France
[2] Univ de Lyon, CNRS, Université Lyon 1, LIRIS, UMR5205, Villeurbanne, France

Abstract

Deanonymization is one of the major research challenges in the Bitcoin blockchain, as entities are pseudonymous and cannot be identified from the on-chain data. Various approaches exist to identify multiple addresses of the same entity, i.e., address clustering. But it is known that these approaches tend to find several clusters for the same actor. In this work, we propose to assign a fingerprint to entities based on the dynamic graph of the taint flow of money originating from them, with the idea that we could identify multiple clusters of addresses belonging to the same entity as having similar fingerprints. We experiment with different configurations to generate substructure patterns from taint flows before embedding them using representation learning models. To evaluate our method, we train classification models to identify entities from their fingerprints. Experiments show that our approach can accurately classify entities on three datasets. We compare different fingerprint strategies and show that including the temporality of transactions improves classification accuracy and that following the flow for too long impairs performance. Our work demonstrates that out-flow fingerprinting is a valid approach for recognizing multiple clusters of the same entity.

Keywords: Bitcoin, Money flow, Taint analysis, Graph embedding

Introduction

Bitcoin is the oldest and most used cryptocurrency, attracting broad interest from the general public and researchers. In contrast to traditional financial networks, transactions can be observed by anyone on the public blockchain, on which users exchange Bitcoins pseudonymously. This data allows researchers to study economic activities in fine detail. One of the objectives of those research is to understand how the Bitcoin socio-technical system works, particularly (1) Who are the important actors of the Bitcoin economy? (Lischke and Fabian 2016; Liu et al. 2021; Meiklejohn et al. 2016); (2) How is the network of transactions organized? (Lischke and Fabian 2016; Nerurkar et al. 2021; Vallarano et al. 2020); and (3) How to identify and track illegal activity? (Chainalysis Team 2022; Bartoletti et al. 2021; Weber et al. 2019).

Tracing the flow of money—where the money goes, to whom, and when—is also an essential task in cryptocurrencies and critical for financial forensics to trace money from suspicious sources and characterize different users' behaviors. However, it is still an understudied and challenging question in the Bitcoin research domain.

In this work, we explore how to leverage money-flow analysis to recognize source actors. We propose an original way to synthesize the money flow from a given source into a concise dynamic network called a *taint network*. We subsequently apply whole graph embedding methods to assign taint networks to their origin actor automatically. Beyond the demonstration that each actor has a characteristic flow allowing us to recognize it, our method can also be helpful for actor tracking as well as actor deanonymization tasks. The embedding of taint flows from different actors is a promising feature for downstream tasks in machine learning models to classify the role of actors (Jourdan et al. 2018; Zola et al. 2019) or predict illegal transaction activities in the Bitcoin blockchain (Weber et al. 2019).

Contributions

In a first conference article (Tovanich and Cazabet 2023), we have shown that taint flows indeed varies from actor to actor, and can thus be used as *fingerprints* of their source. We demonstrated the potential of our approach on a particular type of entities (i.e., mining pools).

In this article, we focus on proposing effective methods to use those taint flows in order to recognize if two flows are from the same actor.

(1) We applied our taint flow embedding dataset to two more datasets, focusing not only on mining pools, but on different kinds of entities, i.e., (1) ransomware and (2) various well-known Bitcoin entities from WalletExplorer database.
(2) We selected a larger number of entities for each dataset and sampled a larger number of taint flows for each entity.
(3) We proposed different configurations to train representation learning models and evaluated which method provides the best performance on the entity classification task.
(4) We studied the role of walk lengths on the taint flows and evaluated which maximum length to choose to optimize the results to differentiate entities.

Context: Bitcoin transaction network

All Bitcoin transactions are stored in a blockchain, i.e., a public, decentralized digital ledger that stores transactions by time order. In practice, each block of this blockchain contains a set of transactions marked by a common timestamp. Each bitcoin transaction corresponds to a transfer of Bitcoin but cannot be interpreted precisely as a usual money transaction between individuals as we are all used to. Indeed, some aspects of these transactions arise from the cryptographic constraints and the peculiarities of the Bitcoin protocol. In particular, Bitcoin uses the unspent transaction output (UTXO) transaction model (Nakamoto 2008). According to it, transactions do not transfer money from one account to another; instead, each output—each UTXO—of a transaction represents an amount of coin belonging to a known Bitcoin address—a cryptographic public key. The rightful owner of the UTXO can use the corresponding private key to claim the money. The owner can spend the UTXO(s) by signing them as input(s) in a new transaction and sending new UTXO output(s) to recipients' addresses.

A Bitcoin *transaction network* can be modeled as a chain of UTXOs as depicted in Fig. 1. A transaction (tx) can be represented as a node. A directed edge represents a transfer of UTXO(s) from one transaction to another. We will refer to the in-edge and out-edge of tx as input(s) (tx_{in}) and output(s) (tx_{out}), respectively. Each UTXO edge (e) is characterized by the amount (value) of Bitcoin ($v(e)$) and the owner's address of that UTXO (e_{owner}). It also contains references to the receiving ($e_{receive}$) and spending transaction nodes (e_{spend}). The total value of a transaction, i.e., the sum of its outputs, is noted ($v(tx)$). The time of the transaction is noted as ($t(tx)$).

A consequence of this mechanism is that Bitcoin's users do not possess, in general, a single *account number* as in a retail bank, in the form of a unique public address. Instead, a typical Bitcoin user will have a collection of cryptographic key pairs—a public key and its associated private key—in their wallet at any given time. These keys allow the user to control (in other words, spend) the corresponding UTXO. Since creating these new keys can be done at no cost and immediately, it is recommended for actors that care about their privacy to generate a new public key every time one receives a new transaction (Nakamoto 2008). Nowadays, this is done automatically by most public wallet applications and services.

Related works

As the oldest cryptocurrency, still having the highest market capitalization (CoinMarketCap 2023), Bitcoin has been widely studied as a representative example of *Unspent Transaction Output (UTXO)* blockchains. Its transaction network has been largely studied to understand the collective behavioral patterns of its users (e.g., Lischke and Fabian 2016; Nerurkar et al. 2021; Reid and Harrigan 2013; Kondor et al. 2014; Maesa et al. 2019).

Our work more precisely build on two types of contributions: on the one hand, methods working on entity identification did not consider the transaction flows. On the other hand, methods that use some form of transaction flows still need to address the problem of entity identification. Our work is thus the first to use transaction flow for entity identification tasks.

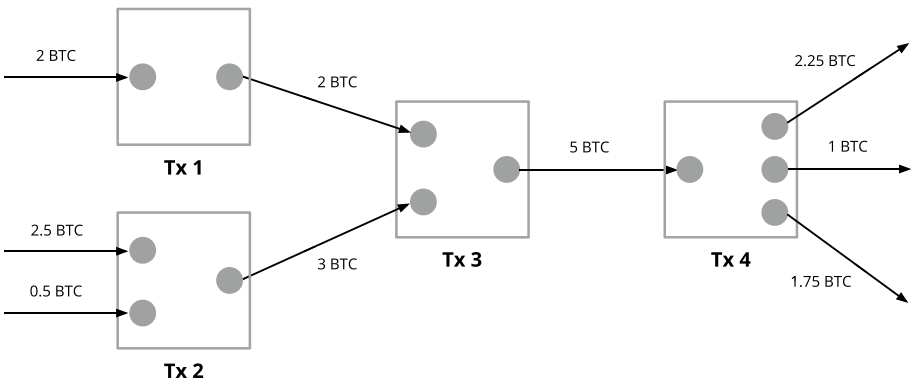

Fig. 1 Bitcoin transaction network model

Entity identification

The data found in the blockchain do not allow analyzing Bitcoin entities (i.e., users) directly because a single entity can control many Bitcoin addresses, and blockchain data only contains address activity. The first task required for any entity-level analysis thus consists in doing *entity identification*, which consists in finding groups of addresses belonging to the same entity. The most common approach, *address clustering*, consists of directly searching for groups of addresses in the blockchain that are likely to belong to the same entity. Although nothing in the protocol allows to know whether two addresses belong to the same entity, an efficient method called the *common input heuristic* (Harrigan and Fretter 2016) exists to solve the problem partially. This heuristic relies on the fact that entities, when they control several UTXOs of small amounts, have practical and economic incentives to join them in the inputs of the same transaction. An obvious example is when an actor needs to make a payment of x Bitcoins but controls only two outputs of value $< x$. Furthermore, there is generally no reason for two distinct actors to make a common transaction, i.e., a transaction having as input UTXOs controlled by different entities. Consequently, the first preprocessing approach of most Bitcoin analysis works consists of addressing clustering by leveraging the common input heuristic, which can be done efficiently. In this work, we used clustering heuristics implemented in BlockSci (Kalodner et al. 2020), having the advantage of also considering some exceptions to this heuristic known as CoinJoin (Goldfeder et al. 2017).

The limit of the common input heuristic is that it misses many address associations. Several works have proposed to improve this heuristic by using *change detection*. The principle is simple: when an actor needs to make a payment of an amount x, but controls an UTXO—or a combination of UTXO—whose sum is $y > x$, then it needs to send back the difference, $y - x$, to itself in an UTXO. This type of UTXO is called a change output, and many methods have been proposed to identify them, based on heuristics (Meiklejohn et al. (2016)) or machine learning approaches (see Ramos Tubino et al. (2022); Möser and Narayanan (2022) for recent articles on the topic). The machine learning approaches can be unsupervised and consider the transaction network as a whole (Cazabet et al. 2018), or be supervised and consists in describing each UTXO using features such as its amount, the number of decimals in its value, the number of times it has been used before, etc.

Once a change transaction detection has been achieved, the result can be used for a second-level entity identification: (1) the common input heuristic is used to find address clusters, and (2) change transactions are used to assign multiple address clusters to the same entity.

However, this approach has some limits: (1) It can lead to dramatic incorrect entity merging, because any UTXO wrongly identified as a change address can have a catastrophic consequence if it leads to wrongly merging two large clusters; (2) These methods are unable to solve the problem of entities keeping separate wallets. To understand this second problem, consider the case of a large company having hundreds or thousands of clients. If they manage a single pool of addresses, it is possible that their whole activity can be tracked using the common input heuristic and change detection. But instead, the company might choose to create, for instance, subgroups of 100 or 1000 clients—a large enough group to benefit from the economic and technical advantages—and manage

them in isolation so that one cannot track its whole activity. Although we have yet to know how actual companies manage their address pools, they are likely to use such strategies, as websites tracking well-known entities such as WalletExplorer (Janda 2023) often manually identify several address clusters for the same company.

Therefore, no common input and change detection methods can match those subclusters. On the contrary, the method we propose in this article can, in principle, achieve this objective or at least give us hints about such possible matches. By assigning activity-based fingerprints to entities, two different pools of addresses belonging to the same entity should have a similar fingerprint. Those addresses from different pools could be matched, even if they never interacted directly.

Methods leveraging transaction flow

Entity identification is one of many tasks on the Bitcoin transaction network. Other popular tasks include (1) Prediction of entity roles or categories (e.g., mining pools, wallet, exchange, and marketplace) and (2) Detection of illegal or suspect entities/transactions (e.g., scam, ransomware, and dark nets). Both of those popular tasks can be tackled with various approaches. Most works use a summary of the entity's activity by computing descriptive features (e.g., frequency and amount of transactions, centralities) (Liu et al. 2021; Bartoletti et al. 2018; Harlev et al. 2018; Akcora et al. 2020; Lin et al. 2019; Goldsmith et al. 2020; Michalski et al. 2020; Gomez et al. 2022; Xiang et al. 2022). Other approaches, however, leverage not only the actor's activity but also its neighbors' activities and the relation between them. Indeed, such approaches are much harder to fool by an entity searching to avoid tracking: it is simple for an entity to control its own behavior by generating deceiving, dummy transactions or by altering its normal behavior to mislead analysts. On the contrary, it is nearly impossible to control other actors' behavior.

Graph motifs

Graph motifs have been used to extract features from the neighbors of the ego network beyond the target node features. Graph motifs are a set of subgraphs describing the neighbors of a node and their connectivity patterns (Milo et al. 2002). In Bitcoin transaction networks, graph motif has been used to extract the features from neighbors (2-motif) and neighbors of neighbors (3-motif) of the address or entity flow networks. To classify entity types, Jourdan et al. (2018) extracted motif-based features (e.g., the total amount sent or received in an incoming/outgoing/loop motif). They showed that these motifs improved the accuracy compared with using only the target node's descriptive and centrality features. Zola et al. (2019) applied cascade machine learning to aggregate predictions of individual address classifiers and motif features to improve entity-level features. Nonetheless, these works do not use the identity of neighbors in the motifs but only numeric descriptions. Wu et al. (2022) propose a temporal motif approach to extract transaction patterns and use positive and unlabeled learning to detect mixing services.

Graph representation learning

Graph representation learning is a neural network-based approach that learns representation features from the entire graph. A few works have applied this approach

to embed addresses or entities in the Bitcoin transaction network and used them for downstream tasks. In Michalski et al. (2020), the authors searched to classify entities into exchange and miners. They encoded the entity position in the transaction network using *node2vec*, but later found that this information did not improve classification performance. However, Hu et al. (2019) showed that *node2vec* embeddings outperform statistical and network features to classify money laundering entities. Weber et al. (2019) applied *EvolveGCN*, which extends *Graph Convolutional Networks (GCN)* to include the temporal information to predict illegal transactions in the Elliptic dataset. *Label-GCN* was proposed by Bellei et al. (2021) to propagate the labels of neighbor transactions in GCN, applying to the same Elliptic dataset. Both models perform better than non-temporal GCN and are comparable to curated features from the dataset with a margin of less than 1%. These results show that adding temporal information can improve the model's accuracy. A recent work by Huang et al. (2022) used *Graph Feature Network (GFN)* to learn graph representations from address subgraphs and use those features to predict the address role. The authors showed that their model provided a very high accuracy on both tasks.

Our work applies graph representation learning techniques to identify money flows likely to belong to a same entity. This task is new in the literature and has never been addressed. Compared to the previously mentioned works that extract features from the entire graph, we use the money flow that tracks the transaction graph from the sources of interest. Moreover, our proposed method relies not on summarized activity features or only direct neighbors of a source node, but on the temporal network describing the whole flow of coins from each entity.

Methodology

Our taint flow representation learning pipeline consists of three steps as depicted in Fig. 2. First, we extract taint flows from an entity of interest, i.e., flows originating from an entity on a particular day. Second, we use random walk approaches on the extracted taint flow to generate inputs for the representation learning model. Third, we apply the representation learning method to assign a fingerprint vector summarizing its flow for each entity of interest. Lastly, the embedding results are subsequently utilized in downstream tasks. In this study, we leverage the fingerprint associated with each taint flow to classify distinct entities and cluster those with similar fingerprint patterns.

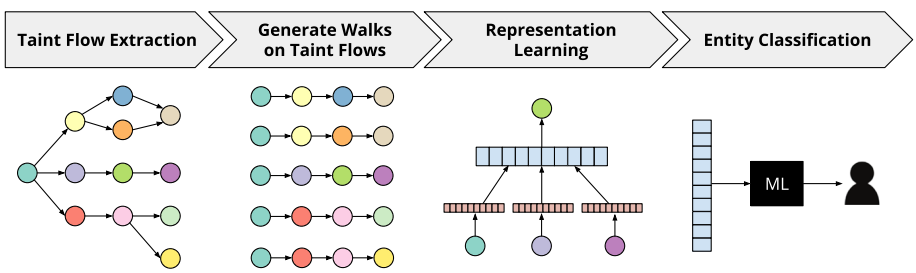

Fig. 2 Taint flow representation learning pipeline

Taint flow extraction

Our objective is to characterize an entity by its money flow. To define a flow, we first select an entity and a time period, short enough to consider that the entity behavior has not changed between the beginning and the end of that period—in this paper, a single day. To construct the money flow, we first select the transactions of the chosen entity during the chosen time period. Coins sent in these transactions are considered *tainted*. The entity receiving those coins will later use them to make a payment in a transaction, that we thus include in our taint flow. The process continues recursively until we consider that the taint is *dissolved*. Indeed, we can consider that the destination of our tainted coins is relevant to characterize the source entity only until a certain distance from the source. Furthermore, small amounts need to be combined with coins from other sources to be spent. We consider that each such event dissolves the taint, and we stop tracking tainted coins when the taint strength is below a threshold.

More formally, we defined *taint flow* as the *directed acyclic graph (DAG)* composed of all transactions involving *tainted* coins until *dissolution*. The dissolution is computed using the *purity* measure (ρ) (Di Battista et al. 2015), defined for a given UTXO as the percentage of its value that is tainted:

$$\rho(tx) = \frac{\sum_{e \in tx_{in}} \rho(e_{receive}) \cdot v(e)}{\sum_{e \in tx_{in}} v(e)} \tag{1}$$

The purity of a root transaction is 1 by definition. In this study, we set a purity threshold $\rho_{min} = 0.001$, meaning that a coin is considered dissolved when it is spent in a transaction with 1,000 times the amount of un-tainted coins. Therefore, the taint amount of the edge, annotated as $v_{taint}(e)$, is calculated as $v_{taint}(e) = v(e) \times \rho(tx)$. Furthermore, we incorporate a criterion to halt the tracking of the flow when a transaction is separated by more than 1 year from the source transactions (denoted as $t_{max} = 1$ year). This criterion effectively prevents the algorithm from tracing the money flow beyond a certain temporal distance from its origin.

Algorithm 1 describes the process of retrieving transaction outputs and adding them to the taint flow graph. Our algorithm applies *haircut tainting*, which assumes that the tainted money is divided equally to all output transactions in proportion to their value (Ahmed et al. (2019); Tironsakkul et al. (2019)). Since the entity of interest can have multiple transactions during the period of interest, we compute a taint flow from each output of each transaction and then combine them together in a single DAG, such as there is a single root node *root*, with outgoing edge to each of the individual flows corresponding to each transaction done by the entity during the period. It is this global flow that we synthesize in a fingerprint.

Input : $\tau_o = \{tx_0, tx_1, ..., tx_n\}$ is a set of transactions as a seeding node of the taint flow.
Input : ρ_{min} is a minimum purity threshold.
Input : t_{max} is a maximum time threshold.
Output: $\mathcal{E}$ is the edge list of the payout flow.
$\mathcal{Q} \leftarrow FIFO(\tau_o)$; // the tx queue to check the purity
$\mathcal{E} \leftarrow LIST()$; // the edge list of the taint flow
while $\mathcal{Q} \neq \emptyset$ **do**
  $tx \leftarrow \mathcal{Q}.pop()$; // Get a tx in the queue to check its purity
  if $\rho(tx) \geq \rho_{min}$ *and* $t(tx) \leq t_{max}$ **then**
    for e ***in*** tx_{out} **do**
      $v_{taint}(e) \leftarrow v(e) \times \rho(tx)$; // Calculate the taint amount of the out edge
      $\mathcal{E}.append(e)$; // Add the out edge to the taint flow
      $\mathcal{Q}.append(e_{spend})$; // Add the spent tx of this out edge to the queue
    end
  end
end

Algorithm 1: Extract a taint flow graph from source transactions. FIFO is a First In First Out data structure, and LIST is a standard ordered list data structure

Taint flow representation learning

Our objective is to identify that two taint flows—i.e., two directed acyclic temporal graphs—are similar and thus likely to have the same source. Our approach consists in synthesizing each taint flow into a *fingerprint* vector, capturing various aspects: its topology, temporal aspects, and labels of encountered nodes.

Graph embedding is a well-adapted approach to deal with this problem. However, because our graphs are DAGs, we cannot use off-the-shelf methods such as *graph-2vec* (Narayanan et al. 2017) or *anonymous walk embedding* (Ivanov and Burnaev 2018). Instead, we define an adapted way to do random walks and then apply a mechanism similar to *graph2vec* to encode the representative feature of graphs into vectors. Another difficulty in applying graph embedding to our setting is that one needs to define how encountered nodes are recorded in the random walk, e.g., by their identity or by one of their property.

Generate the walk sequences from the taint flow

For each taint flow graph, we generate 10,000 random walks from the source to dissolution in order to extract the substructure pattern. We experimented with (1) different random walks on the taint flows and (2) pruning strategies to represent the walk in the representation model.

Taint flow walk methods In our previous work (Tovanich and Cazabet 2023), we generated the walks on taint flows using the shortest path and random walk strategies.

- *Shortest path walk* We randomly choose leaf nodes (i.e., the transaction before the flow is dissolved) and calculate the shortest path from the source to the leaf node.
- *Random walk* We randomly walk from the source node to the next payout transaction without considering the weight until we encounter a dissolved node.

In this work, we test several variants of random walks:

- *Biased random walk* We bias the walk probability using the *node2vec* approach (Grover and Leskovec 2016). In practice, we consider the graph as undirected and start the walks from the source. We set the parameters $p = 2$ and $q = 0.5$, which respectively define the inverse of the bias to go back and to move forward. Compared with previous strategies, this method allows the walk to go back in time and explore in a more Breadth-first search approach.
- *Weighted random walk* The probability of moving to the next payout transaction is proportional to its amount. Let $\mathcal{E}$ be the list of out edges (tx_{out}) from a transaction node tx, we define the probability that the walk from tx will follow the edge e as:

$$P(i) = \frac{v(i)}{\sum_{j \in \mathcal{E}} v(j)} \tag{2}$$

- *Continuous-time dynamic network embeddings (CTDNE)* based approaches (Qu et al. 2020) bias the random walk based on the time difference between transactions. The temporal walk is more likely to select edges of shorter time distance.

  - *Linear CTDNE* the out edges tx_{out} of a transaction tx, are first sorted according to their spending time $t(e_{spend})$ and assigned a rank from first to last UTXO spent. We assign a higher rank to the first UTXO spent and the lowest rank to the last one. The probability for an out edge i of being selected in the next step of the walk is calculated as:

$$P(i) = \frac{rank(t(tx) - t(i_{spend}))}{\sum_{j \in \mathcal{E}} rank(t(tx) - t(j_{spend}))} \tag{3}$$

  - *Exponential CTDNE* We use the exponential of the time difference between the transaction $t(tx)$ and each out edge spent time $t(e_{spend})$. The probability of the out edge being picked in the walk exponentially decays from the first to the last UTXO spent from the transaction. We calculate the probability that the walk will move from a transaction tx to the next edge i as:

$$P(i) = \frac{exp(t(tx) - t(i_{spend}))}{\sum_{j \in \mathcal{E}} exp(t(tx) - t(j_{spend}))} \tag{4}$$

Walk pruning strategies Graph embedding methods rely on the probability of encountering a given code in a walk. To work efficiently, codes need to be frequent enough. Using cluster ids directly as codes is not appropriate due to the large number of clusters seen only once or a few times. We thus prune walks to keep only information about some of the clusters encountered. We compare two pruning strategies, using either frequent clusters or known entities of clusters from an external data source.

- *Frequent clusters* We count the number of occurrences of clusters in all flows and keep only, respectively, the 1%, 5%, 10%, and 20% most frequent ones.
- *Known entities* We extracted the list of clusters of known entities from the WalletExplorer dataset and pruned the walk to preserve only known entity names.

Another variant consists in using as code the category of the entity instead of its name (i.e., exchange, gambling, market, mining pools, mixer, service, and wallet).

For each walk, any occurrence of the source cluster are replaced with a "*black*" label to prevent the model from learning the embedding from it.

Sequence and temporal patterns We tested two ways of encoding the succession of events—sequential and temporal—to input into the representation model.

- *Sequential pattern* We collect the sequence of clusters from the source until dissolution, keeping the sequence but ignoring time. For example: root → black → 107752768 → 103850367 → 104514539 → 106936169 → 98856802 → black → 107752768 → 106936169 → 100364814 → dissolved
- *Temporal pattern* We incorporate the time difference into the code. As the transaction activities tend to be in the early days after the source, we use the Fibonacci sequence to group days into bins (i.e., 0, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377). For example: root → (0, 1] black → (0, 1] 107752768 → (0, 1] 103850367 → (1, 2] 104514539 → (1, 2] 106936169 → (1, 2] 98856802 → (1, 2] black → (1, 2] 107752768 → (1, 2] 106936169 → (1, 2] 100364814 → (3, 5] dissolved

Training the representation learning model

As with other graph embedding methods, the walks we extract from a taint flow can be understood as a sentence in a text document. Instead of a sequence of words, the walk sentence consists of a sequence of node codes, from the source until dissolution. For each taint flow, we concatenate all walk sentences into a document and tag them with the original source (i.e., entity and starting date) of the flow. With this representation, we can feed documents as inputs to the representation learning model to extract the embedding vector for each flow.

In this work, we chose the *Distributed Memory Model of Paragraph Vectors (PV-DM)* to train the embedding of taint flows. It is one of the two variants of the document embedding model in the *doc2vec* article (Le and Mikolov 2014). We chose the PV-DM model because it preserves the order sequence of the walk rather than predicting a bag of words in a sentence in the PV-DBOW model. The objective of the PV-DM model is to train the neural network to predict the middle word (in our case, cluster) with the context words within a specific window size. The average/concatenate layer can be used to derive the representation vector of the document (in our case, taint flow).

In the PV-DM model, we initially set the window size to 5 and represented each taint flow with a 256-dimensional embedding vector. These representations can be used for exploratory analysis and downstream tasks.

The graph embedding approach thus yields one 256-dimensional vector for each taint flow, capturing its structure, that we use as the fingerprint of that taint flow. In this study, we explore the applicability of employing these fingerprints for downstream machine learning tasks, specifically for the classification of distinct entities and the clustering of entities that exhibit similar taint flow patterns.

Entity classification task

Deanonymizing entities is one of the main research challenges in the Bitcoin blockchain. Most of the past work focused on the classification of illegal transactions or the role of entities (i.e., entity type). In this work, we introduce a new entity classification task to identify if the taint flow originated from different pseudonymous addresses or clusters belonging to the same entity. We use the representation vectors from taint flows as the feature to train the entity classification task.

We compare three classification models: *k-nearest neighbors (k-NN)*, *support vector machine (SVM)*, and *random forest (RF)* on the taint flow fingerprints with different configurations. To evaluate the performance of the models, we performed *k-fold cross-validation*, in which k is the maximum number of flows we sampled for each entity in the dataset. For each test set in the cross-validation, we left at least one flow for each entity to evaluate the classification accuracy. This strategy is similar to the *leave one out* approach but keeps one flow of each entity in each fold.

Cluster analysis task

To characterize different taint flow patterns, we adopt *k-means* cluster analysis to form groups of taint flows as part of an unsupervised learning approach. The *k-means* algorithm groups the taint flow embeddings into k clusters, aiming to minimize the embedding distance between flows within the same cluster while maximizing the distance between clusters. Consequently, the resulting clusters should represent groups of taint flows sharing similar fingerprint characteristics.

We execute a k-means algorithm with the number of clusters k from 1 to 100 and evaluate the clustering assignments to their true entities. To measure the quality of clusters, we rely on three clustering performance metrics: *Normalized Mutual Information (NMI)*, *Adjusted Rand Index (ARI)*, and *Adjusted Mutual Information (AMI)*. The higher value for each metric (with a maximum value of 1) indicates that the clusters are identical to their true entities.

Results

We applied our fingerprint approach to three address-entity tagging datasets to evaluate entity classification and clustering tasks.

Datasets

We extracted taint flows from three address-entity tagging datasets representing different types of entities in Bitcoin: mining pools, ransomware families, and well-known Bitcoin entities.

(1) *Mining pools* are the consortium of miners competing to verify transactions in the blockchain and sharing monetary rewards among miners (Tovanich et al. 2022). We extracted 50 taint flows from the top-15 mining pools between 2013 and 2016. For each mining pool, we randomly selected 50 random dates, spanned equally across four years, and obtained all rewards (i.e., coinbase transactions) from the

blockchain on that day as the seeding transactions. As a result, we obtained 750 flows from 15 mining pools.

(2) *BitcoinHeist* dataset contains a list of addresses associated with 24 ransomware families (Akcora et al. 2020). We selected the top 14 more active ransomwares, plus one non-ransomware from the original dataset (white addresses). We randomly sampled 50 address-date pairs from each to construct taint flows, appart from 3 of the ransomwares that have less than 50 flows (34, 32, and 27, respectively). We obtained a total of 693 flows from 15 entities (14 ransomware families and 1 group of white addresses).

(3) *WalletExplorer* is a public address tagging dataset of well-known entities in the Bitcoin network updated until 2016 (Janda 2023). We selected the top-6 entities with the highest number of addresses for each entity type: service, exchange, gambling, wallet, and marketplaces, together with three mixing services. For each selected entity, we built 100 taint flows from different addresses and starting dates. We have in total of 3,300 flows from 33 entities.

Entity classification task

For each dataset, we apply our fingerprint embedding method and assess the performance of the classification models in distinguishing unique entities. First, we systematically compared our approach with two baseline models on classification models (Experiment 1). We also investigate the effectiveness of different pruning strategies tailored to our approach (Experiment 2). Additionally, we assess the performance of various walk-based techniques in terms of classification accuracy (Experiment 3). Finally, we conduct experiments to determine whether the window size setting in the PV-DM model (Experiment 4) and the maximum path length in the taint flows have an impact on model accuracy (Experiment 5).

Definition of baseline models

We define two baseline models to compare with our taint flow representation learning on the entity classification task.

(1) *Graph2Vec* learns the whole-graph representation from the structure of the graph (Narayanan et al. 2017). The method builds graph kernels from *Weisfeiler-Lehman Hashing* and then employs the *skip-gram* model to generate the graph embedding. We built the static entity network from the taint flow and applied *graph2vec* to study the relevance of graph structure in classifying taint flows from different entities.

(2) *Singular value decomposition (SVD)* is a matrix factorization technique that decomposes a matrix into three metrics that minimize the reconstruction error. We built the matrix recording the frequent clusters encountered in the taint flow without considering the sequence of clusters in the flow. We applied SVD to extract a low-dimensional representation of the frequent cluster matrix for each flow.

We apply these baseline models only with the top 1% frequent clusters due to scalability issues given the size of the data. Results show that these methods are clearly outperformed, and thus we did not work on scaling them on larger experiments.

Experiment 1: evaluation of baseline models and walk pruning strategies

First, we fix the embedding parameters and compare the classification results to the baseline methods with the simplest walk strategy, the *random walk* approach. We compare all three classification approaches (k-NN, SVM, and Random Forest) and all pruning strategies. Results are shown in Table 1 and graphically in Fig. 3 with 95% confidence intervals.

Among the three classification models, SVM consistently provides the best result compared to k-NN and random forest for random walk approaches. Random Forests yield the best results for the SVD baseline. Our representation learning method outperforms all baseline models on the same top 1% frequent clusters, showing 20%, 11%, and 23% accuracy improvement for each dataset.

Experiment 2: filtering clusters with pruning strategies

Following that, we evaluate the effectiveness of various pruning strategies aimed at optimizing classification accuracy. As shown in Table 1 and Fig. 3, our results highlight that the top 5% frequent clusters provide the best accuracy on BitcoinHeist and WalletExplorer, while the top 1% offers the best result for the Mining Pools dataset. Pruning the walks with known entities and types gives inferior results to filters with frequent clusters. For different thresholds of top clusters pruning (i.e., the top 1%, 5%, 10%, and 20% frequent clusters), Fig. 3 shows that the accuracy scores are often not statistically different considering the 95% confident intervals.

Overall, using our proposed representation learning method on taint flows, we reached the best accuracy scores of 96% for Mining Pools, 70% for Bitcoin Heist, and 62% for WalletExplorer. These scores can be considered high, given that the task is not binary. Each dataset has respectively 15, 15, and 33 classes, and thus the outcome expected by a random classifier would be respectively (approximately) 6.6%, 6.6%, and 3%.

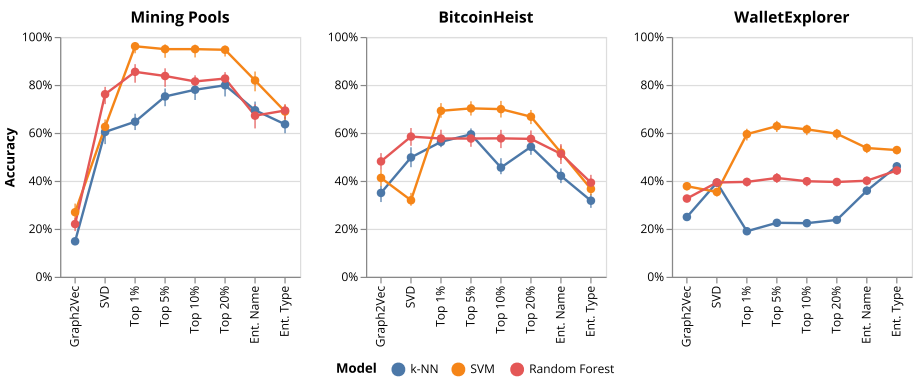

Fig. 3 The average test accuracy of classification models for baseline models and different walk pruning methods on random walks with 95% bootstrap confidence intervals

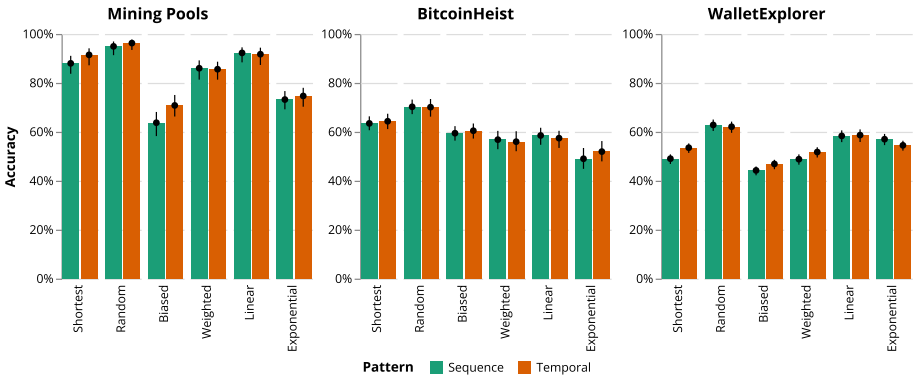

Fig. 4 The average test accuracy of SVM classifiers for different walk and patterns on the top 5% frequent clusters with 95% bootstrap confidence intervals

Experiment 3: evaluation of taint flow walk methods

Based on the previous experiments, we fixed the best settings overall—5% most frequent clusters and SVM classification—and varied the walk strategies. We report accuracy, F1-measure, and AUROC score for each walk and dataset in Table 2. In addition, Fig. 4 depicts the average accuracy for each walk with the 95% confidence intervals.

The result shows that the random walk approach consistently provides the best result to extract walks from taint flow in all datasets. It outperforms the shortest path walk and other variants of random walks (i.e., weighted and biased random walks). Surprisingly, continuous-time temporal walks like linear and exponential CTDNEs cannot achieve a better score than random walks but still provide better evaluation metrics than the rest in BitcoinHeist and WalletExplorer datasets. Linear CTDNE tends to give better accuracy than exponential CTDNE, which can be due to the Fibonacci time-encoding strategy.

When comparing labels including or not temporal information (Sequence VS Temporal), the results show that adding the temporal feature slightly increases the accuracy in Mining Pools and WalletExplorer datasets. The same effect also applies to most of the walks. Nonetheless, we found a more significant improvement in accuracy for the shortest path and biased random walks. Overall, adding the temporal information does not significantly improve nor impair the results when considering the best approach (random walk).

Experiment 4: effect of window sizes on representation learning

We further fine-tuned the PV-DM representation model by varying its window sizes from 3 to 9. The window size indicates how many surrounding contexts in the walk they consider to train the concatenate layer in the PV-DM model. Figure 5 reports the accuracy of entity classification models for each window size. Our experiment shows that the accuracy is indifferent when we increase or decrease the window size for all datasets. Nonetheless, in these additional experiments, we see a slight advantage of adding temporal information across window sizes, particularly on the WalletExplorer dataset.

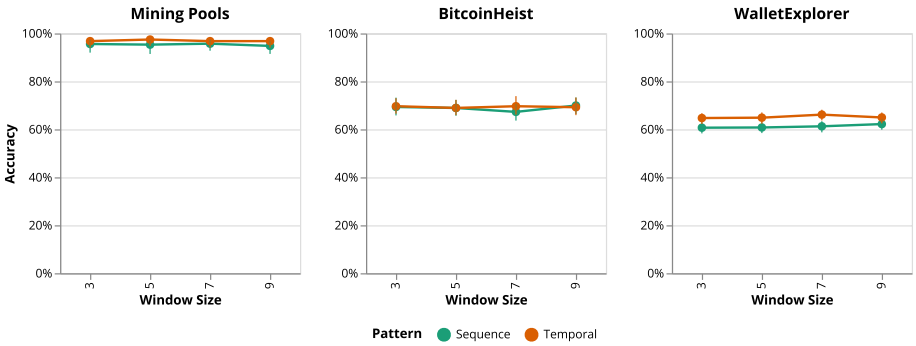

Fig. 5 The average test accuracy of SVM classifiers for different patterns on the top 5% frequent clusters with different window sizes

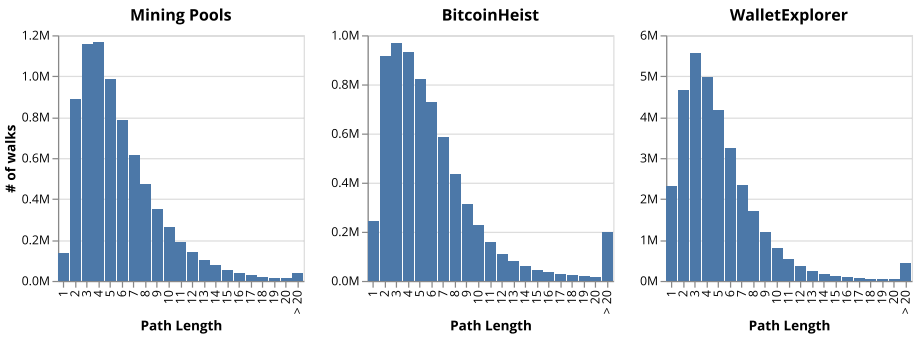

Fig. 6 Distributions of path length for random walks in each dataset

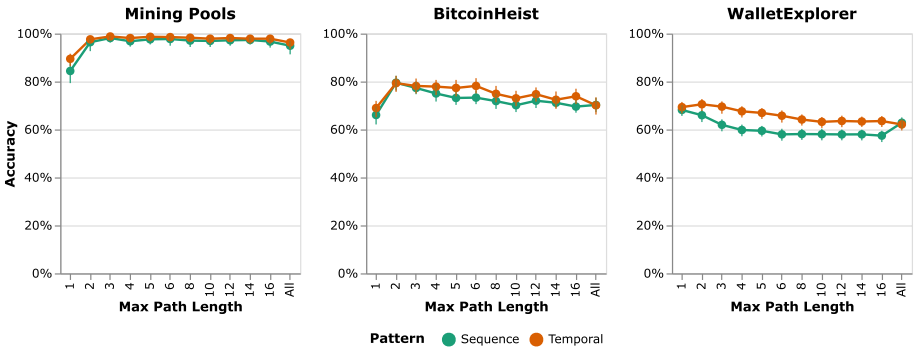

Fig. 7 The average test accuracy of SVM classifiers for the top 5% frequent clusters on random walks with different maximum path lengths

Experiment 5: effect of taint flow sizes on model accuracy

Finally, we analyze whether the size of taint flows affects the entity classification model accuracy. Figure 6 shows the distributions of path lengths on random walks for each dataset. The path lengths from the source until the dissolution is mostly short, with long tails observed. The average path lengths are 5.76, 6.47, and 5.30, while the medians are 5, 5, and 4, respectively, for Mining Pools, BitcoinHeist, and WalletExplorer datasets.

We varied the maximum path length from 1 to 16. Based on the random walks, we filtered the full walk up to the maximum path length for each flow and pruned the walk with the top 5% frequent clusters. For each maximum path length, we trained the SVM

model and reported the accuracy to determine which maximum path length yielded the best result, as depicted in Fig. 7.

Results show that we can accurately predict the source of the flow with only a short path length. For all datasets, the accuracy sharply increases from length 1 to length 2, then remains the same or slowly decreases at longer lengths. The result implies that, although considering only direct neighbors of the entity is not enough to deanonymize them, tracking flows beyond the second neighbors is not a rewarding strategy. Moreover, when using short taint flows, the temporal approach seems more valuable than when using the full flows, particularly for the WalletExplorer dataset.

Overall, the result informs us that walking only two or three steps from the source is sufficient to identify the entity of the flow with high accuracy. It explains why the past works produced decent classification results—although on different tasks—using only graph motif features at length two. We suspect that the full walks of taint flows may include the flow of clusters from longer path lengths beyond relevant information to classify entities. In our taint flow extraction algorithm, the dissolution criteria to stop crawling the taint flow is quite generous (purity < 1/1000 and > 1 year apart from the source). The appropriate parameters to crawl the taint flow should be an interesting question to study in future work.

Visualization of taint flow embeddings

We explore the embedding results of taint flows for each dataset with the best accuracy model, i.e., temporal pattern with length 3 for Mining Pools, sequence pattern with length 2 for BitcoinHeist, and temporal pattern with length 2 for WalletExplorer. We use *t-distributed stochastic neighbor embedding (t-SNE)* (van der Maaten and Hinton 2008) to project the embedding result into 2-dimension to display on scatter plots.

Figure 8 shows the t-SNE projection of mining pools. Most mining pools (in different colors) look clearly separate from each other and form their own clusters, probably because mining pools have a pattern of sharing rewards with their members

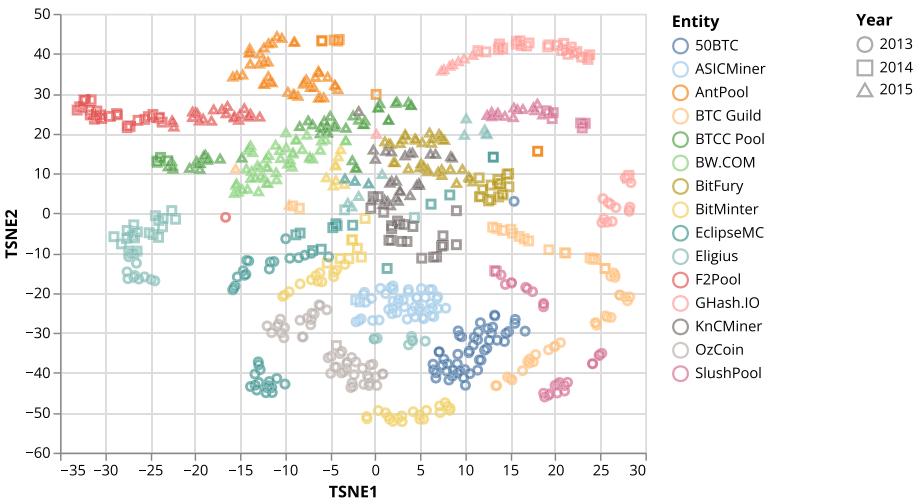

Fig. 8 T-SNE visualization of Mining Pools dataset on the temporal pattern with the maximum length of 3 embedding

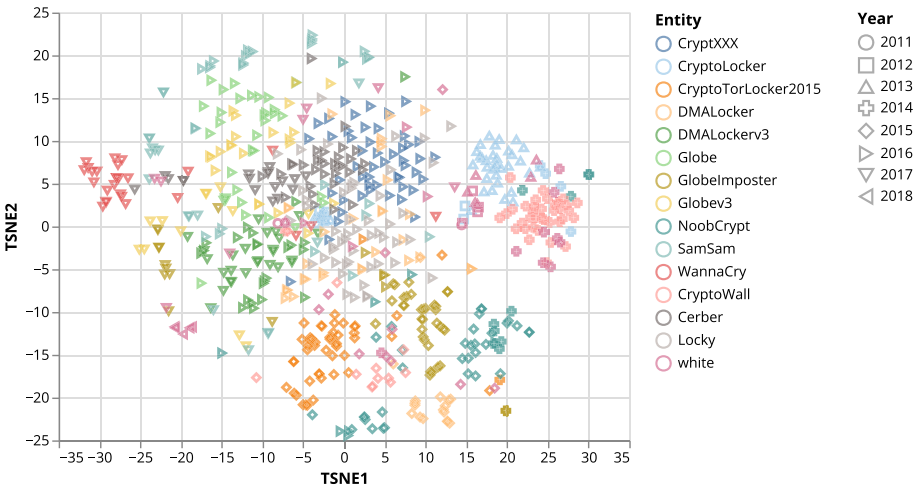

Fig. 9 T-SNE visualization of BitcoinHeist dataset on the sequential pattern with the maximum length of 2 embedding

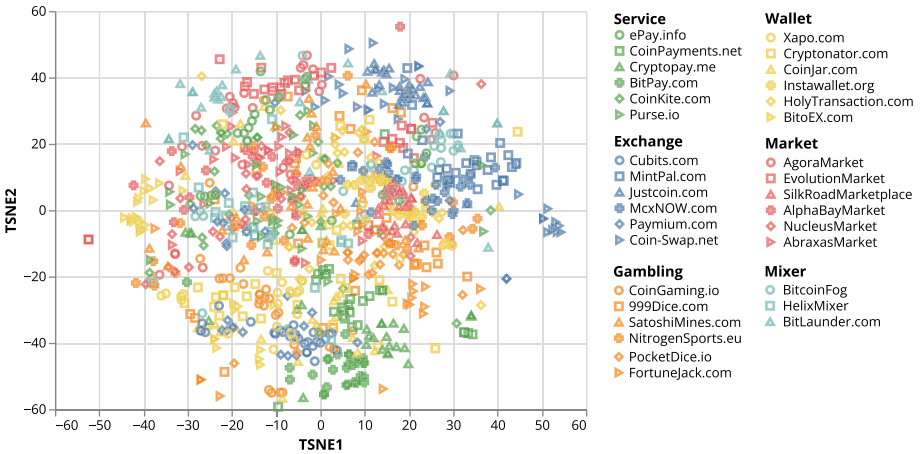

Fig. 10 T-SNE Visualization of WalletExplorer dataset on the temporal pattern with the maximum length of 2 embedding

regularly. We also observe the shift of t-SNE projection within the pool cluster on the payout flows over the years (in different shapes). This result indicates that the flow of mining pools could be regularly changed from new miners joining the pool or current miners changing to newer pools to increase their reward.

The t-SNE projection of Bitcoin heist datasets in Fig. 9 shows more overlap among ransomware families. Some ransomware families are clearly separable from the rest (e.g., CryptoLocker, CryptoTorLocker2015, CroptoWall, and Wannacry), while others are more overlapped with each other (e.g., CryptXXX, Cerber, and Locky). We also observe the same tendency of temporally close sources to be close in the embedding. We suspect that the representation learning model probably learns from the common clusters and group taint flows originated from the same year closer in the embeddings.

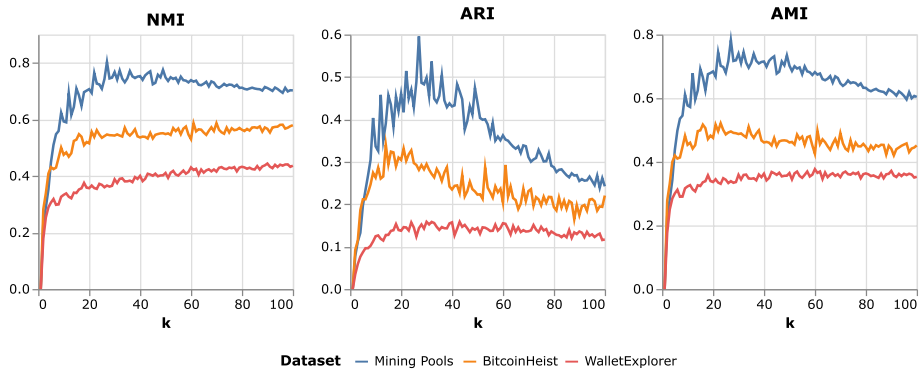

Fig. 11 Performance evaluation of k-means clustering on taint flow embeddings with *k* from 1 to 100

Table 1 Entity classification accuracy for baseline models and different pruning methods on random walks on three classification models: k-nearest neighbors (k-NN), support vector machine (SVM), and random forest (RF)

	Mining Pools			BitcoinHeist			WalletExplorer		
	k-NN	**SVM**	**RF**	**k-NN**	**SVM**	**RF**	**k-NN**	**SVM**	**RF**
Baseline (Freq. 1% Clusters)									
Graph2Vec	0.1480	0.2693	0.2200	0.3497	0.4124	0.4813	0.2494	0.3779	0.3264
SVD	0.6040	0.6253	*0.7613*	0.4977	0.3191	*0.5846*	0.3924	0.3536	*0.3933*
Random Walk									
Freq. 1% clusters	0.6467	**0.9613**	0.8547	0.5627	0.6921	0.5766	0.1900	0.5948	0.3958
Freq. 5% clusters	0.7520	0.9493	0.8373	0.5941	**0.7023**	0.5766	0.2252	**0.6282**	0.4121
Freq. 10% clusters	0.7800	0.9493	0.8147	0.4555	0.6992	0.5773	0.2236	0.6148	0.3985
Freq. 20% clusters	0.7987	0.9467	0.8267	0.5420	0.6676	0.5752	0.2373	0.5967	0.3955
Known entity name	0.6947	0.8187	0.6720	0.4210	0.5178	0.5126	0.3591	0.5370	0.4006
Known entity type	0.6360	0.6893	0.6933	0.3173	0.3664	0.3926	0.4606	0.5285	0.4433

Bold highlights the highest score overall, and *itatic* highlights the highest baseline score for each dataset

For the WalletExplorer dataset, we randomly sample 30% of the flows for each entity to show in Fig. 10. We use the color to encode entity types and the shape to distinguish entities in that type. The t-SNE projection shows more overlapping of flow embeddings, but we can observe that taint flows from the same entity are more likely to be closer. This result indicates that the same entity tends to have the same money flow pattern. We observe that some entity types, like Exchange entities, tend to be in the same projection region. In contrast, Market and Wallet entities are more scattered and mixed with other flows from other entity types in the visualization. Note that the 2-dimensional projection provided by t-SNE necessarily loses information compared with original taint flow fingerprints; thus, these representations are only included to facilitate the interpretation of the embedding results.

Cluster analysis task

Another Bitcoin fingerprint task is to characterize the flows sharing identical fingerprint patterns. To achieve this, we employ k-means clustering and subsequently evaluate the results by comparing them against the actual entities using three distinct

Table 2 Entity classification accuracy, F1-measure, and AUROC of SVM models for the top 5% frequent cluster embeddings with different walks and patterns

	Mining Pools			BitcoinHeist			WalletExplorer		
	Accuracy	**F1**	**AUROC**	**Accuracy**	**F1**	**AUROC**	**Accuracy**	**F1**	**AUROC**
Baseline (Freq. 1% Clusters)									
Graph2Vec	0.2693	0.2168	0.8011	*0.4124*	*0.3364*	*0.8931*	*0.3779*	*0.3054*	0.8968
SVD	*0.6253*	*0.5576*	*0.9556*	0.3191	0.2706	0.8163	0.3536	0.2785	*0.8992*
Sequential Pattern (Freq. 5% Clusters)									
Shortest Path Walk	0.8800	0.8506	0.9894	0.6346	0.5909	0.9399	0.4903	0.4171	0.9293
Random Walk	**0.9493**	**0.9372**	**0.9967**	**0.7023**	**0.6652**	**0.9541**	**0.6282**	**0.5632**	**0.9583**
Biased Random Walk	0.6373	0.5769	0.9501	0.5945	0.5604	0.9283	0.4424	0.3671	0.9094
Weighted Random Walk	0.8600	0.8289	0.9855	0.5681	0.5230	0.9128	0.4882	0.4154	0.9213
Linear CTDNE	0.9227	0.9012	0.9942	0.5852	0.5356	0.9342	0.5833	0.5149	0.9506
Exponential CTDNE	0.7320	0.6752	0.9620	0.4900	0.4463	0.8889	0.5700	0.4997	0.9375
Temporal Pattern (Freq. 5% Clusters)									
Shortest Path Walk	0.9147	0.8929	0.9918	0.6433	0.5980	0.9460	0.5352	0.4624	0.9423
Random Walk	**0.9627**	**0.9521**	**0.9974**	**0.7009**	**0.6687**	**0.9659**	**0.6209**	**0.5535**	**0.9611**
Biased Random Walk	0.7080	0.6537	0.9632	0.6046	0.5733	0.9365	0.4688	0.3957	0.9245
Weighted Random Walk	0.8560	0.8225	0.9815	0.5598	0.5138	0.9209	0.5176	0.4463	0.9312
Linear CTDNE	0.9173	0.8970	0.9907	0.5742	0.5263	0.9205	0.5867	0.5196	0.9516
Exponential CTDNE	0.7467	0.6826	0.9707	0.5193	0.4736	0.8973	0.5448	0.4728	0.9350

Bold highlights the highest score overall for each pattern and dataset. Baseline results are provided for reference in which the *italic* indicates the highest baseline score

measures. Figure 11 depicts The clustering performance metrics (NMI, ARI, and AMI) for varying values of k, ranging from 1 to 100 clusters.

The Mining Pools dataset scores higher for all measures, followed by BitcoinHeist. WalletExplorer clustering has the lowest score across the range of k clusters in all measures, probably because a cluster may include many entities that are overlapped to the embedding space. The clustering metrics align with the result from classification accuracy and t-SNE visualizations. Mining Pools have a clear taint flow pattern to distinguish mining pools, while the pattern of entities may not be apparent or unique in other datasets.

For all three datasets, NMI and AMI scores tend to increase rapidly from 1 to around 20–30 clusters before becoming more stable at the higher k clusters. The ARI, for all datasets, tends to provide lower scores than other metrics and decreases with a higher number of clusters. The number of clusters that maximize the ARI score is 27, 14, and 30 for Mining Pools, BitcoinHeist, and WalletExplorer, respectively. The number is close to the number of entities we have for each dataset, except for mining pools, which are closer to two times the number of entities. According to these results, cluster analysis can be a good indicator of the number of entities present in a set of taint flow sources. However, the lower scores in some datasets imply that the cluster of flows may not be a reliable approach to deanonymize entities based only on their taint flow fingerprints.

Conclusion

In this article, we proposed a method to fingerprint Bitcoin entities using taint flow embedding. Our assumption is that an entity, even though it uses multiple clusters of addresses, has a unique fingerprint on how it spends the money and how the money flows from the source transactions. In contrast to previous works that focused on classifying the entity's role or deciding whether the transaction is illegal or not, we introduced an entity classification task to deanonymize entities from their taint flow fingerprints.

Using the PV-DM representation learning model, we extracted the embedding features from taint flows of different entities in different time periods. We extracted the substructure patterns of the flow and experimented with different graph walking methods and cluster pruning strategies. We tested our approach on three known entities datasets. The classification accuracy is highest for the Mining Pools dataset, followed by BitcoinHeist and WalletExplorer. The evaluation shows that random walk patterns on taint flows with the top 5% of frequent cluster pruning provide the best classification accuracy among different substructure pattern configurations. In addition, adding temporal information to the embedding model can help improve accuracy, especially on the WalletExplorer dataset.

Furthermore, we investigated the effect of taint flow size on the model performance. To our surprise, using only two steps from the source transactions can provide a high, even better, accuracy than the full taint flows. The possible explanation is that our criteria for the dissolution are too generous and allow non-relevant clusters to be included in the taint flow graph. This finding opens a new research question on which criteria we should use to extract the meaningful taint flow graph to identify entities.

Our work provides evidence that using taint flows to fingerprint Bitcoin entities is a relevant approach. We think, however, that the results could be further enhanced by extracting taint flows using more appropriate stopping criteria so that no noise is included in it. In some cases, it might be relevant to track money flow over long distances—for instance, if the money stays in a close circle of contacts from the original source—, while sometimes the flow should stop early—for instance, when the entity sends money to mixers or exchange platforms, that will spend those coins irrespective of where it comes from. Another source of improvement would be to take into account the evolution of the Bitcoin ecosystem over time. As seen in t-SNE embeddings, our fingerprints simultaneously capture the source entity and the time period. Removing this time component from the fingerprints, in principle, could increase further the accuracy of the entity recognition task.

Abbreviations

AMI	Adjusted mutual information
ARI	Adjusted rand index
AUROC	Area under the receiver operating characteristic
CTDNE	Continuous-time dynamic network embeddings
DAG	Directed acyclic graph
GCN	Graph convolutional network
GFN	Graph feature network
k-NN	K-nearest neighbors
NMI	Normalized mutual information
PV-DM	Distributed memory model of paragraph vectors
RF	Random forest
SVD	Singular value decomposition
SVM	Support vector machine

t-SNE T-distributed Stochastic neighbor embedding
UTXO Unspent transaction output

Acknowledgements
Not applicable.

Author contributions
NT and RC wrote the manuscript text. NT and RC developed the methods. NT prepared the data and performed experiments. All authors reviewed the manuscript.

Funding
This project was partly founded by BITUNAM grant ANR-18-CE23-0004.

Availability of data and materials
The datasets used and analysed during the current study are available from the corresponding author on reasonable request.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 1 March 2023 Accepted: 7 September 2023
Published online: 15 September 2023

References

Ahmed M, Shumailov I, Anderson R (2019) Tendrils of crime: Visualizing the diffusion of stolen Bitcoins. In: Cybenko G, Pym D, Fila B (eds) Graphical Models for Security, pp 1–12. Springer, Cham. https://doi.org/10.1007/978-3-030-15465-3_1

Akcora C.G, Li Y, Gel Y.R, Kantarcioglu M (2020) BitcoinHeist: Topological data analysis for ransomware prediction on the Bitcoin blockchain. In: Bessiere, C. (ed.) Proceedings of the 29th international joint conference on artificial intelligence, pp. 4439–4445. https://doi.org/10.24963/ijcai.2020/612

Bartoletti M, Pes B, Serusi S (2018) Data mining for detecting Bitcoin ponzi schemes. In: 2018 Crypto valley conference on blockchain technology, pp. 75–84. https://doi.org/10.1109/CVCBT.2018.00014

Bartoletti M, Lande S, Loddo A, Pompianu L, Serusi S (2021) Cryptocurrency scams: analysis and perspectives. IEEE Access 9:148353–148373. https://doi.org/10.1109/ACCESS.2021.3123894

Bellei C, Alattas H, Kaaniche N (2021) Label-GCN: an effective method for adding label propagation to graph convolutional networks. https://doi.org/10.48550/ARXIV.2104.02153

Chainalysis Team (2022) The 2022 Crypto Crime Report. Chainalysis Inc., New York, NY, USA. Accessed 28 Feb 2022. https://go.chainalysis.com/2022-Crypto-Crime-Report.html

CoinMarketCap (2023): Bitcoin market price. Accessed 28 Feb. https://coinmarketcap.com/currencies/bitcoin/

Di Battista G, Di Donato V, Patrignani M, Pizzonia M, Roselli V, Tamassia R (2015) Bitconeview: visualization of flows in the Bitcoin transaction graph. In: 2015 IEEE symposium on visualization for cyber security, pp 1–8. https://doi.org/10.1109/VIZSEC.2015.7312773

Goldfeder S, Kalodner H, Reisman D, Narayanan A (2017) When the cookie meets the blockchain: Privacy risks of web payments via cryptocurrencies. https://doi.org/10.48550/ARXIV.1708.04748

Goldsmith D, Grauer K, Shmalo Y (2020) Analyzing hack subnetworks in the Bitcoin transaction graph. Appl Netw Sci 5(1):1–20. https://doi.org/10.1007/s41109-020-00261-7

Gomez G, Moreno-Sanchez P, Caballero J (2022) Watch your back: Identifying cybercrime financial relationships in Bitcoin through back-and-forth exploration. In: Proceedings of the 2022 ACM SIGSAC conference on computer and communications security, pp. 1291–1305. ACM, New York. https://doi.org/10.1145/3548606.3560587

Grover A, Leskovec J (2016) node2vec: Scalable feature learning for networks. In: Proceedings of the 22nd ACM SIGKDD international conference on knowledge discovery and data mining, pp. 855–864. ACM, New York. https://doi.org/10.1145/2939672.2939754

Harlev M.A, Sun Yin H, Langenheldt K.C, Mukkamala R, Vatrapu R (2018) Breaking bad: De-anonymising entity types on the Bitcoin blockchain using supervised machine learning. In: The 51st Hawaii international conference on system sciences. https://doi.org/10.24251/HICSS.2018.443

Harrigan M, Fretter C (2016) The unreasonable effectiveness of address clustering. In: 2016 Intl IEEE conferences on ubiquitous intelligence & computing, advanced and trusted computing, scalable computing and communications, cloud and big data computing, internet of people, and smart world congress, pp. 368–373. https://doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0071

Huang Z, Huang Y, Qian P, Chen J, He Q (2022) Demystifying Bitcoin address behavior via graph neural networks. https://doi.org/10.48550/ARXIV.2211.14582

Hu Y, Seneviratne S, Thilakarathna K, Fukuda K, Seneviratne A (2019) Characterizing and detecting money laundering activities on the Bitcoin network. https://doi.org/10.48550/ARXIV.1912.12060
Ivanov S, Burnaev E (2018) Anonymous walk embeddings. https://doi.org/10.48550/ARXIV.1805.11921
Janda A (2023) WalletExplorer.com. Accessed 28 Feb. https://www.walletexplorer.com/info
Jourdan M, Blandin S, Wynter L, Deshpande P (2018) Characterizing entities in the Bitcoin blockchain. In: IEEE international conference on data mining workshops, pp. 55–62. IEEE, Singapore. https://doi.org/10.1109/ICDMW.2018.00016
Kalodner H, Möser M, Lee K, Goldfeder S, Plattner M, Chator A, Narayanan A (2020) BlockSci: Design and applications of a blockchain analysis platform. In: Proceedings of the 29th USENIX conference on security symposium. USENIX Association, USA. https://doi.org/10.5555/3489212.3489365
Kondor D, Pósfai M, Csabai I, Vattay G (2014) Do the rich get richer? an empirical analysis of the Bitcoin transaction network. PLoS ONE 9(2):1–10. https://doi.org/10.1371/journal.pone.0086197
Le QV, Mikolov T (2014) Distributed representations of sentences and documents. https://doi.org/10.48550/ARXIV.1405.4053
Lin YJ, Wu P.-W, Hsu C.-H, Tu I.-P, Liao S.-w (2019) An evaluation of Bitcoin address classification based on transaction history summarization. In: 2019 IEEE international conference on blockchain and cryptocurrency, pp. 302–310. https://doi.org/10.1109/BLOC.2019.8751410
Lischke M, Fabian B (2016) Analyzing the Bitcoin network: the first four years. Future Internet, 8(1) https://doi.org/10.3390/fi8010007
Liu XF, Ren H-H, Liu S-H, Jiang X-J (2021) Characterizing key agents in the cryptocurrency economy through blockchain transaction analysis. EPJ Data Sci 10(1):21. https://doi.org/10.1140/epjds/s13688-021-00276-9
Maesa DDF, Marino A, Ricci L (2019) The bow tie structure of the Bitcoin users graph. Appl Netw Sci 4(1):1–22. https://doi.org/10.1007/s41109-019-0163-y
Meiklejohn S, Pomarole M, Jordan G, Levchenko K, McCoy D, Voelker GM, Savage S (2016) A fistful of Bitcoins: characterizing payments among men with no names. Commun ACM 59(4):86–93. https://doi.org/10.1145/2896384
Michalski R, Dziubałtowska D, Macek P (2020) Revealing the character of nodes in a blockchain with supervised learning. IEEE Access 8:109639–109647. https://doi.org/10.1109/ACCESS.2020.3001676
Milo R, Shen-Orr S, Itzkovitz S, Kashtan N, Chklovskii D, Alon U (2002) Network motifs: simple building blocks of complex networks. Science 298(5594):824–827. https://doi.org/10.1126/science.298.5594.824
Möser M, Narayanan A (2022) Resurrecting address clustering in Bitcoin. In: Eyal I, Garay J (eds) Financial Cryptography and Data Security, pp 386–403. Springer, Cham. https://doi.org/10.1007/978-3-031-18283-9_19
Nakamoto S (2008) Bitcoin: A peer-to-peer electronic cash system. Technical report. Accessed 28 Feb 2023. http://bitcoin.org/bitcoin.pdf
Narayanan A, Chandramohan M, Venkatesan R, Chen L, Liu Y, Jaiswal S (2017) graph2vec: learning distributed representations of graphs. https://doi.org/10.48550/ARXIV.1707.05005
Nerurkar P, Patel D, Busnel Y, Ludinard R, Kumari S, Khan MK (2021) Dissecting Bitcoin blockchain: empirical analysis of Bitcoin network (2009–2020). J Netw Comput Appl 177:102940. https://doi.org/10.1016/j.jnca.2020.102940
Qu L, Zhu H, Duan Q, Shi Y (2020) Continuous-time link prediction via temporal dependent graph neural network. In: Proceedings of the web conference 2020, pp. 3026–3032. ACM, New York. https://doi.org/10.1145/3366423.3380073
Ramos Tubino R, Robardet C, Cazabet R (2022) Towards a better identification of Bitcoin actors by supervised learning. Data Knowl Eng 142:102094. https://doi.org/10.1016/j.datak.2022.102094
Reid F, Harrigan M (2013) An analysis of anonymity in the Bitcoin system. In: Altshuler Y, Elovici Y, Cremers AB, Aharony N, Pentland, A. (eds.) Security and privacy in social networks, pp 197–223. Springer, New York. https://doi.org/10.1007/978-1-4614-4139-7_10
Cazabet R, Rym B, and Latapy M (2018) Tracking Bitcoin users activity using community detection on a network of weak signals. In: Cherifi, C., Cherifi, H., Karsai, M., Musolesi, M. (eds.) Complex networks & their applications VI, pp. 166–177. Springer, Cham. https://doi.org/10.1007/978-3-319-72150-7_14
Tironsakkul T, Maarek M, Eross A, Just M (2019) Probing the mystery of cryptocurrency theft: an investigation into methods for taint analysis. https://doi.org/10.48550/ARXIV.1906.05754
Tovanich N, Soulié N, Heulot N, Isenberg P (2022) The evolution of mining pools and miners's behaviors in the Bitcoin blockchain. IEEE Trans Netw Serv Manage 19(3):3633–3644. https://doi.org/10.1109/TNSM.2022.3159004
Tovanich N, Cazabet R (2023) Pattern analysis of money flows in the Bitcoin blockchain. In: Cherifi H, Mantegna RN, Rocha LM, Cherifi C, Micciché S (eds.) Complex Networks and Their Applications XI, pp. 443–455. Springer, Cham. https://doi.org/10.1007/978-3-031-21127-0_36
Vallarano N, Tessone CJ, Squartini T (2020) Bitcoin transaction networks: an overview of recent results. Front Phys. https://doi.org/10.3389/fphy.2020.00286
van der Maaten L, Hinton G (2008) Visualizing data using t-SNE. J Mach Learn Res 9(86):2579–2605
Weber M, Domeniconi G, Chen J, Weidele D.K.I, Bellei C, Robinson T, Leiserson C.E (2019) Anti-money laundering in Bitcoin: Experimenting with graph convolutional networks for financial forensics. In: KDD workshop on anomaly detection in finance. https://doi.org/10.48550/arXiv.1908.02591
Wu J, Liu J, Chen W, Huang H, Zheng Z, Zhang Y (2022) Detecting mixing services via mining Bitcoin transaction network with hybrid motifs. IEEE Trans Syst Man Cybern Syst 52(4):2237–2249. https://doi.org/10.1109/TSMC.2021.3049278
Xiang Y, Lei Y, Bao D, Ren W, Li T, Yang Q, Liu W, Zhu T, Choo KKR (2022) BABD: a Bitcoin address behavior dataset for pattern analysis. https://doi.org/10.48550/ARXIV.2204.05746
Zola F, Eguimendia M, Bruse J.L, Urrutia R.O (2019) Cascading machine learning to attack Bitcoin anonymity. In: IEEE international conference on blockchain, pp. 10–17. IEEE, Atlanta. https://doi.org/10.1109/Blockchain.2019.00011

Publisher's Note