Applied Soft Computing 109 (2021) 107507

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

AWAP: Adaptive weighted attribute propagation enhanced community detection model for bitcoin de-anonymization

Xie Xueshuo [a,1], Wang Jiming [b,1], Ye Junyi [c], Fang Yaozheng [b], Lu Ye [b], Li Tao [a,b,*], Wang Guiling [c]

[a] College of Computer Science, Nankai university, Tianjin, 300350, China
[b] Tianjin Key Laboratory of Network and Data Security Technology, Tianjin, 300350, China
[c] New Jersey Institute of Technology, Newark, NJ, 07102, USA

ARTICLE INFO

Article history:
Received 31 October 2020
Received in revised form 12 January 2021
Accepted 6 May 2021
Available online 21 May 2021

Keywords:
Bitcoin anonymity
Community detection
Attribute propagation
Feature engineering

ABSTRACT

Bitcoin is a kind of decentralized cryptocurrency and widely used in online payment partially for its anonymity mechanism. The anonymity, however, also attracts the usage of cryptocurrency by criminals in ransomware and money laundering, and limits its further application and development. In this paper, we aim to improve Bitcoin's auditability with de-anonymization. Many previous studies have used heuristic clustering or supervised machine learning to analyze the historical transactions for identifying user behaviors. However, heuristic clustering only considers the topological structure of the transaction graph and ignores the transaction attributes. While supervised learning is usually limited by the size of labeled datasets, resulting in an unsatisfactory *accuracy*. To resolve the above problems, we propose an Adaptive Weighted Attribute Propagation enhanced community detection model, named AWAP, which considers both the transaction's topological structure and the transaction attributes. We first parse the transaction data from the public ledger and construct a bipartite graph to describe correlations between addresses and transactions. Then, we use a five-step feature engineering pipeline to extract Bitcoin address attributes and build an attribute graph. Finally, we design an adaptive weighted attribute propagation algorithm running on the attribute graph to classify the Bitcoin addresses and identify user behaviors. Extensive experiments highlight that AWAP model achieves 12% higher *accuracy* and 25% higher *F-score* on average, compared to the state-of-the-art Bitcoin address classifiers and other community detection algorithms. To evaluate the effectiveness of AWAP, we also present two case studies on Bitcoin address classification and Bitcoin trace-ability in ransomware.

1. Introduction

Nowadays, cryptocurrency has become a buzzword in both industry and academia. Bitcoin, as the largest and most popular cryptocurrencies, is proposed by Satoshi Nakamoto in 2008 [1]. As a global decentralized cryptocurrency, Bitcoin has received extensive attention because of its anonymity and security. In practice, users can register a Bitcoin account without any real-world identity. The Bitcoin system generates a pseudonym, such as a hash value or a public key to identify the user. The anonymity mechanism ensures that the real identity of a trader is not exposed to the real world. This property attracts a large number of users to join the Bitcoin community. However, the anonymity also makes Bitcoin a tool of circulating currency in some illegal activities, which limits its applications due to the lack of regulation. Recently, Bitcoin has been abused in ransomware, thefts, and scams [2,3], such as the well known black market Silk Road [4]. From the perspective of regulatory purposes, it is crucial to understand the anonymity of the Bitcoin system and detect criminal transactions.

Therefore, a healthy cryptocurrency ecosystem should (1) support technical legal investigations to ensure the safety and legality of transactions and (2) provide sufficient anonymity to protect user privacy [5]. In this paper, we mainly study the following two questions: (1) How much anonymity does the Bitcoin system provide? (2) Can we obtain user behaviors by analyzing their historical transactions?

In the Bitcoin ecosystem, a transaction is a cryptocurrency transfer record between addresses. The transaction contains Bitcoin addresses and some other information. All the transactions

* Corresponding author at: College of Computer Science, Nankai university, Tianjin, 300350, China.
E-mail address: litao@nankai.edu.cn (L. Tao).

[1] These authors contributed equally to this work and should be considered co-first authors.

https://doi.org/10.1016/j.asoc.2021.107507
1568-4946/

are recorded on the public ledgers to assure their validity. Therefore, anyone who join the Bitcoin system is able to obtain and analyze these large amounts of transaction records. With these public transactions, we are able to construct a transaction graph to track user transactions and further analyze the Bitcoin user behaviors. Heuristic clustering has been proposed to build a mapping between the Bitcoin addresses and the Bitcoin users by studying the transaction graph [6,7]. This mapping improves transaction traceability and statistical feature analysis. However, it only considers the relationship between Bitcoin addresses, ignoring many other transaction information, such as Bitcoin transaction type, value, and time. Recently, some supervised learning-based methods [8] are proposed to classify Bitcoin addresses. However, these methods are limited by the size of the labeled dataset. Therefore it is not suitable for large-scale analysis of historical transactions with limited labels.

Furthermore, community detection based approaches are widely used to study the relationship among nodes in a graph. The goal of community detection is to group closely connected nodes into a community so that the nodes in the same community are tightly connected, while the connections between different communities are very sparse. Early community detection algorithms are mainly running on the topological structure, such as graphs. In recent years, many studies have added node attributes to the graph to form an attribute graph, which achieves better performance. The community detection on the attribute graph is now widely used in user similarity analysis and content recommendation systems in social networks. However, there are some challenges we are facing to apply the community detection algorithm to the huge Bitcoin transaction dataset. (1) How to effectively extract the useful information of a specific Bitcoin address from the massive historical Bitcoin transactions? (2) How to handle the mismatching between the topological structure and the node attributes? and (3) How to tune the weights of the node attributes?

To address the above challenges, we design an Adaptive Weighted Attribute Propagation enhanced community detection framework, named AWAP. First, we parse the transactions in the public ledger and construct a bipartite graph [9] to describe the correlations between Bitcoin addresses and transactions. Then, we use a five-step pipeline to extract the Bitcoin address attributes from historical transactions and construct an attribute graph. Finally, we design an adaptive weighted attribute propagation algorithm running on the attribute graph to classify Bitcoin addresses and study the de-anonymization. In AWAP, we treat the transaction attribute graph as a dynamic system and propagates the attributes between nodes across the graph. We name this process *attribute propagation*. We use the results of Bitcoin address classification to analyze the relationship between the nodes. Two case studies are implemented to reveal user behaviors. The experimental results show that AWAP outperforms other state-of-the-art methods. Our contributions are summarized below:

- We design a community detection model that takes transactions from public Bitcoin ledgers as input. The model leverages the attribute propagation to group the Bitcoin addresses with similar behaviors into the same community.
- We propose an adaptive weighted attribute propagation algorithm running on the attribute graph. The algorithm mainly consists of: (1) a random walk to choose the start nodes from the graph, (2) an adaptive weight matrix to control the contribution of attributes, and (3) a voting mechanism to adjust the attribute weights.
- We present a five-step feature engineering pipeline to preprocess the variable-length attribute vector. The pipeline includes (1) data normalization by MinMax, (2) feature generation by feature combination, (3) feature selection by logistic regression, (4) dimensionality reduction by Principal Component Analysis (PCA), and (5) feature discretization by converting numerical features into binary features.

The remainder of this paper is organized as follows: Section 2 introduces the background and related work. Section 3 presents the feature engineering techniques and the proposed framework in detail. Section 4 reports and analyzes the experimental results. Section 5 discusses two case studies and Section 6 gives a brief review of related work. Finally, Section 7 concludes the paper.

2. Background and motivation

In this section, we first briefly introduce the background of Bitcoin, community detection algorithms, and recent research progress of de-anonymization in Bitcoin. Then, we analyze the characteristics of Bitcoin ledger and the feature engineering techniques used to generate the node attributes. Finally, we summarize the goals and challenges of our work.

2.1. Bitcoin

Nowadays, Bitcoin is still the most widely known cryptocurrency and application in Blockchain. It applies Proof of Work (PoW) to eliminate the central bank, and decentralize and secure the transaction ledger in a cooperative, distributed manner [10]. The main challenge to implement Bitcoin is to achieve consensus in generating coins, storing and managing the ledger in a distributed and trustless environment. In a Bitcoin network, each participant generates at least a public/private key pair. When people transfer bitcoins, one participant will generate a transaction which is digitally signed to prove his/her ownership of the bitcoins and announce the transaction publicly. Any Bitcoin user is able to verify the signed transaction with the associated public key. With the increase of Bitcoin transactions over time, Bitcoin becomes a composition of large-scale transactions. Each transaction can hold multiple inputs and multiple outputs. Transactions record the trading relationships between Bitcoin addresses and other trading information, such as transaction fees and the generation time of blocks. We can examine all transactions to extract useful information, such as Bitcoin traders' addresses.

The key information of a transaction is the hash value serving as the transaction identifier and the list of all the inputs and outputs. In Bitcoin, the inputs of the transaction do not specify how many bitcoins are spent. Each output of the transaction can only be used once as an input of another transition over its life cycle. The output is categorized as either the Unspent Transaction Output (UTXO) if it has not been referenced by a subsequent transaction or the Spent Transaction Output (STXO) otherwise. In a standard Bitcoin transaction, the total amount of all the inputs must be at least as much as the total amount of all the outputs. They are not necessary to be equal. If the summation of all the input values is greater than the summation of all the output values, the difference is implicitly assigned as the transaction fee to the miner who validates the block. Transactions are collected in a block by Bitcoin miners through solving a computationally difficult puzzle. The puzzle is to calculate a hash value of the block and to adjust a nonce so that the hash value is lower than or equal to a threshold. Once a miner finds such a nonce, the block along with the respective nonce will be distributed in the network, and all the participants will update their ledgers locally.

The transactions in the ledger are linked over time, and all the transactions are broadcasted to all the participants and stored in their local ledgers. Given the public Bitcoin ledger, we can easily trace any transaction. The identities in Bitcoin are private keys.

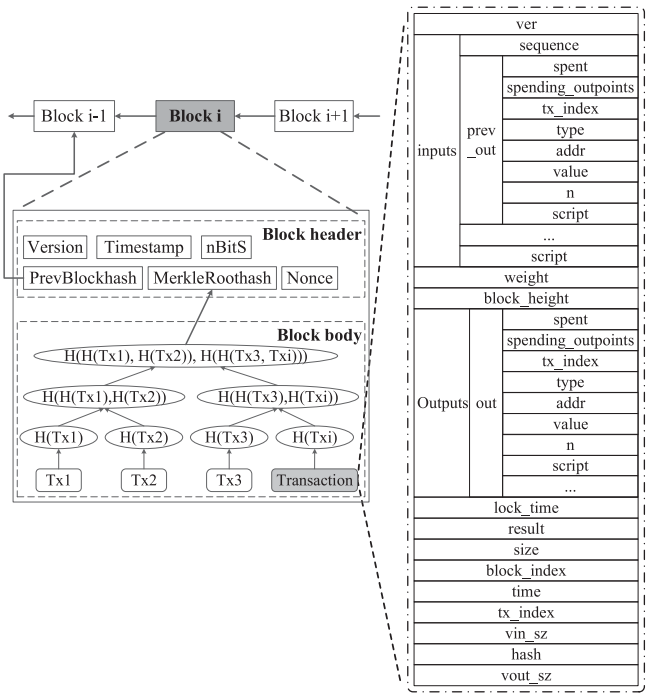

Fig. 1. The structure of the Bitcoin ledger.

Each private key generates a public key for public identification which is associated with some Bitcoin addresses. These addresses keep user anonymity since they contain no links to the real-world entities. A Bitcoin address is also called a pseudonym. According to Nakamoto and Bitcoin [1], the pseudonym mechanism guarantees complete anonymity under the following two conditions. One is that the pseudonym has no connection to the real world identities, and the other is that a new pseudonym should be generated for each new transaction. However, in reality, only few people follow these two rules [8]. In our experiment, by leveraging the information extracted from the Bitcoin transactions, we can reveal the activities among the Bitcoin addresses and eventually de-anonymize the Bitcoin users.

In contrast to Ethereum, Bitcoin transactions on the blockchain are only used for accounting without storing codes or running contracts [11]. Therefore, the data structure of the Bitcoin ledger is simple. Fig. 1 shows the structure of the Bitcoin ledger. The Bitcoin ledger uses a Merkle tree to verify transactions and ensure that transaction information is not tampered with. The Merkle tree is recorded in the block body. *H* represents Hash-256 encryption algorithm. The root hash value of the Merkle tree is stored in *MerkleRoothash* of the block header. *PrevBlockhash* in the block header records the hash value of the previous block header. The leaf nodes of the Merkle tree are transactions. There may be many inputs and outputs in one transaction. *Vin_sz* and *vout_sz* respectively record the number of inputs and outputs in this transaction. *prev_out* in the figure represents one of the input of the transaction. According to the Bitcoin protocol, the inputs of this transaction comes from the outputs of the previous transaction. *Script* is used to unlock the output of the previous transaction. Hash values and *script* bring no information in the feature construction, so we drop them from the raw data. The rest of the data is transformed into structured data. Finally, we use feature engineering methods to generate a feature vector for each Bitcoin address.

2.2. De-anonymization in Bitcoin

Decentralization is the fundamental feature of Bitcoin. The Bitcoin protocol is mainly designed to avoid double-spending attacks on the premise of decentralization. In fact, anonymity is not the main issue considered in the design of Bitcoin. The complete anonymity of Bitcoin is not technically guaranteed but depends on the transaction behaviors of traders. In contrast to other online payment systems, Bitcoin traders can use Bitcoin address to trade directly without verifying user identity information, such as ID cards, SMS verification codes, or face recognition. Therefore, Bitcoin addresses have no link to the real traders, and a trader may own numerous Bitcoin addresses. This anonymity of Bitcoin is designed to protect limited user privacy. Even though all the transaction information is disclosed to the public, we are not able to know the purpose and identity of traders. But, Bitcoin's anonymity mechanism is a double-edged sword. It attracts a large number of users while it also makes the Bitcoin be a tool of circulating currency for illegal activities, such as ransomware, laundering, thefts, and scams [2–4,12]. People can freely disclose their Bitcoin addresses for illegal transactions because no real-world identity will be exposed to others. However, a healthy cryptocurrency system needs to support technical legal investigations to ensure the safety and legality of transactions. We need to de-anonymize the Bitcoin to assist in tracking and verifying those illegal transactions. Currently, there are many methods to identify Bitcoin users' real identities, such as network analysis, address clustering, and graph analysis.

Network analysis. Bitcoin is a decentralized digital currency. Transactions are widely transmitted via a peer-to-peer network. When a node transmits a transaction, the Bitcoin address in the application layer and the IP address in the network layer will appear in the same data packet. We can use specific techniques, such as IP address sniffing, to obtain the mapping between Bitcoin addresses and IP addresses. Koshy et al. [13] develop heuristics to identify three relay patterns for network analysis which maps Bitcoin addresses to IP addresses. Mastan and Paul [14] use a Bitcoin session graph to organize the block requests made by the nodes. Even if the nodes connect to the Bitcoin network over Tor, the method still links the sessions of unreachable nodes. Juhász et al. [15] propose a probabilistic approach that links Bitcoin addresses and transactions with the originator IP addresses. They install more than one hundred modified Bitcoin clients in the network to observe messages and successfully identify several thousand Bitcoin clients.

Address clustering. Some studies cluster Bitcoin addresses into distinct entities. An entity is either a Bitcoin user or a Bitcoin community. We can trace the Bitcoin transaction and detect the transactions by analyzing the flow of bitcoins among these entities [12,16]. Furthermore, Bitcoin users or communities can be identified by combining entities with offline information [17,18]. There are many Bitcoin address clustering methods, such as heuristic clustering [16,19,20], data mining methods [2,21–24] and deep learning methods [8,25,26]. In heuristic clustering methods, the authors propose different heuristic rules according to the Bitcoin transactions characteristics, such as the multi-input heuristic rule. That is, multiple inputs of a transaction may belong to the same entity, and a Bitcoin address may appear in multiple transactions. Under this assumption, multi-input heuristic rules are able to cluster Bitcoin addresses and the clustering *accuracy* achieves almost 70%. Heuristic clustering mainly relies on the transaction behavior of traders and clusters Bitcoin addresses based on the topological structure of the transaction graph. In data mining methods, the authors usually use feature engineering methods to extract feature vectors for each Bitcoin address. Then, they train a classifier such as decision trees, logistic regression, or

random forests to classify Bitcoin addresses. However, these data mining methods usually need a lot of transaction information to generate address feature vectors, and it is hard to discover the relationship among Bitcoin addresses because they ignore the topological structure of the transaction graph.

Graph analysis. Based on the relationship between the transactions and addresses, we can construct an address-transaction graph. Each edge in the graph links a Bitcoin address to a Bitcoin transaction. Therefore, the address-transaction graph is bipartite. The bipartite graph can be further converted into an address graph and a transaction graph. The address graph describes the connections between Bitcoin addresses and reflects intimacy among different addresses. The transaction graph uses the Bitcoin addresses as the edges to describe the connection between transactions, reflecting the input–output relationship of the transactions and the flow path of bitcoins. We can also obtain an entity graph with address clustering. The entity graph provides a clearer picture of the flow of bitcoins between addresses, which reveals the behavior and purpose of transactions between entities. Ranshous et al. [27] introduce the idea of motifs in the directed address-transaction graph. They propose a particular 2-motif and statistical properties of addresses to identify exchange addresses. Ron and Shamir [6] analyze the properties of the transaction graph composed of the full transaction history and answer a series of questions about user behaviors. Chen et al. [28] divide the transactions of Mt.Gox Bitcoin exchange into three categories and construct them into three graphs. They analyze the graphs with Singular Value Decomposition (SVD) and discover plenty of market manipulation patterns.

2.3. Community detection

Community detection is one of the major topics in data mining. It can be used to study the structural characteristics in the graph, such as functional modules of protein–protein interaction networks [29] and groups of people with similar interests in social networks [30]. Graph is one of the popular data structure that is widely used to model the spatial relationship between objects. In real-world graph problems, in addition to the topological structure, nodes are usually associated with node attributes, such as the user profiles in the social networks and the gene functional information in the biological networks. Node attributes can be added to the graph to form an attribute graph. Community detection on attribute graphs aims to group nodes with common properties into a community. However, most community detection methods only focus on either graph topology or node attributes. Examples of topological structure-based methods include modularity [31], spectral clustering [32] and non-negative matrix factorization [33], while node attribute-based methods include k-SNAP [34].

The topological structure and node attributes are both key information for the success of community detection. There is no evidence that topological structure and node attributes share the same characteristics in any case as illustrated in [35]. Node attributes may also unexpectedly mismatch the topology. Besides, different types of attributes usually have different degrees of contribution to the community detection algorithm. To solve the above problems, the input graph of the community detection algorithm should include both topological structure and node information so that the node information is able to propagate across the graph, which is one of the fundamental factors in the social networks [36]. In such a graph, the node at one end of the edge can propagate its attributes to the other end. The whole process describes as follows. A node collects the attributes from its neighboring nodes and merges them with its own attributes to produce the new attributes, and then propagates them to its neighbors. After several iterations, nodes with similar attributes tend to be closer and can be grouped into the same community. In attribute propagation, edges are the medium of the network. Disconnected nodes are not able to propagate attributes to others. In our experiment, we build an attribute graph considering both topological structure and node attributes.

2.4. Feature engineering

Feature engineering is an essential step for the preprocesses of unstructured data. Data and features determine the upper bound of prediction performance, while models and algorithms are just the tools to approach that bound. In Bitcoin applications, the main problem of feature engineering is to deal with the huge size Bitcoin ledger and construct useful features for each Bitcoin address. Jourdan et al. [9] study address-specific features such as the total amount of received bitcoins, the number of input transactions, temporal features such as week and month information, and motif features. Toyoda et al. [24] extract eight transaction characteristics of Bitcoin addresses including transaction frequency and the ratio of received transactions to all transactions. Lin et al. [23] classify all the Bitcoin transaction information into three types of features, basic statistics, extra statistics, and transaction moments. However, these feature engineering methods need to manually determine what attributes should be selected according to the domain knowledge. Furthermore, some latent features may be ignored by the experts. In this paper, we propose a feature engineering pipeline to automatically extract important features among all the transaction information without making any manual choice.

2.5. Goals and challenges

Goals. To achieve Bitcoin de-anonymization, we aim to establish a relationship between the Bitcoin pseudonym and user identification such as user profiles or behaviors. We cluster the Bitcoin addresses with similar behaviors according to the transactions recorded in the public ledger. If one of the Bitcoin address belongs to illegal activities, such as ransomware, we are able to find a cluster of all the illegal addresses so that we can trace malicious sources. In this paper, our goal is **to design an accurate Bitcoin address classifier to group nodes with similar behaviors into the same community and further de-anonymize Bitcoin users.**

Challenges. Recently, the ledger size of Bitcoin transactions has become larger and larger and is almost 300 GB now. It is time-consuming to construct the transaction graph and extract useful features for all the Bitcoin address. Furthermore, the traditional community detection algorithms are not able to work directly on the address-transaction graph. Meanwhile, due to the lack of labeled Bitcoin samples, the performance of supervised learning models are unsatisfactory. Motivated by the information propagation theory, we propose an adaptive weighted attribute propagation enhanced community detection model for Bitcoin de-anonymization. Our model not only considers the topological structures of Bitcoin address and transactions but also includes the node attributes propagation across the graph. In summary, we aim to address the following challenges:

- How to efficiently convert the massive Bitcoin transaction data into a graph?
- What node attributes should be chosen to describe a Bitcoin address for the address classification problem?
- How to construct an attribute graph that can be applied to community detection considering the bipartite address-transaction graph is not suitable for community detection?
- How to combine topological structure and node attributes together considering node attributes sometimes mismatch with topology and how to evaluate the contributions of different node attributes for community detection algorithm?

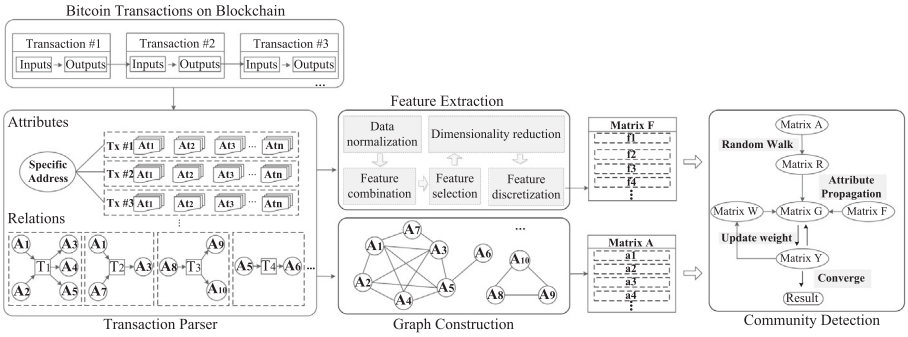

Fig. 2. An overview of AWAP.

3. The AWAP model

In this section, we first give an overview of the AWAP model. Then we present in detail its components of transaction parser, feature extraction, graph construction, and community detection. Finally, we analyze the time complexity of our model.

3.1. Model overview

In Fig. 2, AWAP starts with collecting transaction data from the Bitcoin ledger. After applying the transaction parser, we obtain the Bitcoin address attributes and the address-transaction graph. Then, we pre-process the Bitcoin address attributes with the feature engineering pipeline and convert the address-transaction graph to the Bitcoin address graph. Finally, we combine the pre-processed address attributes and the address graph into an attribute graph as the inputs of the community detection model. Our AWAP model has four components:

- **Transaction parser.** We parser raw transactions into structured data and extract some transaction information as node attributes, such as *spent, value, size, weight* et al. We also build some small bipartite address-transaction graph by linking Bitcoin addresses with transactions.
- **Graph construction.** We convert the bipartite graph into an address graph. For each transaction node T_i, we delete the transaction node T_i and all the edges connected to it, and then we fully connect all the input addresses and output addresses of Transaction T_i. We use an adjacency matrix to represent the undirected address graph.
- **Feature extraction.** We implement a feature engineering pipeline to generate a feature vector for each Bitcoin address from the transaction data. The pipeline consists of data preparation and normalization, feature combination, feature selection, dimensionality reduction, and feature discretization.
- **Community detection.** We combine the address graph and the feature vectors of addresses together to form an attribute graph as the input of the community detection algorithm. Our algorithm clusters nodes with similar properties into a community by propagating node attributes throughout the graph. We also use a voting mechanism to adjust the attribute weights in each iteration.

3.2. Transaction parser

Nowadays, the Bitcoin ledger size has become larger and larger, almost 300 Gigabytes (GB), and still grows over time. In order to process the ledger, the transaction parser needs to analyze and organize the transactions efficiently. Ron and Shamir [6] employ a forked version of *bitcointools* to obtain transaction data and use LevelDB to index the full ledger. Fleder et al. [37] utilize *Armory* to parse transaction data on a full-featured Bitcoin client. Kalodner et al. [38] design a software platform named *BlockSci* to parse and analyze the blockchain.

To collect the data efficiently, we use the *blockchain.info* API to implement the parser. The API retrieves the transactions from the massive transaction history and returns data in JSON format. Then, we convert the JSON format into vectors. In such a way, each Bitcoin transaction becomes a vector of node attributes. Note that the vector has variable length because the number of transactions per address is different. For each transaction, we keep almost all transaction information, such as *value, size, weight,* et al. excluding the hash value that is of no help for the feature construction.

In this step, we construct some small address transaction graphs. For each transaction, we connect it with all the input and output addresses. Each address links to at least one transaction. Each transaction may connect with multiple input addresses and multiple output addresses to form a small bipartite graph. There is no direct link among address nodes or among transaction nodes. We use the same address in different transactions as the medium and splice these binary graphs to form a complete Bitcoin address transaction graph. If Bitcoin users use a unique address for each transaction, each address only appears in one transaction. Then, the small bipartite graph cannot be spliced into a large address transaction graph. From the perspective of graph construction, this kind of bipartite graph is more secure because it has no connection to the other transactions. Thus, using a new address for each transaction can greatly enhance the anonymity of Bitcoin.

3.3. Graph construction

We use a 2-tuple data structure to efficiently store an address graph $G = (V, E)$, where $V = \{v_i | i \in [1, N]\}$ is the set of vertices, $E = \{(v_i, v_j) | v_i, v_j \in V, i \neq j\}$ is the set of edges. The graph is an undirected graph. The adjacency matrix A of the graph G can be represented as:

$$A_{i,j} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E, \text{ or } (v_j, v_i) \in E \\ 0, & \text{otherwise} \end{cases} \quad (1)$$

As shown in the top left part of Fig. 3, we construct the bipartite address-transactions graph in the transaction parser step, where A_i represents the address node i and T_j represents the transaction node j. The bipartite graph shows the relationship between addresses and transactions, but it cannot be applied directly to the community detection algorithm. Because for the community detection task, all the vertices in the graph should be of the same node type. Thus, we convert the bipartite graph to an address graph which only keeps the user address nodes

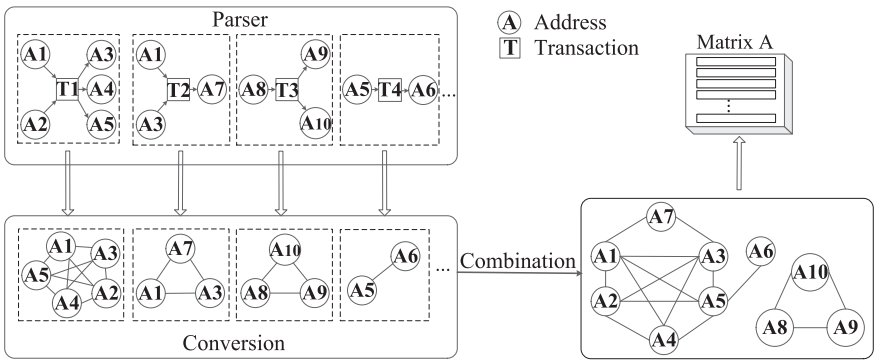

Fig. 3. The construction process of a community graph.

Table 1
Transaction information.

Features	Description
Weight	Block weight for segregated witness.
block_height	Block height of the transaction.
lock_time	Transaction time lock.
Result	The income and expenses of the address
Size	The size of transactions.
Time	Transaction time.
Vin_sz	The number of transaction inputs.
Vout_sz	The number of transaction outputs.
Sum_input	Bitcoins for all input addresses in one transaction.
Aver_input	Average input bitcoins per address.
Max_input	Maximum input bitcoins.
Min_input	Minimum input bitcoins.
Sum_output	Bitcoins for all output addresses in one transaction.
Aver_output	Average output bitcoins per address.
Max_output	Maximum output bitcoins.
Min_output	Minimum output bitcoins.

shown in the bottom part of Fig. 3. In reality, multiple inputs of a transaction may belong to the same Bitcoin user, while multiple output addresses may be of the same entity category, such as Exchange, Gambling, Mining, Service, Darknet. We expect that the address graph is able to describe these relationships as well. To construct the address graph, for each transaction node T_j, we delete the node T_j and all the edges connect with it and add an edge for each pair of (input, output). Then, we merge these small address graphs into a large address graph. Many publications show that the direction of the edges between the vertices is not important as long as the edges can propagate the information between neighboring vertices. Following the above steps, we convert the directed bipartite graph into the undirected graph.

In Bitcoin, there is no transaction between the same addresses, and the diagonal elements of the adjacency matrix are all 0. In order to facilitate the subsequent attribute propagation, we further express the adjacency matrix A of the graph G as:

$$A_{i,j} = \begin{cases} 1, & \text{if } (v_i, v_j) \in E, \text{ or } (v_j, v_i) \in E, \text{ or } i = j \\ 0, & \text{otherwise} \end{cases} \tag{2}$$

3.4. Features extraction

In this part, we use feature engineering techniques to process the above variable-length attribute vector and obtain the feature vector for each user address. Our feature engineering pipeline consists of five parts: data preparing and normalization, feature combination, feature selection, dimensionality reduction, and feature discretization. We show the process of feature extraction in the upper middle part of Fig. 2.

Data preparation and normalization. In the parser step, the transaction attributes used in the experiments are shown in Table 1. Because the number of input addresses and output addresses of each transaction are different, we calculate the *sum, average, maximum, and minimum* values of all the inputs and outputs:

$$Sum\_input = \sum_{i=1}^{N_{in}} input\_value_i \tag{3}$$

$$Aver\_input = \frac{Sum\_input}{N_{in}} \tag{4}$$

$$Max\_input = \max_{1 \le i \le N_{in}} input\_value_i \tag{5}$$

$$Min\_input = \min_{1 \le i \le N_{in}} input\_value_i \tag{6}$$

Here N_{in} represents the total number of input addresses in a transaction and $input\_value_i$ represents the attribute of the ith address. Each transaction node contains 16 attributes. When converting the address-transaction graph to the address graph, we need to define the attributes of address nodes. Each Bitcoin address nodes may appear in multiple transactions. Therefore, we calculate the *average, maximum and minimum* values of attributes for all the transactions that are associated with this address. Finally, we obtain 48 attributes for each address. Here is an example of calculating the address attributes for the transaction feature *Min_input*:

$$Aver\_Min\_input = \frac{\sum_{i=1}^{N_{tx}} Min\_input}{N_{tx}} \tag{7}$$

$$Max\_Min\_input = \max_{1 \le i \le N_{tx}} Min\_input_i \tag{8}$$

$$Min\_Min\_input = \min_{1 \le i \le N_{tx}} Min\_input_i \tag{9}$$

Here N_{tx} represents the total number of transactions that are associated with the current address. Min_input_i represents the *Min_input* of the ith transaction obtained from Eq. (6). Adding the four more address attributes including *N_tx*, *BTC_received*, *BTC_sent*, and *Final_balance* provided by the API, we finally construct a 52-dimensional feature vector for each address. We normalize all the address attributes into the same range with Z-score and MinMax functions.

Feature combination. Feature combination is one of the data pre-processing tools that create new features. It is able to add

the non-linearity into the feature vector. Shao et al. [8] use a polynomial feature combination when constructing Bitcoin address features. Similarly, for each feature, we add the squared value and square root value into the feature vector. For example, the feature *Max_Min_input* is able to generate two new features $\sqrt{Max\_Min\_input}$ and $(Max\_Min\_input)^2$.

Feature selection. Feature selection is such a tool that helps us remove redundant or irrelevant features. It may also improve the performance and reduce the runtime because we only keep the highly important features. Feature selection methods are mainly classified into two categories: statistical-based feature selection and model-based feature selection. In the model-based approach, we pre-train a decision tree or a logistic regression model to evaluate the importance of each feature. In the statistical-based methods, we use the Pearson coefficient and hypothesis test *p*-value to select features. Here is the equation of the Pearson coefficient:

$$Pearson_{X,Y} = \frac{cov(X, Y)}{\sigma_X \sigma_Y} \tag{10}$$

Here X represents a feature, Y represents the dependent variable, σ is the standard deviation, and cov is covariance. Pearson coefficient reflects the degree of linear correlation between feature X and dependent variable Y. The closer the coefficient is to 1, the stronger the correlation between the feature X and the dependent variable Y. In the hypothesis test, we use the chi-square test to compute the p-value. The chi-square value expresses the independence of two variables. The larger the chi-square value is, the smaller the p-value is, the weaker the independence between the feature X and the dependent variable Y.

Dimensionality reduction. We use Principal Components Analysis (PCA) to reduce the dimensionality of the feature vector. In the above step, we have already used the feature selection to reduce the data dimensionality. We use PCA because PCA projects a dataset with some related features onto a coordinate system to generate a new representation of fewer related features. These new related features are called principal components. PCA creates fewer features but has a much stronger representation ability by maximizing the data variance. Therefore, the new representation is usually more effective to identify patterns comparing to the original feature representation.

Feature discretization. Discretization is the process of transforming continuous variables into discrete variables by using a set of contiguous intervals. To speed up the community detection, we need data discretization methods to convert numerical features into binary features. Besides, data discretization can make features obtain exponential representation capabilities. If we have an n-dimensional continuous vector, we will get 2^n feature combinations even though we use the simplest feature binarization. We use a decision tree to determine the boundary of each feature value, map the feature value to the corresponding interval, and finally complete the data discretization.

We combine all the feature vectors into a matrix F, $F = [\boldsymbol{f_1}, \boldsymbol{f_2}, \ldots, \boldsymbol{f_N}]^T$. Specifically, $\boldsymbol{f_i}$ is the feature vector of vertices i. $G = (V, E, F)$ denotes the attribute graph. In the experiment, we test all the combinations of different feature engineering techniques and select the best approach based on their classification accuracy.

3.5. Community detection

In this subsection, we introduce our adaptive weighted attribute propagation enhanced community detection algorithm. An attribute graph with both topological structure and node attributes can be represent by a graph $G = (V, E, F)$, where V is the set of vertices, E is the set of edges and F is the set of feature vectors, $F = [\boldsymbol{f_1}, \boldsymbol{f_2}, \ldots, \boldsymbol{f_N}]^T$. $\boldsymbol{f_i}$ is a feature vector associated with vertex i. Given the graph G, our goal is to cluster all the vertices into K different communities. To achieve that, we utilize a random walk based attribute propagation algorithm.

Firstly, we introduce the process of attribute propagation. Considering a random walk on the attribute graph G, given an arbitrary pair of vertices (i, j), the one-step transition probability P_{ij} denotes the probability that a walk starts from vertex i at $t = 0$ and stops to vertex j at $t = 1$. We can further represents the P_{ij} as follows:

$$P_{ij} = P_{t=1|0}(j|i) = \frac{A_{ij}}{\sum_{l=1}^{N} A_{il}}, \tag{11}$$

where A is the adjacency matrix of graph G and A_{ij} is the corresponding value in the adjacency matrix A at row i and column j. N is the total number of vertices in the graph. In the adjacency graph, A_{ij} equals to 1 if and only if vertex i and vertex j are neighbors, otherwise A_{ij} equals to 0. The denominator $\sum_{l=1}^{N} A_{il}$ equals to the total number of vertex i's neighbors. From Eq. (11), we can conclude that the more neighbors the vertex i has, the larger the denominator is and the smaller the one step transition probability is. In real application, we treat step $\mathbf{t}$ as a random variable from $[0, \infty)$ and assume $\mathbf{t}$ follows a geometric distribution and the probability of $P(\mathbf{t} = s) = \lambda(1 - \lambda)^s$, where s is an integer in the range of $[0, \infty)$ and λ is a parameter in the range of $(0, 1)$. Furthermore, the expectation of transition probability from vertex i to vertex j is defined as:

$$P_{\mathbf{t}|0}(j|i) = \sum_{s=0}^{\infty} P_{s|0}(j|i) \cdot P(\mathbf{t} = s) = \sum_{s=0}^{\infty} P_{s|0}(j|i) \cdot \lambda(1 - \lambda)^s \tag{12}$$

In Eq. (13), the probability that vertex j receiving attribute propagation from vertex i can be calculated as the ratio of the expectation of transition probability from vertex i to vertex j to the summation of the expectation of transition probability from any vertex j's reachable vertex l to vertex j.

$$P_{0|\mathbf{t}}(i|j) = \frac{P_{\mathbf{t}|0}(j|i)}{\sum_{l=1}^{N} P_{\mathbf{t}|0}(j|l)} \tag{13}$$

Based on Eq. (13), if many vertices are able to reach vertex j, the probability of vertex j receiving the attribute propagation of a particular vertex i will be low. A matrix R is utilized to store all the attribute propagation probabilities where $R_{ij} = P_{0|\mathbf{t}}(i|j)$. Given feature matrix F and R, we can calculate the total attributes received by vertex j as $\boldsymbol{g_j} = \sum_{i=1}^{N} \boldsymbol{f_i} R_{ij}$. The attribute propagation matrix G can be written in matrix form:

$$G = FR \tag{14}$$

As mentioned in Section 2, node attributes may mismatch with topology and different types of attributes have different degrees of contribution to the results of community detection. An adaptive weight matrix W is used to set the weight for each node attribute according to their contribution. W is a diagonal matrix that $W_{ii} = w_i$, $\forall i \in [1, m]$, where m is the size of feature vectors. We initialize $w_1, \ldots, w_m$ to 1.0, which indicates that all the attributes are equally important. Thus, the weighted attribute propagation matrix G is represented as:

$$G = WFR \tag{15}$$

Specified in Liu et al. [36], the key element of attribute propagation is the assumption of community consistency. When the propagation reaches stability, vertices in the same community are likely to receive the same amount of attribute propagation. Now, we define $\boldsymbol{\mu_k}$ as the expected attribute propagation for the community C_k.

$$\boldsymbol{\mu_k} = \frac{\sum_{j=1}^{m} \boldsymbol{g_j}}{m}, \tag{16}$$

where $\boldsymbol{g_j}$ denotes the attribute received by vertex j from community C_k, $\forall j \in [1, m]$ and m is the total number of vertices in the community C_k. The expected attribute propagation of the community can be obtained by averaging the attribute propagation of all vertices in this community. We use the matrix Y to represent the belonging relationship between the vertices and the communities. Specifically, $Y_{ik} = 1$ indicates that vertex i belongs to community C_k. $E[\boldsymbol{g_j}]$ represents the expectation of attribute propagation received by vertex j. K is the number of communities.

$$E[\boldsymbol{g_j}] = \sum_{k=1}^{K} \boldsymbol{\mu_k} \cdot Y_{jk}, \ \ where\ Y_{jk} = \begin{cases} 1, & j \in C_k \\ 0, & j \notin C_k \end{cases} \tag{17}$$

In our AWAP model, a vertex j can only be assigned to one community if all the attribute propagation received by vertex j is solely from the vertices in its community. Then, community detection based on attribute propagation can be obtained by solving the following optimization problem:

$$\arg\min_{Y, \boldsymbol{\mu_k}} \sum_{i=1}^{N} \| \boldsymbol{g_i} - E[\boldsymbol{g_i}] \|_2^2 \tag{18}$$

To solve the above optimization function, we minimize the difference between the actual $\boldsymbol{g_i}$ and expected $\boldsymbol{g_i}$. Given the dependent variable Y, we are able to solve $\boldsymbol{\mu_k}$ with Eq. (16):

$$\boldsymbol{\mu_k} = \frac{\sum_{i=1}^{N} \boldsymbol{g_i} \cdot Y_{ik}}{\sum_{j=1}^{N} Y_{jk}} \tag{19}$$

Taking Eqs. (17) and (18), we can solve matrix Y by calculating the first K eigenvectors of matrix $R^\top F^\top W^\top WFR$ according to Xu et al. [39]. After initializing the matrix Y, we use Eq. (19) to calculate $\boldsymbol{\mu_k}$. During each iteration, we update $\boldsymbol{\mu_k}$ and Y as follows:

$$\boldsymbol{\mu_k}^{t+1} = \frac{\sum_{i=1}^{N} \boldsymbol{g_i} \cdot Y_{ik}^{t}}{\sum_{j=1}^{N} Y_{jk}^{t}} \tag{20}$$

$$Y_{ik}^{t+1} = \begin{cases} 1, & \forall m \in [1, K], \|\boldsymbol{\mu_k}^t - \boldsymbol{g_i}\| \leq \|\boldsymbol{\mu_m}^t - \boldsymbol{g_i}\| \\ 0, & otherwise \end{cases} \tag{21}$$

Finally, a voting mechanism similar to Zhou et al. [40] is utilized to adjust attribute weights. Let $w_1^t, \ldots, w_m^t$ be weights of attributes $a_1^t, \ldots, a_m^t$ in the tth iteration, where m is the size of the feature vector. The weight of attribute a_i in the $(t+1)th$ iteration is computed as:

$$\boldsymbol{w_i}^{t+1} = \frac{1}{2}(\boldsymbol{w_i}^t + \Delta \boldsymbol{w_i}^t) \tag{22}$$

We can calculate $\Delta \boldsymbol{w_i^t}$ with the voting mechanism. If vertices in the same community share the same attribute value a_i, it means attribute a_i can reflect the community characteristics. Then the weight w_i of a_i should increase, while if vertices in the same community have a random distribution on attribute a_i, the weight w_i should decrease. The voting process can be computed as:

$$vote_i(v_p, v_q) = \begin{cases} 1, & if\ v_p, v_q\ share\ the\ same\ value\ on\ a_i \\ 0, & otherwise \end{cases} \tag{23}$$

Based on Eq. (23), $\Delta \boldsymbol{w_i}^t$ is calculated as

$$\Delta \boldsymbol{w_i}^t = \frac{\sum_{j=1}^{K} \sum_{v \in V_j} vote_i(c_j, v)}{\frac{1}{m} \sum_{p=1}^{m} \sum_{j=1}^{K} \sum_{v \in V_j} vote_p(c_j, v)}, \tag{24}$$

where V_j denotes the set of vertices in community C_j, c_j denotes a virtual vertex with expectation attributes of community C_j. The algorithm of community detection on AWAP model is summarized in Algorithm 1.

Algorithm 1 Adaptive Weighted Attribute Propagation

Input: adjacency matrix A; feature matrix F; number of clusters K; parameter λ;
Output: detected communities indicated by Y;
1: Initialize $w_1 = w_2 = ... = w_m = 1.0$;
2: Calculate R matrix with random walk, according to equation (13);
3: Calculate G based on equation (15);
4: Calculate the first K eigenvectors and initialize Y;
5: Initialize μ with equation (19);
6: **while** not converged **do**
7: Update Y with μ and W based on equation (21);
8: Update μ with Y based on equation (20);
9: Update weights $w_1, w_2, ..., w_m$ based on equation (22);
10: **end while**
11: Return Y;

Complexity analysis. With the growth of ledger size in Bitcoin, community detection has become a time consuming and space consuming process. In the AWAP framework, there are a large number of matrix operations. The most time consuming process is division operation in Eq. (11) and matrix multiplication in Eq. (15). Their time complexity is $O(n^2)$ and $O(n^3)$ respectively. Besides, we have to store some intermediate result of matrices or vectors for following-up computation in Eq. (15), and its space complexity is $O(n^2)$. The total time complexity of AWAP is $O(2n^3 + n^2 + (m+K+1)n)$ and the total space complexity is $O(3n^2 + Kn + K + m)$, where n, m and K are respectively the size of samples, the size of features, and the size of clusters. Furthermore, parallel computing is able to speed up the computational process, which is beyond the scope of this paper.

4. Evaluation

In this section, we demonstrate the performance of our AWAP model on Bitcoin address classification and community detection. We choose seven state-of-the-art Bitcoin address classification methods and seven state-of-the-art community detection algorithms as the baselines. The evaluation process studies the following questions:

- Is AWAP model effective in Bitcoin address classification compared with state-of-the-art classifiers?
- Does AWAP model outperform other state-of-the-art algorithms in the community detection task?
- How does feature engineering influence AWAP model's performance?

4.1. Experimental setup

Testbed. We develop the AWAP model on a Windows 10 machine with Intel Core 2.20 GHz CPUs and 16 GB RAM. We implement the transaction parser in Python 3.7.2 and implement AWAP model in Matlab. We collect 300 GB of raw transactions to test the traceability of the ransomware address in Bitcoin. To compare the performance between the AWAP model and other state-of-the-art methods, we select a labeled Bitcoin dataset to evaluate the address classification performance [9]. Furthermore, we use two standard citation datasets (Cora and Citeseer) to evaluate community detection performance. The details of the datasets are listed in Table 2.

Table 2
The experimental dataset used to evaluate our model.

Dataset	Data size	#Item	Description	
Cora	7.52 MB	N/A	Node Link Dimension Class	2708 5294 1433 7
CiteSeer	23.5 MB	N/A	Node Link Dimension Class	3312 4732 3703 6
Data(I)	20.9 GB	59 836	Exchange Gambling DarkNet Mining Service	11 808 11 923 12 016 12 042 12 047
Data(II)	N/A	10 000	Exchange Gambling DarkNet Mining Service	2000 2000 2000 2000 2000
Data(III)	N/A	7882	N/A	N/A

Measured metrics. *F-score*, *Jaccard similarity* and Normalized Mutual Information (*NMI*) are used to evaluate the performance of community classification. *F-score* mainly describes the *accuracy* of detected communities, *Jaccard* is a statistical method used for comparing the similarity of detected communities and the ground truth, and *NMI* offers an entropy measure of the overall matching. These metrics are formulated as follows:

$$F_{score}(C, C^*) = \sum_{C_i \in C} \frac{|C_i|}{\sum_{C_j \in C} |C_j|} \max_{C_j^* \in C^*} F_{score}(C_i, C_j^*) \tag{25}$$

$$\begin{aligned} Jac(C, C^*) = &\sum_{C_j^* \in C^*} \frac{\max_{C_i \in C} Jac(C_i, C_j^*)}{2|C^*|} \\ &+ \sum_{C_i \in C} \frac{\max_{C_i^* \in C^*} Jac(C_i, C_j^*)}{2|C|} \end{aligned} \tag{26}$$

$$\begin{aligned} NMI(C, C^*) = &\sum_{C_i, C_j^*} p(C_i, C_j^*)(log\ p(C_i, C_j^*) \\ &- log\ p(C_i)p(C_j))/\max(H(C), H(C^*)) \end{aligned} \tag{27}$$

Here C^* is the real community and the C is the predicted community. $H(C)$ is the entropy of the community C. *Accuracy*, *Precision* and F_{1_score} are used to evaluate the performance of Bitcoin address classification. *Accuracy* is the proportion of correct predictions to total predictions. *Precision* is the proportion of positive predictions to the total positive predictions. *Recall* represents a measure of the completeness of a classifier. F_{1_score} is the harmonic mean of *Precision* and *Recall*. All the above six metrics range from $[0, 1]$. The larger the metric is, the higher performance the classifier achieves. F_{1_score} can be computed with the formula below:

$$F_{1\_score} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \tag{28}$$

4.2. Model performance

In this part, we report and discuss the performance of our model on both the Bitcoin address classification task and the community detection task. We also analyze how the parameters and weight distribution influence the model performance.

Table 3
Performance comparison on Bitcoin address classification task.

Algorithm	*Accuracy*	*Precision*	F_{1_score}
Logistic regression	0.88	0.89	0.88
LightGBM	0.90	**0.91**	0.90
Random forest	**0.90**	0.90	0.90
Decision tree	0.86	0.88	0.86
XGBoost	0.80	0.81	0.81
BAGC	0.55	0.47	0.50
CP	0.89	0.71	0.79
AWAP	**0.95**	0.88	**0.92**

Table 4
Bitcoin address classification results.

Category	*Accuracy*	*Precision*	F_{1_score}
Exchange	0.96	0.95	0.90
Gambling	0.98	0.95	0.97
DarkNet	**0.93**	0.98	**0.77**
Mining	0.99	0.99	0.99
Service	**0.90**	**0.68**	0.93

Address classification performance. In this task, we use seven state-of-the-art community detection algorithms, such as BAGC [41] and CP [36], to classify Bitcoin addresses, and summarize the *accuracy*, *precision*, and F_{1_score} of address classification in Table 3. To prove that our model can effectively work on a small sub-graph of the whole Bitcoin dataset, we only select the first 2000 addresses for each category from the labeled dataset Data(I) to form the dataset Data(II). The results show that our method has higher *accuracy* compared to all the other methods. AWAP increases accuracy by 5% compared to Random Forest, and by 40% compared to BAGC. Regarding F_{1_score}, AWAP outperforms all the others by 2% the minimum (compared to LightGBM) to 42% the maximum (compared to BAGC). Regarding *precision*, AWAP is inferior to LightGBM and Logistic Regression by about 3%. According to their definitions, *accuracy* reflects the ratio of the number of correctly predicted samples to the total number of predictions, regardless of whether the predicted samples are positive or negative, while *precision* only focuses on the data points that are predicted to be positive. Thus, the reason why *accuracy* is higher while *precision* is slightly lower may be because the classification of most addresses is correct, but some samples from one of categories are misclassified.

We visualize the results of Bitcoin address classification to further analyze the experiment results. The comparison results between prediction and the ground truth are shown in Fig. 4. Our graph visualization is not fixed so that nodes in the two graphs are not one-to-one spatial correspondence. Each node in the graph represents a Bitcoin address, and the edge represents that two end nodes connect with the same transaction. They can be any one of the addresses of the inputs/outputs of the transaction. By observing the visualization results shown in Fig. 4, our model is able to accurately classify most of the nodes. The nodes connect in the graph are more likely to share some node attributes. The nodes connected in the topology are more likely to belong to the same community, but this is not absolute. The difference in attributes of each address may still divide the nodes connected by the topology into different communities. So the address attribute determines the type of community, and AWAP can make good use of topology and node attributes for classification. Our proposed community detection model further makes a prediction based on the propagation of node attributes.

The *accuracy*, *precision*, and F_{1_score} of the five address types are summarized in Table 4. After checking the misclassified samples, we find that some DarkNet addresses are misclassified to Service addresses, resulting in a decrease in the *accuracy* of DarkNet

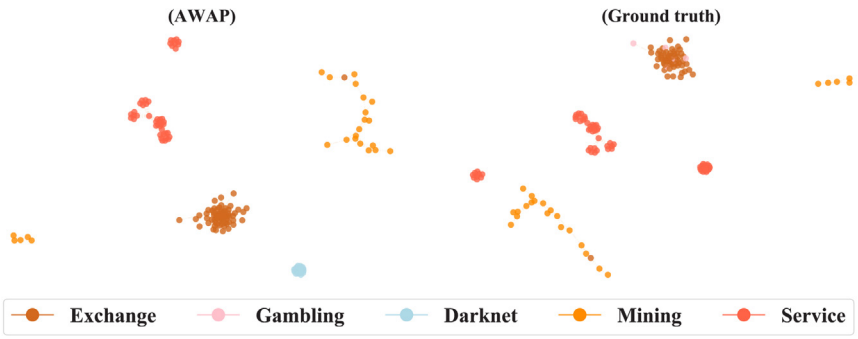

Fig. 4. Comparison between AWAP and ground truth on Bitcoin address classification.

and Service, and *precision* of Service. We can make the same conclusion from Fig. 4. Some DarkNet nodes marked with blue are misclassified to Service nodes marked with red. In other words, our model recognizes some of the DarkNet addresses as Service addresses since the node attributes of these two types have high similarity.

Community detection performance. In this task, we choose seven state-of-the-art community detection methods as baselines and compare AWAP model performance with them on Citeseer and Cora datasets. These datasets are summarized in Table 2. *F-score*, *Jaccard similarity* and *NMI* are used to evaluate the performance of community detection. We summarize the comparison results in Table 5.

The results show that AWAP model outperforms the baselines. From Table 5, the algorithms that comprehensively consider the topology and node attributes are superior to the algorithms that only consider the topology. Dynamically adjusting attribute weights also improves the quality of community detection. In our model, we use the feature engineering methods to increase the weights of important attributes and reduce the weights of attributes that mismatch with the topology. We adjust the attribute weights based on the experimental analysis and discuss the different distribution of weighted attributes in the next part. We also design an attribute propagation mechanism in AWAP model based on information propagation. In this mechanism, the topology works as a medium to transfer node attributes. In this way, the topological structure and the node attributes are naturally combined together to dynamically adjust the attribute weights. Thus, our adaptive weighted attribute propagation enhanced community detection methods to achieve the best performance in address classification in Bitcoin.

Discussion. In this part, we discuss the influence of parameters in AWAP model. We mainly focus on how data size and attribute weights influence the performance. We also analyze the parameter λ in Eq. (12). A smaller λ enhances the ability of attribute propagation in the graph. In our experiments, we test λ from 0.05 to 1, but the value of λ has a limited influence on the classification results. One possible reason is that the model has already finished the attribute propagation in the feature engineering step because the address attributes are collected from all the transactions that it involves.

Data size. In this task, we choose the data size from 2500 to 15,000 with a step of 2500 from Data(I) in Table 2. The experimental results are summarized in Fig. 5(a). With the growth of the data scale, all three metrics gradually increase. When the data scale is close to 10,000, the values of metrics tend to be stable. When the data size is 12,500, they reach the maximum values. Small data size causes a small graph and low connectivity between nodes, which limits the attributes propagation throughout the graph. Therefore, a sufficient big dataset is a key factor in the success of our community detection model.

Attribute weight. In this task, we use Data(II) in Table 2 to discuss how the attribute weights influence AWAP model. We aggregate the feature weight vectors to calculate the weights of 52 attributes and summarize them in Fig. 5(b). Two attributes have the highest weights, *N_tx* (the number of transactions) and *BTC_sent* (the total amount of bitcoins that an address is sent). Two attributes have the lowest weights are *Max_Vout_sz* (the maximum number of transaction outputs) and *Aver_Min_input* (the average value of the minimum transaction input value). We show the distribution of *BTC_sent* and *Aver_Min_input* in Fig. 5(c) and 5(d) respectively. Different colors from left to right indicate different communities, Exchange, Gambling, DarkNet, Mining, and Services. In Fig. 5(c), we observe that nodes in DarkNet are concentrated from 0.25 to 0.8, while nodes in the other types are mainly concentrated below 0.2. Thus, *BTC_sent* is an effective feature to distinguish the nodes between DarkNet and the rest types. This is also the reason why our model set a higher weight for *BTC_sent* compared to other attributes. While the value of *Aver_Min_input* in Fig. 5(d) is relatively small so that the importance of this feature is weak.

4.3. Feature engineering analysis

In this part, we analyze the influence of different feature engineering methods on AWAP model. In Section 3.4, we design a feature engineering pipeline that combines five state-of-the-art feature pre-processing methods to generate the feature vectors. In order to find the best feature engineering strategy, we evaluate all the combinations. In this task, we use Data(II) in Table 2 for performance evaluation.

Method encoding. In Table 6, the options for different feature engineering combinations are illustrated. To test all the combinations, we implement 60 feature engineering pipelines with permutation. Due to logistic regression method in the feature selection stage always require data normalization, so we only implement 56 feasible feature engineering pipelines in the end. To facilitate the description, we use a five-digit number to represent each feature engineering pipeline. The five numbers correspond to the five steps of the pipeline. The value on each digit indicates the method used in that step. For example, "21201" represents a feature engineering pipeline that uses MinMax for data normalization, performs feature combination, uses *p*-value for feature selection, does not perform PCA, and uses a decision tree for feature discretization. We also definite a wildcard "x" to represent an arbitrary method in this step.

Data normalization. In data normalization steps, there are three options: no data normalization, Z-score, and MinMax. In Fig. 6(a), we show the *accuracy* of different data normalization methods associate with other features engineering methods. The difference in results is mainly reflected in the feature selection using logistic regression, that is, the feature engineering pipeline

Table 5
Performance evaluation on community detection task.

Algorithm	Information	Citeseer			Cora		
		F-score	*Jaccard*	*NMI*	*F-score*	*Jaccard*	*NMI*
CNM	Topology	0.1735	0.1094	0.2290	0.4210	0.2315	0.1491
DeepWalk	Topology	0.2699	0.2481	0.0878	0.3917	0.3612	0.3270
Big-CLAM	Topology	0.5114	0.0872	0.2197	0.4829	0.2340	0.2919
Circles	Topology+Attributes	0.3405	0.1867	0.0024	0.3595	0.1810	0.0064
CP	Topology+Attributes	0.6918	0.4991	0.4314	0.6770	0.5168	0.4863
CODICIL	Topology+Attributes	0.5953	0.4041	0.3392	0.5857	0.3947	0.4254
CESNA	Topology+Attributes	0.5240	0.1158	0.1158	0.6059	0.3254	0.4671
AWAP	Topology+Attributes	**0.7134**	**0.4570**	**0.5205**	**0.7583**	**0.5875**	**0.5683**

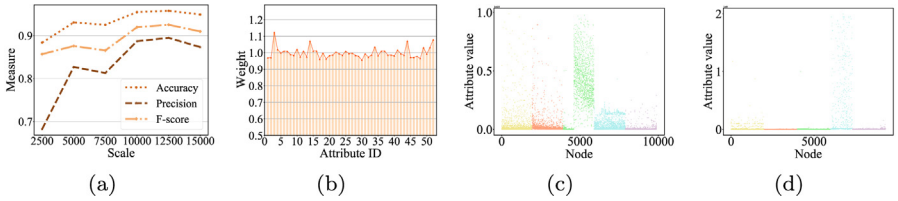

Fig. 5. The performance of AWAP on: (a) different data size; (b) weight distribution of different attributes; (c) attribute distribution of *BTC_sent*; (d) attribute distribution of *Aver_Min_input*.

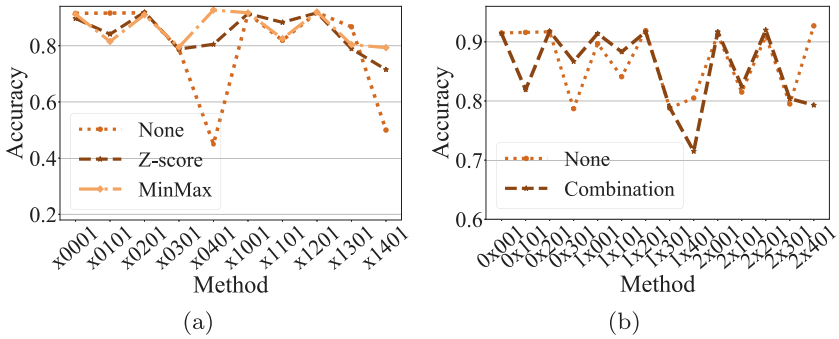

Fig. 6. The performance of AWAP model on: (a) different data normalization; (b) different feature combination.

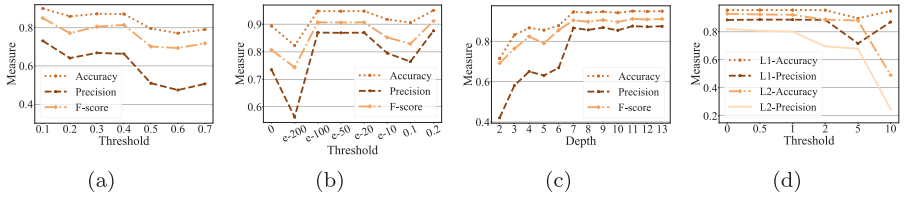

Fig. 7. The performance of AWAP on: (a) different Pearson thresholds; (b) different *p*-value thresholds; (c) different depths of decision tree; (d) different logistic regression thresholds.

Table 6
Representation of feature engineering methods.

Steps	Methods
Data normalization	No(0),Z-score(1),MinMax(2)
Feature combination	No(0),Yes(1)
Feature selection	No(0),Pearson(1),P-value(2), Decision tree(3), Logistic regression(4)
PCA	No(0),Yes(1)
Feature discretization	Yes(1)

is "x0401" and "x1401". For logistic regression, we can use MinMax to get the highest *accuracy*, but if we do not use any data normalization methods, the logistic regression result is bad. From Fig. 6(a), we know that data normalization is not necessary for other feature selection methods. When the Pearson correlation coefficient used for feature selection and feature combination not performed, the *accuracy* without data normalization is even better than the result with data normalization. When we apply feature combination, data normalization can help slightly improve the *accuracy*.

Combination. In feature combination steps, there are two options: no combination and combination. In Fig. 6(b), we show the effect of feature combinations on *accuracy*. Feature combination can create new features and enhance the non-linearity of input data. In most cases, the results of the feature combination method are slightly better than the results without using the feature combination. However, for those methods applying logistic regression in feature selection, the classification accuracy is worse when using the feature combination. One possible reason is that logistic regression is a linear classifier, while non-linear features may reduce its ability. In addition, after using MinMax for data

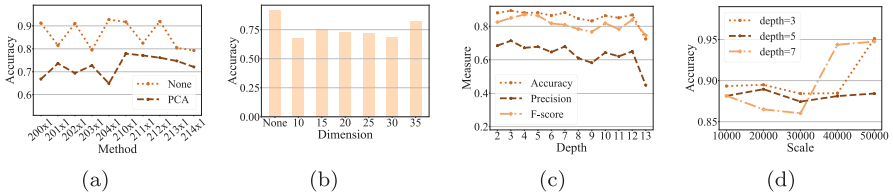

Fig. 8. The performance of AWAP on: (a) PCA; (b) different feature vector dimensions; (c) different depths of decision tree in feature discretization; (d) different data scales in feature discretization.

normalization, the feature combination cannot give full play to its advantages. Feature combinations are more suitable for use in decision trees without data normalization.

Feature selection. In feature selection steps, there are five options: no selection, Pearson coefficient, hypothesis test, decision tree, and logistic regression. This step can speed up the classification process and eliminate noise features. The detailed performance of feature selection methods is shown in Fig. 7.

In Fig. 7(a), we choose the Pearson coefficient as the feature selection method and study the different thresholds. Firstly, we calculate the Pearson coefficient for each feature. The closer the absolute value of the Pearson coefficient is to 1, the more important the feature is. Secondly, we summarize the performance with different thresholds. As the threshold continues to increase, the number of selected features decreases and the performance of classification is also declining. When we set the threshold to 0.1, the result is optimal among all the thresholds.

In Fig. 7(b), we show the classification results of the hypothesis tests with a different p-value. The closer p-value is to 0, the higher correlation between the feature and the label. As the threshold increases, the number of selected features increases as well. When the threshold is between 10^{-100} and 10^{-20}, the number of selected features is stable and all three metrics are maximized.

In Fig. 7(c), we study decision trees for feature selection and analyze the effect of tree depth on the classification task. As the depth of the tree increases, the *accuracy* of the classification results gradually increases and then tends to be stable. If we set the threshold of feature weight to 0, we can select features with a weight greater than 0. With the depth increasing, the structure of the tree becomes more and more complex, and the number of selected features increases. After the depth reaches 7, the performance is no longer improved.

In Fig. 7(d), we show the result of feature selection using the logistic regression model. We use L1 regularization and L2 regularization for experiments. In terms of classification *accuracy* and *precision*, L1 regularization is better than L2 regularization in general. The horizontal axis represents the threshold of feature weight. As the threshold increases, the number of selected features decreases and the performance of L2 regularization gradually decreases.

PCA. In Fig. 8(a), we show the influence of Principal Component Analysis (PCA) on the experimental results. PCA reduces the data dimension and improves the classifying ability on high variance features. In the experiment, we use PCA to reduce the size of the original feature vectors to 15. The experimental results show that the classification *accuracy* is greatly descended when applying PCA. We also study the influence of different output feature size for PCA and show the result in Fig. 8(b). Whatever output feature size we choose, the accuracy is always worse than the framework without PCA. One possible reason is that PCA loses some important information during computing the principal components, where this information is important for our model.

Decision tree depth. To accelerate the speed of community detection and improve the feature representation ability, we use data discretization to convert a numerical vector into a binary vector. In this stage, we use a decision tree to determine the boundaries of each feature. The depth of the decision tree is an important factor in determining the boundaries. In Fig. 8(c), we test different tree depths for the data size of 10,000. From the results of all three metrics, the best three depth is 3. As the depth continues to increase, the *accuracy* declines gradually. A deep depth makes each feature be divided into lots of intervals. In the meanwhile, we explore the influence of data size along with tree depth in Fig. 8(d). Small-scale data does not require too deep decision trees. A decision tree with a depth of 3 obtains the best results. As the size of the data increases, deeper decision trees have more advantages.

5. Case studies on Bitcoin de-anonymization

In this section, we focus on two case studies on Bitcoin de-anonymization: Bitcoin address classification and Bitcoin traceability. We use Data(II) in the former case and Data(III) in the latter case from Table 2. We use AWAP to analyze a public dataset and trace three Bitcoin addresses disclosed in the ransomware respectively. The results show that AWAP framework is able to cluster similar addresses into a community and mining more related addresses for further analysis.

5.1. Address classification

Address classification is one of the basic steps of Bitcoin De-anonymization. In Bitcoin network, the user identity is a hash value or public key which has no relationship with the real-identities in the real-world. However, we still can analyze the similarity of transaction behaviors and cluster similar addresses into a group. In this way, the regulators are able to discover more illegal transactions by running address classification algorithm on the public ledger. In the first study case, we randomly selected 10,000 Bitcoin addresses from the dataset Jourdan et al. [9] to evaluate AWAP model's classification capability. We build an address graph based on transactions among addresses and use a feature engineering pipeline to generate a feature vector for each address. In the pipeline, we choose the MinMax method to normalize data, a logistic regression based on L1 regularization to select features, and a decision tree to decompose each feature into a set of bins feature. The depth of the decision tree is set to 3. The threshold is set to 1. We also select 24 features from 52 original features. Finally, each Bitcoin address can be represented by a 184-dimensional feature vector. Our community detection algorithm takes the attribute graph as the inputs to cluster the Bitcoin addresses into different types of groups. The experimental results show in Fig. 9.

Our model is able to group the same types of addresses into the same community even though they may not connect with each other topologically. Nodes in the same community tend to belong to users with similar trading behaviors or even belong to the same user. In Fig. 9, if we only know in advance that an address *1DJT8D...* is a Mining address, we can find that address

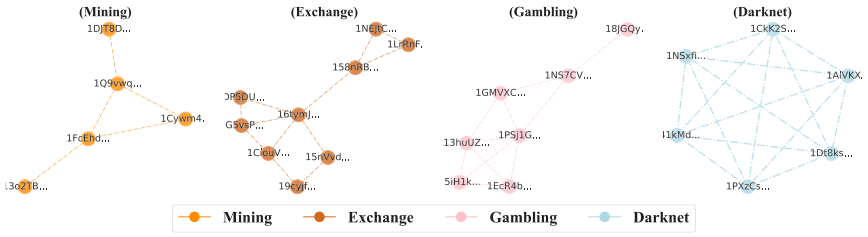

Fig. 9. The community of different types obtained from AWAP.

1Q9vwq... may also be a Mining address from the classification results of AWAP model. They may belong to the same mining pool or even the same Bitcoin user. Even if there is no direct transaction between address *1DJT8D...* and address *13o2TB...*, we cannot rule out the possibility that they belong to the same user or community.

In Fig. 9, given a labeled node, we are able to classify all the nodes in the same community. Especially in Fig. 9, the nodes in the Darknet community constructs into a complete graph, where all the nodes in the community tend to connect to each other. This shows that the trading behaviors among Darknet addresses are more frequent compared to other types of community. The traders in the Darknet community tend to only trade with the user within the community. The above trading behavior pattern may help us better detect those illegal transitions in Bitcoin network. In the Darknet community, hundreds of black market addresses are connected to each other and do not trade with other types of addresses. This shows that transactions between Darknet addresses are very frequent and traders are very cautious. In order to prevent tracking, addresses used in black market transactions are only used in the black market and not in other places, so Darknet addresses are not connected to other types of addresses. In addition, there are a large number of addresses in one black market transaction. We guess they may use mixed transactions to prevent tracking. Taking into account the characteristics of different types of transactions, we can similarly analyze the experiment results shown in Fig. 9. If we know *158nRB...* is an Exchange address, we can realize that all these addresses in the same community are of behaviors of Exchange nodes. Especially, the address *16tymJ...* appears in multiple different transactions. This phenomenon infers that *16tymJ...* may belong to a certain organization that is responsible for the exchange services. The addresses that have transacted with *16tymJ...* may be customers of the organization.

5.2. Bitcoin trace-ability

Recently, Bitcoin has become the most popular cryptocurrency for ransomware due to its anonymity mechanism. Hackers usually disclose their Bitcoin addresses for extortion. However, the regulators are not able to trace these malicious and illegal behaviors with these public Bitcoin addresses directly. With our proposed AWAP model, we can classify these disclosing Bitcoin addresses into different communities using the transactions in the public ledger of Bitcoin, which may provide some clues for mining Bitcoin crimes.

To study the trace-ability of AWAP model in Bitcoin, we use three disclosed Bitcoin addresses involving in ransomware as the starting addresses, and then recursively crawl those Bitcoin addresses trading with the starting addresses. Finally, we obtain 7882 Bitcoin addresses. We use AWAP to classify them, and set the number of detection communities from 2 to 1000. When the number of communities is set to greater than 500, the number of communities detected keeps stable at around 120. Thus, we finally classify 7882 addresses into approximately 120 entities. This means Bitcoin addresses in one entity exhibit extremely similar characteristics and cannot be classified anymore. The result of 1000 Bitcoin addresses is shown in Fig. 10.

In Fig. 10(a), we show three communities distinguished with different colors and shapes. The large red node in the middle is the Bitcoin address *12t9...* disclosed as the ransomware, and we use it as a starting address. In Fig. 10(b), we show five types of communities. We use address *115p...* and address *13AM...* as the other two starting addresses. No matter how much we set the number of communities, address *115p...* and address *13AM...* belong to the same community, and the address *12t9...* always belongs to another community. This shows that these three addresses do not belong to the same Bitcoin user, and may belong to different individuals or organizations. We can find that many addresses have transacted with the initial ransomware address. These addresses may be victims, such as the yellow triangle nodes and the pink star nodes. We also found many other addresses that belong to the same community as the ransom address, such as the red round nodes and the green square nodes. To reveal the identity, we can further track the red and green nodes, they may belong to the same organization as the ransomware addresses. In Fig. 10, there is no direct transaction between address *115p...* and address *12t9....* But Fig. 10(a) shows that there are many transactions between the red round nodes and the green square nodes because these two organizations are distributing and diverting the blackmailed bitcoins. The blue triangle nodes belong to an interesting community and are not like ordinary victims. Their trading behaviors are unique and can be divided into a separate community and the number is small, so the blue community deserves further study.

6. Related work

6.1. Address clustering

Heuristic clustering. Nakamoto and Bitcoin [1] propose the multi-input heuristic clustering methods to cluster Bitcoin addresses. They find that multiple inputs of a transaction are likely to belong to the same user. Reid and Harrigan [16] derive two topological utilize multi-input heuristic clustering to investigate the alleged theft in Bitcoins. Androulaki et al. [19] study the privacy provisions in Bitcoin using multi-input heuristic and shadow heuristic. They simulate a Bitcoin ecosystem within a university and the results show that almost 40% of users can be identified even though they use a new Bitcoin address for each new transaction. Nick [20] proposes two new heuristic clustering algorithms, consumer heuristic, and optimal change heuristic. They apply four heuristic clustering methods on a dataset collected from a vulnerability in Connection Bloom Filtering. They show the heuristic clustering algorithms can achieve 69% *accuracy* on average.

Data mining approaches. Jourdan et al. [22] develop a probabilistic model that accounts for five features including address

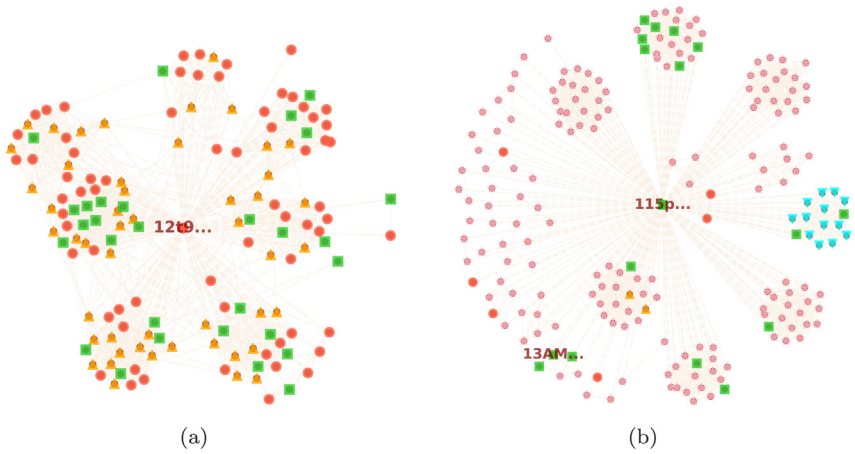

Fig. 10. Transaction trace-ability graph of ransomware addresses: (a) address *12t9...*; (b) address *115p...* and address *13AM....*

features, entity features, temporal features, graph centrality metric features, and motif features. Toyoda et al. [24] use a scraping-based method to collect more than 2000 HYIP operators' Bitcoin addresses and identify them based on the rate conversion technique and the sampling technique. Their method classifies 95% of the HYIP addresses correctly. Lin et al. [23] propose a classifier by adding new features, such as transaction time, to identify Bitcoin addresses. They train eight classifiers such as Logistic Regression, SVM, Random Forest, and LightGBM. Eventually, LightGBM achieves the highest Micro-F1 of 87%. Huang et al. [21] design an algorithm named BPC which is similar to k-means clustering to study the Bitcoin behavior pattern. Bartoletti et al. [2] train the RIPPER, Bayes Network, and Random Forest models on a real-world Ponzi schemes dataset. The best algorithm achieves 99% *accuracy*.

Deep learning-based methods. Shao et al. [8] propose a deep learning method achieving Bitcoin address–user mapping. They train a deep neural network to obtain address feature vectors and classify the Bitcoin addresses by calculating the distance between address feature vectors. Tang et al. [25] design a deep learning-based approach named PeerClassifier to capture the behavior pattern in the Bitcoin system. They extract sequence representation from node behaviors. Liang et al. [26] develop an address identification algorithm with high fault tolerance. They divide Bitcoin addresses into four types, exchange, gambling, service, and general, and use network representation learning to improve the classification performance.

6.2. Community detection

Community detection can be categorized into two different types. One only considers the topology of the graph, while the other comprehensively considers both the topological structure and node attributes. Blondel et al. [31] use a heuristic method to extract the community structure of the large networks based on modularity optimization. Yang and Leskovec [42] present an overlapping community detection method, BIGCLAM that scales to large networks of millions of nodes and edges. Clauset et al. [43] present a hierarchical agglomeration algorithm for detecting community structure. Perozzi et al. [44] propose a structure-only representation learning method, DeepWalk, and use local information obtained from truncated random walks to learn the latent representations.

Leskovec and Mcauley [45] propose a model for detecting circles that combine network structure as well as user profile information. They learn members and user profile similarity metric for each circle. A Bayesian probabilistic model(BAGC) for attribute graph clustering is proposed in Xu et al. [41]. The model provides a principled and natural framework for capturing both structural and attribute aspects of a graph, avoiding the artificial design of a distance measure. Yang et al. [46] develop CESNA for overlapping community detection which has a linear runtime in the network size. Liu et al. [36] treat a network with a dynamic system and uses the principle of information propagation to integrate the structure and contents in a network. Ruan et al. [47] design a mechanism for fusing content and link similarity. They present a biased edge sampling procedure and finally cluster an edge set with similar properties.

7. Conclusion

In this paper, we address the Bitcoin de-anonymity by studying the Bitcoin addresses classification to further understand user trading behaviors. To achieve this goal, we first parser the raw Bitcoin transaction data into the structured data and design a feature engineering pipeline to process node attributes. Then we construct an attribute graph with both the topological structure and the node attributes. Finally, we propose a community detection method based on adaptive weighted attribute propagation to cluster the Bitcoin addresses and further analyze the influence of different feature engineering strategies. Our AWAP model outperforms all the baselines on both public datasets and the Bitcoin transactions dataset. Our model can efficiently resolve the Bitcoin addresses classification, community detection, and ransomware Bitcoin address trace-ability. One future direction is to include node embedding techniques in the community detection model that can automatically extract useful information from input data. The other one is to explore bitcoin user's behaviors, further contribute to the healthy development of Bitcoin, and design privacy-preserving cryptocurrencies.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This work is partially supported by the National Key Research and Development Program of China (2018YFB2100304), the Zhejiang Lab (2021KF0AB04), and the Natural Science Foundation of Tianjin(20JCZDJC00610).

References

[1] S. Nakamoto, A. Bitcoin, A peer-to-peer electronic cash system, Bitcoin (2008) –URL: https://bitcoin.org/bitcoin.pdf.

[2] M. Bartoletti, B. Pes, S. Serusi, Data mining for detecting bitcoin ponzi schemes, in: 2018 Crypto Valley Conference on Blockchain Technology (CVCBT), IEEE, 2018, pp. 75–84.

[3] H.H. Sun Yin, K. Langenheldt, M. Harlev, R.R. Mukkamala, R. Vatrapu, Regulating cryptocurrencies: a supervised machine learning approach to de-anonymizing the bitcoin blockchain, J. Manage. Inf. Syst. 36 (1) (2019) 37–73.

[4] N. Christin, Traveling the Silk Road: A measurement analysis of a large anonymous online marketplace, in: Proceedings of the 22nd International Conference on World Wide Web, 2013, pp. 213–224.

[5] L. Zhu, Y. Wu, K. Gai, K.-K.R. Choo, Controllable and trustworthy blockchain-based cloud data management, Future Gener. Comput. Syst. 91 (2019) 527–535.

[6] D. Ron, A. Shamir, Quantitative analysis of the full bitcoin transaction graph, in: International Conference on Financial Cryptography and Data Security, Springer, 2013, pp. 6–24.

[7] C. Zhao, Y. Guan, A graph-based investigation of bitcoin transactions, in: IFIP International Conference on Digital Forensics, Springer, 2015, pp. 79–95.

[8] W. Shao, H. Li, M. Chen, C. Jia, C. Liu, Z. Wang, Identifying bitcoin users using deep neural network, in: International Conference on Algorithms and Architectures for Parallel Processing, Springer, 2018, pp. 178–192.

[9] M. Jourdan, S. Blandin, L. Wynter, P. Deshpande, Characterizing entities in the bitcoin blockchain, in: 2018 IEEE International Conference on Data Mining Workshops (ICDMW), IEEE, 2018, pp. 55–62.

[10] K. Gai, J. Guo, L. Zhu, S. Yu, Blockchain meets cloud computing: A survey, IEEE Commun. Surv. Tutor. (2020).

[11] K. Gai, Y. Wu, L. Zhu, M. Qiu, M. Shen, Privacy-preserving energy trading using consortium blockchain in smart grid, IEEE Trans. Ind. Inf. 15 (6) (2019) 3548–3558.

[12] K. Liao, Z. Zhao, A. Doupé, G.-J. Ahn, Behind closed doors: measurement and analysis of cryptolocker ransoms in bitcoin, in: 2016 APWG Symposium on Electronic Crime Research (ECrime), IEEE, 2016, pp. 1–13.

[13] P. Koshy, D. Koshy, P. McDaniel, An analysis of anonymity in bitcoin using p2p network traffic, in: International Conference on Financial Cryptography and Data Security, Springer, 2014, pp. 469–485.

[14] I.D. Mastan, S. Paul, A new approach to deanonymization of unreachable bitcoin nodes, in: International Conference on Cryptology and Network Security, Springer, 2017, pp. 277–298.

[15] P.L. Juhász, J. Stéger, D. Kondor, G. Vattay, A bayesian approach to identify bitcoin users, PLoS One 13 (12) (2018) e0207000.

[16] F. Reid, M. Harrigan, An analysis of anonymity in the bitcoin system, in: Security and Privacy in Social Networks, Springer, 2013, pp. 197–223.

[17] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G.M. Voelker, S. Savage, A fistful of bitcoins: characterizing payments among men with no names, in: Proceedings of the 2013 Conference on Internet Measurement Conference, 2013, pp. 127–140.

[18] Z. Zhang, T. Zhou, Z. Xie, Bitscope: Scaling bitcoin address deanonymization using multi-resolution clustering, in: Proceedings of the 51st Hawaii International Conference on System Sciences, 2018.

[19] E. Androulaki, G.O. Karame, M. Roeschlin, T. Scherer, S. Capkun, Evaluating user privacy in bitcoin, in: International Conference on Financial Cryptography and Data Security, Springer, 2013, pp. 34–51.

[20] J.D. Nick, Data-Driven De-Anonymization in Bitcoin (Master's thesis), ETH-Zürich, 2015.

[21] B. Huang, Z. Liu, J. Chen, A. Liu, Q. Liu, Q. He, Behavior pattern clustering in blockchain networks, Multimedia Tools Appl. 76 (19) (2017) 20099–20110.

[22] M. Jourdan, S. Blandin, L. Wynter, P. Deshpande, A probabilistic model of the bitcoin blockchain, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, 2019.

[23] Y.-J. Lin, P.-W. Wu, C.-H. Hsu, I.-P. Tu, S.-w. Liao, An evaluation of bitcoin address classification based on transaction history summarization, in: 2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC), IEEE, 2019, pp. 302–310.

[24] K. Toyoda, P.T. Mathiopoulos, T. Ohtsuki, A novel methodology for HYIP operators' bitcoin addresses identification, IEEE Access 7 (2019) 74835–74848.

[25] H. Tang, Y. Jiao, B. Huang, C. Lin, S. Goyal, B. Wang, Learning to classify blockchain peers according to their behavior sequences, IEEE Access 6 (2018) 71208–71215.

[26] J. Liang, L. Li, W. Chen, D. Zeng, Targeted addresses identification for bitcoin with network representation learning, in: 2019 IEEE International Conference on Intelligence and Security Informatics (ISI), IEEE, 2019, pp. 158–160.

[27] S. Ranshous, C.A. Joslyn, S. Kreyling, K. Nowak, N.F. Samatova, C.L. West, S. Winters, Exchange pattern mining in the bitcoin transaction directed hypergraph, in: International Conference on Financial Cryptography and Data Security, Springer, 2017, pp. 248–263.

[28] W. Chen, J. Wu, Z. Zheng, C. Chen, Y. Zhou, Market manipulation of bitcoin: evidence from mining the mt. Gox transaction network, in: IEEE INFOCOM 2019-IEEE Conference on Computer Communications, IEEE, 2019, pp. 964–972.

[29] E. Ravasz, A.L. Somera, D.A. Mongru, Z.N. Oltvai, A.-L. Barabási, Hierarchical organization of modularity in metabolic networks, science 297 (5586) (2002) 1551–1555.

[30] D.J. Watts, P.S. Dodds, M.E. Newman, Identity and search in social networks, science 296 (5571) (2002) 1302–1305.

[31] V.D. Blondel, J.-L. Guillaume, R. Lambiotte, E. Lefebvre, Fast unfolding of communities in large networks, J. Statist. Mech.: Theory Exp. 2008 (10) (2008) P10008.

[32] X. Li, B. Kao, Z. Ren, D. Yin, Spectral clustering in heterogeneous information networks, in: Proceedings of the AAAI Conference on Artificial Intelligence, Vol. 33, 2019, pp. 4221–4228.

[33] D. Kamuhanda, K. He, A nonnegative matrix factorization approach for multiple local community detection, in: 2018 IEEE/ACM International Conference on Advances in Social Networks Analysis and Mining (ASONAM), IEEE, 2018, pp. 642–649.

[34] Y. Tian, R.A. Hankins, J.M. Patel, Efficient aggregation for graph summarization, in: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, 2008, pp. 567–580.

[35] M. Qin, D. Jin, K. Lei, B. Gabrys, K. Musial-Gabrys, Adaptive community detection incorporating topology and content in social networks*, Knowl.-Based Syst. 161 (2018) 342–356.

[36] L. Liu, L. Xu, Z. Wangy, E. Chen, Community detection based on structure and content: A content propagation perspective, in: 2015 IEEE International Conference on Data Mining, IEEE, 2015, pp. 271–280.

[37] M. Fleder, M.S. Kester, S. Pillai, Bitcoin transaction graph analysis, 2015, arXiv preprint arXiv:1502.01657.

[38] H. Kalodner, S. Goldfeder, A. Chator, M. Möser, A. Narayanan, Blocksci: Design and applications of a blockchain analysis platform, 2017, arXiv preprint arXiv:1709.02489.

[39] L. Xu, M. White, D. Schuurmans, Optimal reverse prediction: a unified perspective on supervised, unsupervised and semi-supervised learning, in: Proceedings of the 26th Annual International Conference on Machine Learning, 2009, pp. 1137–1144.

[40] Y. Zhou, H. Cheng, J.X. Yu, Graph clustering based on structural/attribute similarities, Proc. VLDB Endow. 2 (1) (2009) 718–729.

[41] Z. Xu, Y. Ke, Y. Wang, H. Cheng, J. Cheng, A model-based approach to attributed graph clustering, in: Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data, 2012, pp. 505–516.

[42] J. Yang, J. Leskovec, 2013,

[43] A. Clauset, M.E. Newman, C. Moore, Finding community structure in very large networks, Phys. Rev. E 70 (6) (2004) 066111.

[44] B. Perozzi, R. Al-Rfou, S. Skiena, Deepwalk: Online learning of social representations, in: Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2014, pp. 701–710.

[45] J. Leskovec, J.J. Mcauley, Learning to discover social circles in ego networks, in: Advances in Neural Information Processing Systems, 2012, pp. 539–547.

[46] J. Yang, J. McAuley, J. Leskovec, Community detection in networks with node attributes, in: 2013 IEEE 13th International Conference on Data Mining, IEEE, 2013, pp. 1151–1156.

[47] Y. Ruan, D. Fuhry, S. Parthasarathy, Efficient community detection in large networks using content and links, in: Proceedings of the 22nd International Conference on World Wide Web, 2013, pp. 1089–1098.