Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

Simulation of Bitcoin Transactions to Identify Money Laundering Activities

Relatore:
Chiar.mo Prof.
Stefano Ferretti

Presentata da:
Bruno Battaglia

Sessione: II - primo appello
Anno Accademico: 2020/2021

“Ma io sono fiero del mio sognare,
di questo eterno mio incespicare..."
- Francesco Guccini

Contents

Abstract

The terms "blockchain" and "cryptocurrencies", with what revolves around them, have forcefully become part of the common jargon. At the same time, however, disinformation reigns around this increasingly popular world. We find ourselves dealing with a real economy which, like a self-priming process, has exponentially fueled the desire of people - even the "common" one - to become an integral part of a simple reality only in general perception. The full understanding of the subject, in fact, requires the prior acquisition of a multitude of notions and skills of information technology. Nevertheless, in the "vulgate", this world is faced with superficiality. Consequently, the more these themes become "familiar" in the social, the more context is used the desire to find the way of the aforementioned technologies to enslave them for "not very noble" purposes.
What is it that attracts this form of thought? Since, by now, numerous illegal activities have completely moved to the Web, the (pseudo) anonymity guaranteed by the blockchain is an incentive for which they want to keep borderline operations hidden from the point of view of legality. if, within a blockchain network (in the specific case: Bitcoin), it is possible to identify money trafficking that someone tries to "clean up". The activity of money laundering, moreover, is extremely widespread in the vast universe of crime and, as already mentioned, the basic characteristics of the blockchain lend themselves - at least, in theory - to this practice.
The thesis does not claim to be exhaustive about the technical peculiarities of the Bitcoin structure; the same intends to work at a higher level, and, through a simulator, generate the highest layer, that is the one given by a graph representing the social network. This, in particular, is characterized by nodes that identify the addresses and oriented arcs that move from the sender to the recipient of the transaction. The weight of the transaction, on the other hand, is offered by the amount spent in bitcoin. Both nodes and transactions can carry additional features such as timestamps, membership or membership in a mixer.
The use of techniques for the analysis of social networks is the key to the interpretation of the relationship between data. Indices relating to degree and its distribution, such as those of centrality, can prove to be crucial. However, it should be

borne in mind that, in parallel with technology, crime is evolving. This assumption opens the way to the analysis of multiple problems.
First, each user has the opportunity to generate a new address for each transaction; in fact, the number of addresses generated has grown in a direct proportion, in recent times, to that of transactions. In relation to this, a mechanism has been implemented within the thesis project generation of clusters based on the state of the art and on certain heuristics already demonstrated functioning.
A second problem concerns the existence of so-called "mixer mechanisms", which can be of two types: centralized and decentralized. Consequently, in the graph, they will result connections that, on closer inspection, turn out to be fictitious. Still in the context of the thesis, they were made the subject of evaluation some characteristics that such systems, precisely because automated, they possess; through a reverse engineering process, among other things, it is possible to identify patterns to search for within the same graph. An other complication relates to the computational power required: the download process and data analysis, although parallelized processes, requires, in fact, a considerable outlay of time; by virtue of this consideration, it was decided to simulate an extremely smaller network which, however, fully reflects all the characteristics of Bitcoin: in this way, it has become relatively easy to work on it rather than on the original. The approach used could, perhaps, be called "investigative": it was hypothesized, in fact, that a node could be malicious and, therefore, subgraphs were generated that could actually lead to money laundering activities. In the chapters belowe, there will be discussed:

- **Chapter 1**:
  - *Bitcoin*: a bit of histotry about blockchain and, deeply, the structure of Bitcoin. In particular, whats is and what contains the block header, how to validate a block, the mining operation, a brief explanation of Merkle trees and why they are used in this technology, the concept of fork and how a transaction is composed;
  - *Social Network Analysis*: the main concepts of this subject, helpful to the work of the dissertation. Get into the specifics, there will be explained different kind oh graph, their peculiarities, and some parameters to elaborate the network situation. Furthermore, different types of path found within a network will be analyzed and described.
  - *State of the Art*: for the last section of the chapter, the state of the art is of fundamental importance. It lays the starting points for the entire work. Specifically, in the section, some scientific articles, that have applied the social network analysis to money laundering activities, will be

dissected albeit not in blockchain environments;

- **Chapter 2**: it deals with some problems that it is obligatory to face and solve in order to reach the purpose of the dissertation. One of the most ingenious methods that have been built to remedy the privacy that was no longer total was that of mixers. People can rely on both third party services and organize themselves to generate multi-input and multi-output transactions in order to hide any tracking.Moreover, as stated before, there is the possibility to always create a new address and this makes the problem more complicated to solve. In fact, first of all, there is the need to cluster the addresses of a single person in order to decrease the size of the network and make it easier to track transactions.
  For these reason, in one of the section of the chapter, there will be analyzed some articles that explain the two famous heuristics created to solve the problem just mentioned.
  In the last section foresees the expansion of the DiLeNa tool to which the possibility of identifying clusters has been added thanks to the implementation of the heuristics presented previously.

- **Chapter 3**: although the DiLeNa tool is very efficient and allows processes to be parallelized, the cost in terms of time, especially on blockchains such as Bitcoin, is really high. For this reason, in this chapter we will see how a simulator has been implemented and how it allows to reproduce the highest layer, that is, the one that concerns the analysis of social networks, discarding all the basic architecture of the blockchain.

- **Chapter 4**: in the last chapter the embryonic algorithm that is able to identify a transaction aimed at money laundering is presented (if it has certain conditions) and a simulation of it is illustrated. By applying an investigative approach that foresees the hypothesis that a node is dangerous, a lot of attention will be paid to what will be the size of the sub-graph to be examined to study the movements of the suspect node.

Chapter 1

Blockchain and Social Network Analysis

In this section, we're gonna introduce the blockchain and the social network analysis in order to discuss their use cases, their structure and the importance they have to the entire project we're going to discuss in all the chapters below.

- *Blockchain*: the first blockchain was introduced in 2008 by a person or a group of people under the psudonym of Satoshi Nakamoto in their whitepaper about Bitcoin. The goal was to design a structure that would allow users to spend without a trusted authorithy or a central server. In fact, the blockchain is described as "Trustless and fully decentralized peer-to-peer immutable data storage." [22].
  The name of the first cryptocurrency is **bitcoin** and it was created to manage the problem of the *double-spending*, which is a potential flaw in a digital cash scheme where the same single digital token can be spent more than once [6].
  The biggest peculiarity of blockchains is that they are *immutable* and *public data structures*.
  The *ledger* is defined by blocks linked together in chronological order and whose integrity is guaranteed by the fact that each block contains a hash code identifying the previous block, a timestamp and transaction data. The timestamp is the guarantee that the transaction data existed when the block was already present in the ledger.
  Since each block contains the hash reference of the previous block, this forms a chain. This is very important and allows to justify the immutability property because, even if only one block were modified, this would involve the modification of the entire chain [22].

- *Social Network Analysis*: the subject deals with the analysis of any kind of networks to understand the relationship between the interconnected elements, study the behavior of a given network, regardless of the type of the same.
  The analysis of social networks is a methodology used to model and find structural approaches for analyzing the relationships that exist between individuals, groups or institutions.
  The advantage of seeing reality from this point of view is that focus is putted on the interaction rather than on the behavior of the individual. Network analysis allows to examine how the configuration of networks affects the functioning of individuals.
  It also permits to predict how a network have evolved, discover patterns and make prevision about the future structure of the network. [25]

1.1 Bitcoin

The first and the most popular blockchain is, for sure, Bitcoin. As stated before, it's composed of blocks and each of them is divided into headers and sets of transactions.

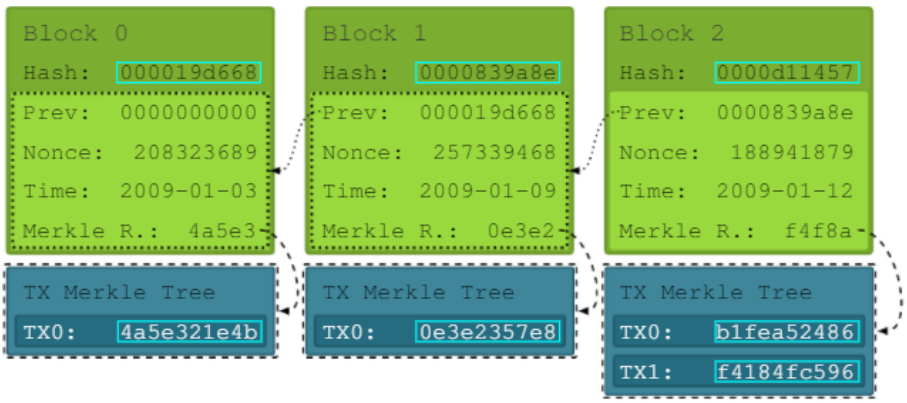


Figure 1.1: First three blocks of a blockchain. The only block with a real transaction is the block 2. Block 0 and 1 have only one transaction, called **coinbase**. A coinbase transaction is the first transaction that took place in a Bitcoin block and it is not the result of a payment between two people. It is a special transaction of the genesis block that formatted reward transactions for miners.

1.1.1 Header

The header include the data reported below [19]:

- *Version*: the version of the used software;
- *Hash*: the hash code of the previous block. The first block of the blockchain is called *genesis block;*
- *Nonce*: is an 8 byte value that is added to the block so that the output of the hash function varies so that it is less than the target value. The value is recalculated until the hash of the block contains the required number of zeros;
- *Bits*: encode the network target difficulty. Cryptocurrency difficulty is important since a high difficulty can help secure the blockchain network against malicious attacks. Difficulty is saved in encoded form on the block, is based on the hashing power of the network and is updated every 2016 blocks [16];
- *Timestamp*: the timestamp of the last transaction is generated with the Unix hex timestamp and not the timestamp UNIX;
- *Merkle R.*: the hash code of the all transactions' hash code inside the block;

Under these conditions, if a node sent a modified block, there would be an integrity error and it would not be added.

1.1.2 Block validation

Each block has an unique hash code and each oh them are generated through a system called Proof of Work (PoW). With the proof of work, there is the need to solve a complex mathematical problem that allows the generation of the new block. This process is know as **mining**.
To be more specific, a node collect the new transaction generated but not validated and suggests to the network what the new block should be. Using the hash function to estimate the output until it is less than the target bits value present in the header. The first node that solves the block transmits it to the network where it is accepted as the next block in the chain.
After a block has been solved, the network uses the hash of the block as identifier and upload the bits value. Miners are incentivated to find valid blocks for two main reason [16]:

- Fees: fees assigned by Bitcoin users to transactions, which are included in a block, are added to the block reward;
- Block reward: each block has a transaction called coinbase. This transaction goes to an address of the miner. The first reward was of 50 BTC.

If all bitcoin would have been mined, fees would be the only reward for miners.

Fork

It is possible, for different nodes, to validate several blocks at the same time, thus leading to a bifurcation of the chain. In this case, miners work to validate blocks on both bifurcations of the chain, but as soon as a new block is validated and added in one of the two, all miners working on the other move to the one to which a new block has been added, thus transforming the abandoned block into an "orphan block". This is because the miners' goal is to extend the chain in length [16].
A fork introduces changes in the blockchain software protocol. Whit a fork, the networks divides into two chains. There are different kind of forks [9]:

- *Soft fork*: a soft fork is backwards compatible. The updated blockchain is responsible for approving transactions, but nodes that are not updated will continue to consider the new blocks valid. This only works in one direction; the updated blockchain does not recognize nodes that have not been updated. For a soft fork to work, most miners need to upgrade. The more miners accept the new rules, the safer the network will be after the fork. They are generally used to implement software updates;

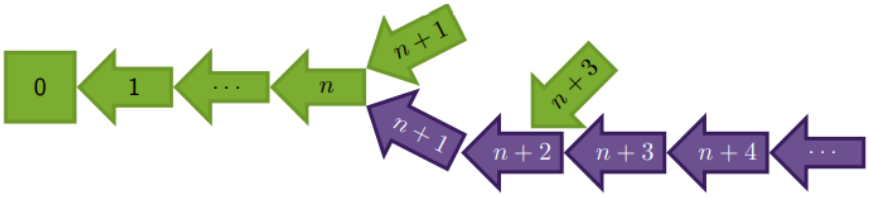


Figure 1.2: How soft fork works.

- *Hard fork*: a hard fork is a radical change in software that requires all users to update to the latest version of the software. Nodes on the previous version of the software will no longer be accepted on the new version. A hard fork is a permanent divergence from the previous version of the blockchain. If there is no unanimous consensus on the new version, this can result in two blockchains using a variant of the same software.

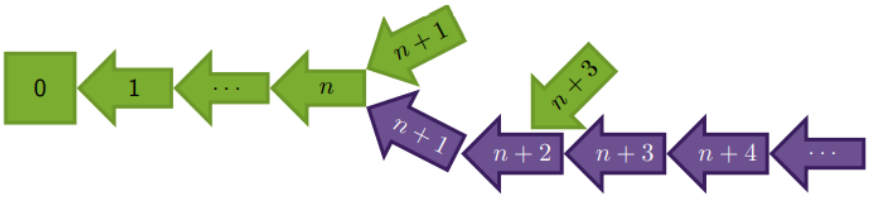


Figure 1.3: How hard fork works.

1.1.3 Transaction

As already stated, transactions use Merkle trees.

> In cryptography and computer science, a hash tree or Merkle tree is a tree in which every leaf node is labelled with the cryptographic hash of a data block, and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes. Hash trees allow efficient and secure verification of the contents of large data structures. Hash trees are a generalization of hash lists and hash chains.

Merkle tree has a computational cost equals to O(log n) so, it allows fast verification of transactions.
Bitcoins contain their owner's public key (i.e. address). When a user A transfers money to user B, he renounces his ownership by adding B's public key (his address) to the coins in question and signing them with his own private key. It then transmits these coins in a message, the **transaction**, through the peer-to-peer network. The rest of the nodes validate the cryptographic signatures and the amount of digits involved before accepting it [5].
So, it's possible to say that a transaction involved one or more actors in input and one or more actors in output. To send a transaction, each input needs a **UXTO**, that stands for *Unspent Transaction Output* and indicates the amount of cryptocurrency exchange remaining after performing a cryptocurrency transaction. The mechanism is similar to the change someone receives after making a cash transaction at a phisical store.
UTXOs are processed continuously and are responsible for the beginning and end of each transaction.

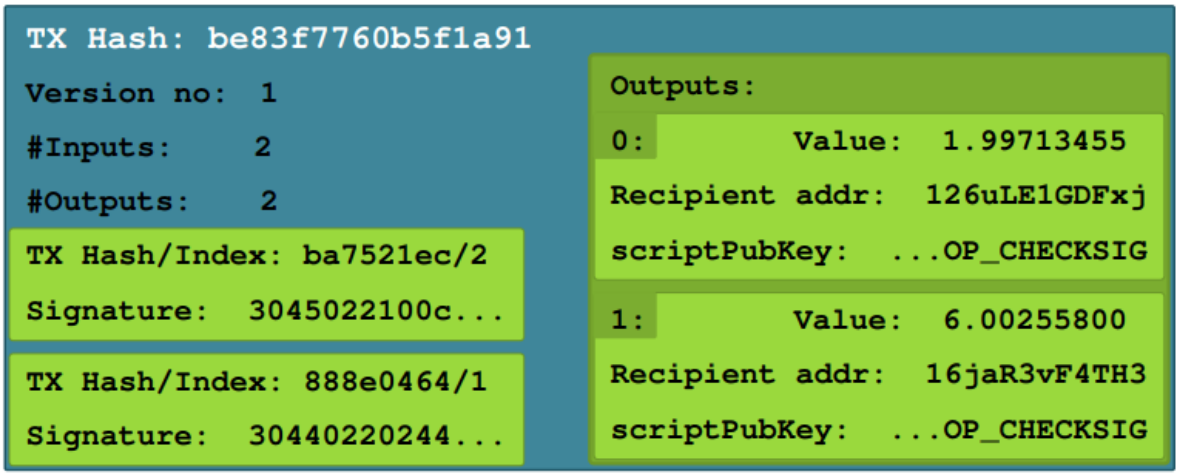


Figure 1.4: Transaction's structure.

Referring to figure 1.4, a transaction is defined as [16]:

- *Transaction hash*: hash code for the current transaction;
- *Number of inputs*: number of addresses in input;
- *Number of output*: number of addresses in output;
- *Signature*: for each input is provided a signature that proves that the signer is authorized to spend the output;
- *Output*:
  - Index: the number of the transaction;
  - Value: the amount in bitcoin;
  - ScriptPublicKey: used to verify the signature provided by the sender.

1.2 Social Network Analysis

Within the SNA there are two main families of networks:

- *Blockmodeling*: block model capable of identifying clusters;
- *Dynamic networks*: block model that can identify clusters; networks that grow and evolve in form based on time.

Graph theory does not focus on social networks in the proper sense of the term but represents a real concept based on weights.
Another known type of network concerns the algebra of matrices, or the representation of a **graph in matrix form** in which the connections are represented through values numbers inserted in a matrix.

1.2.1 Main concepts

There are some words to explain in order to understand the work of the subsequent sections.

- *Actor*: social entity, identified by a node;
- *Diade*: representation of two actors and bonds;
- *Triad*: representation of three actors and ties;
- *Group*: collection of all the actors on which bonds are measured;
- *Ties*: ties between pairs of actors (friendship, business etc.);

There are also different kind of network, with specific features.

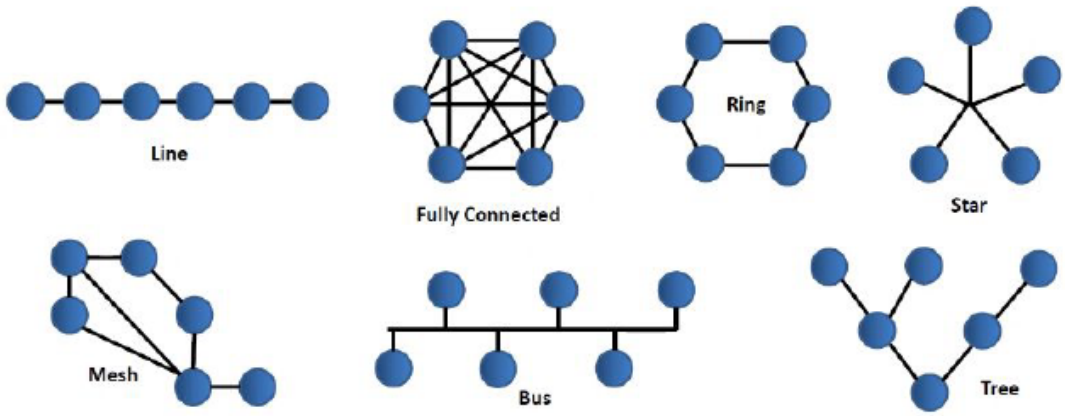


Figure 1.5: Known network models.

- *Line*: the first and the last node have one link, while the others have two;
- *Fully Connected*: all elements are connected to each other. Useful for identifying clusters within social structures;

- *Ring*: also called circle. All elements have the same weight , so there is no node more central than others;
- *Star*: network characterized by a topology that provides for the presence of a central node, to which all the other nodes are directly connected;
- *Mesh*: set of other networks;
- *Bus*: the nodes are linked by peculiar relationships, so nodes are directly connected to a common half-duplex link called a bus;
- *Tree*: characterized by the fact that several distinct and non-intersecting linear chains can depart from each node, thus creating a multilevel structure. Also in this type of topology, for each pair of nodes there is only one connection path; each node is connected to a single upper level node (parent node) through a single branch and to one or more lower level nodes (child nodes) through one or more dedicated branches (branch). The node from which the whole topology originates is also called the "root node" while the terminal nodes are called "leaves".

1.2.2 Typology of graph

In the graph theory, it's possible to distinguish between different kind of graphs. Each of them have some features that identifies this entity. A graph is characterized by a set of N nodes and a set of L links. Mathematically, a graph is described by the equation $G = (N, L)$.

- *Oriented graph*: also called *Digraph*. It is said to be oriented if for the ordered pair $\langle n_i, n_j \rangle$ there is such a bond to be written as $n_i \rightarrow n_j$. The maximum possible number of links is: $g \cdot (g-1)$, where g is the number of the nodes;
- *Undirected graph*: it is only interesting to know if there are links, without indicating the direction. The maximum possible number of links is: $\frac{g \cdot (g-1)}{2}$, where g is the number of the nodes.

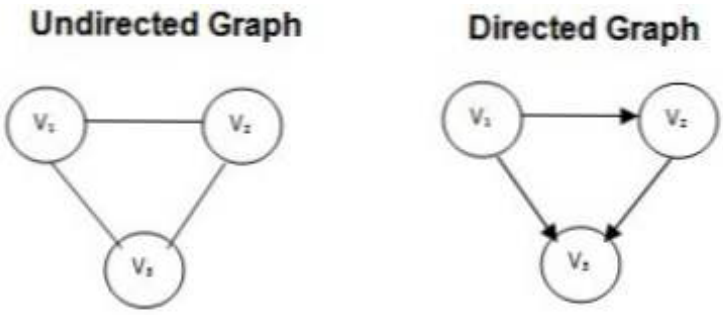


Figure 1.6: Difference between oriented and undirected graph.

1.2.3 Useful Parameters

To work with a network and especially to analyse it, it's necessary to understand some concepts that allow us to make sense at some peculiarities of the network.

Degree

The degree of the node is indicated by $d(n_j)$ and is the number of lines ending in the node itsel, so the number of adjacent nodes. The degree ranges from 0 to $g-1$. The degree is very important because it tells us how influential a node is within the network.
Adding the degree of all the nodes, we obtain that the degree can be calculated by multiplying the number of arcs by two. Mathemathically, it can be write as:

$$\sum_{i=1}^{g} d\left(n_i\right) = 2 \cdot L$$

To check how much a node affects a graph the *medium nodal degree* is the parameter to calculate. It is a statistic that reports the average degree of the nodes in the graph and is given as:

$$\bar{d} = \frac{\sum_{i=1}^{g} d\left(n_i\right)}{g} = \frac{2L}{g}$$

In a directed graph, there are two important values:

- *In-degree*: a parameter that indicates how many arches enter into a node;
- *Out-degree*: a parameter that indicates how many arches come out from a node.

Variance

Variance is an index used to determine how a network is regular. The more the variance tends to 0, the more the network is regular.

- In the hypothesis that the degree $d(n_i) = 0$, the graph is said to be *d-regular* and the variance is calculated as:

$$S_D^2 = \frac{\sum_{i=1}^{g} \left(d\left(n_i\right) - \bar{d}\right)^2}{g}$$

- In the case of a *non-d-regular graph*, so with variance not equals to 0, the variance is calculated as:

$$S_D^2 = \frac{\sum_{i=1}^{g} \left(d\left(n_i\right) - \bar{d}\right)^2}{g}$$

Density

It describes how many bonds there are within the graph compared to how many bonds they could really be there in total. Trivially it tells us how much the graph is “dense” with bonds.
It assumes values between 0 (if no node is connected) and 1 (if all nodes are connected to each other). The density between the two extremes belongs to the intermediate graphs.
A low density indicates a poorly interconnected network and vice versa

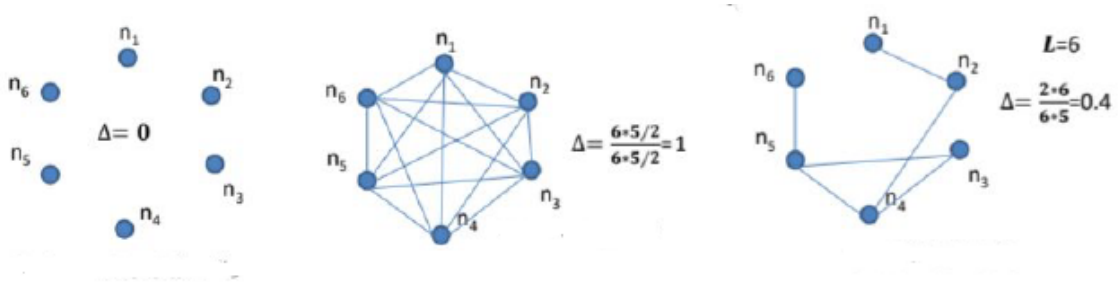


Figure 1.7: From left to right: graph with density equals to 0; complete graph with density equals to 1; intermediate graph with density equals to 0.4.

Connectivity measures

There are some metrics that allow us to provide information on the type of path that the arcs follow within the network. In particular:

- **Walk**: is a sequence of nodes and lines that start and end with nodes and in which a node is incident with the lines that precede and follow it. The length of a path is given by the number of arches to be covered. The *inverse* of the path is the reversed transposition of the walk. In other words, the path is rode in reverse;
- **Trail**: it is a path in which all the lines are distinct even if the nodes could be included more than once;
- **Path**: it is a path in which all the nodes and all the lines are distinct. If there is a path between two nodes, they are called *reachable nodes*;
- **Closed walk**: a path that starts and ends in the same node.

A graph is called *connected* if there is a path between any pair of nodes. In a connected graph, all nodes are reachable. If there is an unreachable node in a graph, it is said to be *disconnected*. The nodes of a disconnected graph are divided into two subgroups: each is defined as a *component*.
Other important metrics are:

- *Geodesic distance*: the shortest path between two nodes while instead, two unreachable nodes have an indefinite distance;
- *Eccentricity*: also referred to as an association number, it is the largest geodesic distance between a specific node and any other node;
- *Diameter*: is the largest geodesic distance, i.e. the maximum distance dividing two or more nodes

1.3 State of the Art

The first step is to understand if and how to be analyzed of social networks should be analyzed in order to recognize a round of dirty money.. At this stage, the focus was not on the blockchain, but on any generic social network.

In the article proposed by Andrea Fronzetti Colladon and Elisa Remondi [12] the database of a bank was examined, which must record transactions of an amount equal to or greater than 15,000 euros or transactions of a smaller amount whose sum exceeds this threshold.
The network was built by analyzing the transactions (network links) between sellers and debtors (network nodes). Nodes that are both sellers and debtors have

both inbound and outbound connections. The degree is weighted and the weight is determined by the amount of money moved in the transaction:

- Transactions with amounts less than 50k euros;
- Transactions with amounts between 50k and 250k;
- Transactions with amounts greater than 250k.

In addition, two further networks have also been created:

1. The first one is based on the economic sector in which the company of a node operates (construction, metallurgy, automobiles, consultancy);
2. the second one is based on the percentage of crime by geographic area.

Starting from the first network, some centrality measures were considered: in-degree, out-degree, all-degree, closeness, betweenness, network constraint. A parameter (missing id) has been added to these, indicating the absence of some data (eg seller, representative or debtor) within a record. The starting hypothesis was that more central nodes, with a greater number of connections and with a greater presence in transactions, have a higher risk profile.

The results on the first graph showed that degree indices are important predictors since they have multiple central nodes associated with risky profiles. Betweenness also produced similar results. On the other hand, closeness did not seem significant.
Network constraint has shown how the most open ego-networks should capture the attention of the analysis. Unlike the first network, the missing id was found to coincide with nodes potentially at risk. Betweenness, on the other hand, turned out to be less predictive. Hence, having worked on more graphs led to greater contributions in terms of results.
In the figure 1.8 , the blue nodes are those involved in money laundry activities. No clustering action was required following the analyzes seen.

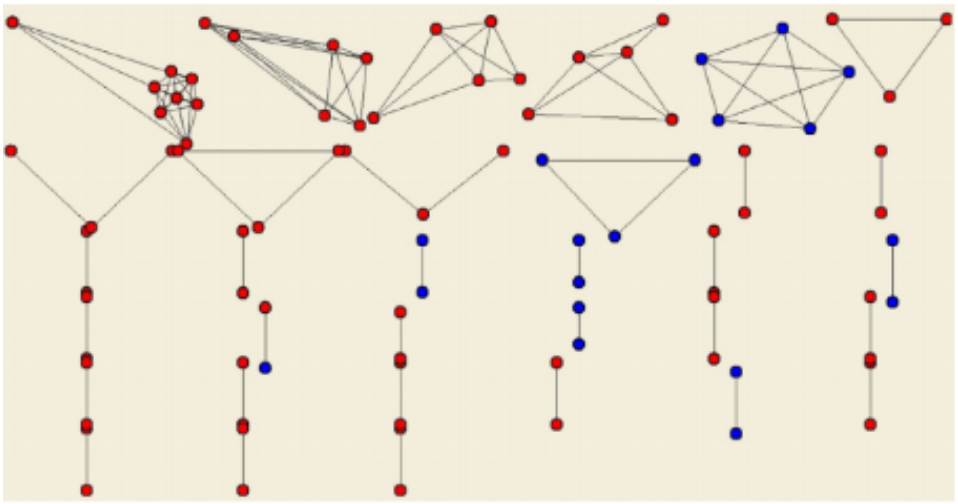


Figure 1.8: The clusters are represented in blue

This system considers two types of transactions: large bank deposits and international fund transfers. The resulting model is a graph in which the nodes represent the subjects involved in the transfer and the arcs the link between the subjects. The link can be of two types: direct (transactions) and supplementary (groups accessing the same account). An example in the image below.

Another great contribution is given by the ArXiv article [27] where the authors built an automated system based on social network analysis and machine learning techniques based on the data found in an AUSTRAC report.
This system considers two types of transactions: large bank deposits and international fund transfers. The resulting model is a graph in which the nodes represent the subjects involved in the transfer and the arcs the link between the subjects. The link can be of two types: direct (transactions) and supplementary (groups accessing the same account). An example in the image below.

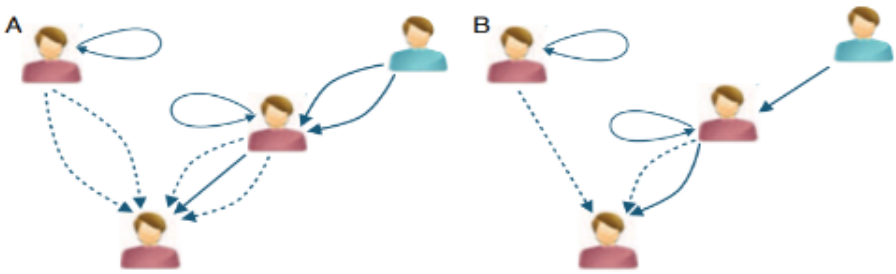


Figure 1.9: Example of multiple access

Some arcs are weighted because if different groups access the same account, each group will be connected to the others thus forming a clique.
The goal is to identify groups with suspicious behavior using the k-steps neighborhood, with k equal to 3 to allow you to connect even groups that are not directly connected but rely on a third party.
The supervised learning part is entrusted to random forest and SVM. Once extracted, a group must be validated in order to avoid false positives and a k-fold-like algorithm has been implemented for verification.

Chapter 2

Transaction's anonimity

One of the main reasons why blockchains have reached such peaks in popularity, as to be on everyone's lips, is certainly the factor linked to anonymity. But as people's vision has evolved, so too has the awareness that anonymity is not absolute and that, through clustering processes or retracing the paths carried out, it is possible to identify which bitcoins belong to a particular user.
For this reason, systems have been worked on to increase the anonymity of users to the point of complicating both the clustering techniques or making them lose track of their bitcoins, replacing them with bitcoins belonging to many other addresses. Specifically, we will focus on two mechanisms: *mixer* and *coinjoint*.

2.1 Mixer

As reported from the European Business Review: "*Bitcoin mixers, also called tumblers, "washing" or "laundries", are services that allow you to get rid of the history of previously conducted coin transactions.*" [31].
In other words, the main goal is to increase the privacy of users by losing track of the path of the coins sent by the users themselves.
The definition of mixer derives from the fact that these services mixes, in fact, different quantities of coins sent from separate addresses and send a random quantity of bitcoins at each address. This causal process is repeated until the total amount of coins is returned to the user's wallet.
There exists two macro-categories of mixers:

- *Centralized mixer*: a mixing service is called a centralized mixing service if it relies on a central server to perform the mixing [35]. These are services that accept users' cryptocurrencies and return totally different ones in order

to disperse the traces between the money sent and the user.
This category applies a commission that varies in percentage in exchange for the service offered. It is evident that more people adhere to the use of these services and more strength is gained by the mixer.
But, like everything, there is the downside. In this specific case, although greater privacy is the basis of tumblers, the security and privacy offered by these third party services are questionable. As a user, you need to entrust your funds to the mixer. There is no guarantee that these mixers will return your funds.
In terms of privacy, centralized mixers have access to your Bitcoin and IP addresses. Ultimately, they know which address sent and received which coins and likely keep records. They may decide to sell this information, or they may be forced by law to share the data, compromising your privacy. In an ideal centralized mixer, all this information about the user is deleted [31].
The main characteristics of the mixers, therefore, can be summarized as follows [17]:

- **Service fee**: a price to pay for using the services. Usually it's between 0.4% and 5%;
- **Delay**: if the mixer operated instantaneously, the transactions to mix the bitcoins would be easily identifiable. For this reason, mixers allow the user (or have default parameters) to set a time within which mixing operations must be performed;
- **Maximun number of address**: some mixers allows to separate one input into multiple outputs and, often, allows to set a different delay for each of these outputs. Clearly, this makes demixing much more dispensable.

As reported by The World Financial Review [32] and The European Business Review [31], in 2021 some of the most commen services are: BitMix; CoinMixer; BitcoinMixer, Blender; Bitcoin Laundry, Privcoin and others.
Thanks to the contribution of a study conducted by the University of Korea [17], some of these services have already been studied and the characteristics mentioned above have been extrapolated.

Mixing service	Mixing service fee	Delay	Max_output_address
CoinMixer	1 - 3%	~120h	5
BitcoinBlender	1 - 3%	~99h	10
CryptoMixer	0.5 - 3%	~48h	10
BitMix	0.4 - 4%	~24h	5
PrivCoin	0.8 - 3.8%	~24h	10
Bitcoin Fog	1 - 3%	~48h	20
BitCloak	1 - 3%	~8h	1
Bitcoin Mixer	1.5%	~24h	10
Helix	2.5%	~24h	5
Helix light	2.5%	~6h	5

Figure 2.1: Most common mixers' features

- *Decentralized mixer*: decentralized mixers are peer to peer mixing services. These mixers don't depend on a central server, remedying the problem of centralized mixers. The most common name by which these mixers are identified is **Coinjoin**.

2.2 Coinjoin

CoinJoin transactions were initially proposed in 2013 by Bitcoin developer Gregory Maxwell in an intervention in a blog [7].
The idea is that multiple parties coordinate to create a transaction, each providing the desired inputs and outputs. As all inputs are combined, it becomes impossible to say for sure which output belongs to which user. Of course, the greater the number of users in the pool, the greater the randomization.
It is clear, however, that unlike the mixing operations, the coinjoin practice can be implemented without turning to a third party service, which, as it is automated, can be studied through reverse engineering processes. The coinjoin practice, on the other hand, makes everything more complicated.

To better understand the mechanism of coinjoin, in the following lines we will see a practical example of this protocol. In the figure 2.2, there are four users with the will to interrupt the link between transactions. They need a coordinator o can coordinate each other and announce inputs and the outputs they want to include in the transaction.

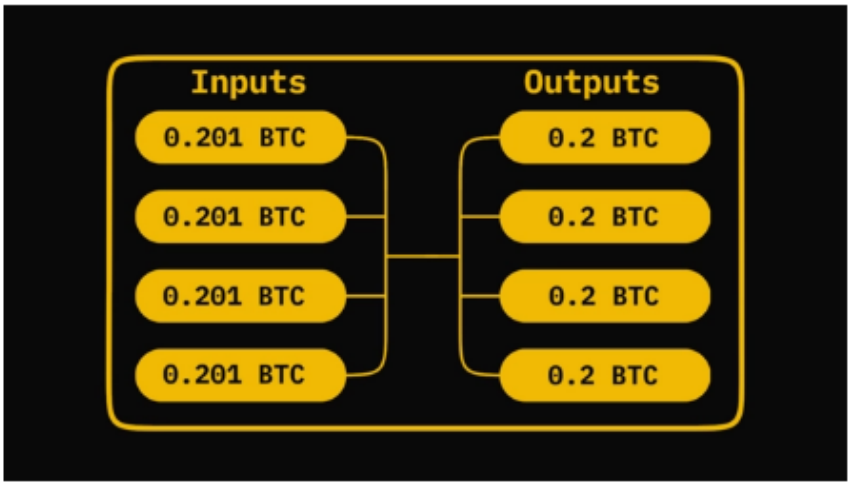


Figure 2.2: Most common mixers' features

The coordinator collects all the information, generate a new transactions and then he asks all the participants sign the transaction. Once the transaction is signed, there is no possibility to modify it and, consequently, the coordinator can't defrauded the others members.
In other words, the transaction is like a black box where UTXO are destroyed to generate new ones.
One important thing to think about: *no one can ensure that the participants are actually four*. There is the possibility that one single person are sending found from his addresses.
As Gregory Maxwell says, the output value is uniform, but if not, it is quite easy to recreate distinct subsets. If, on the other hand, n participants produced n uniform outputs, it would be impossible to subdivide inputs and outputs into subsets. Since the minimum number of participants is two, it is assumed that behind a CoinJoin there are at least four outputs. Added to the uniform output is the fact that the higher the number of participants, the more effective the CoinJoin becomes.

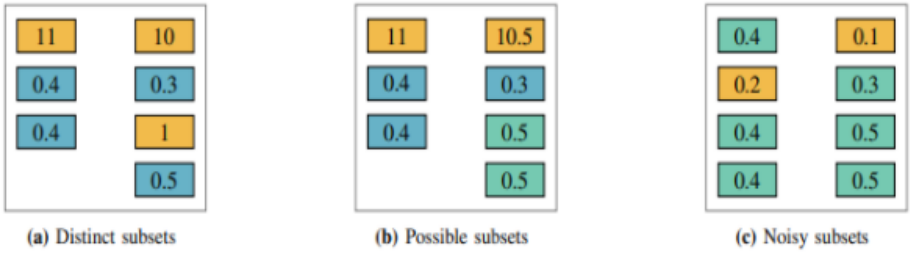


Figure 2.3: Identification of a coinjoin transaction

Despite this, creating a transaction with uniform output would mean having a transaction with a characteristic that distinguishes it from all the others, characteristics that would allow you to easily understand that behind it there is a CoinJoin operation. To this it must be added that 90% of transactions within the blockchain have two or fewer outputs, so the more the number of participants increases, the more the transaction tends to be unique.

About the most popular non-custodial mixers, Wasabi Wallet and the Samourai Whirlpool are the most know for sure, so that in May 2019, Whirlpool exceeded 4,784 BTC [18].

2.3 De-anonymization

Taking Bitcoin into analysis, it has been shown that the number of addresses has grown directly with the number of transactions. In order to simplify the representation of a blockchain in the form of a graph, it is therefore necessary to carry out clustering operations in which a cluster is generated containing a series of addresses belonging to the same person.

To better understand the problem, we refer to the image below, which illustrates how, since the end of 2013, the number of addresses per month has been growing more and more, even exceeding that of transactions. [14]

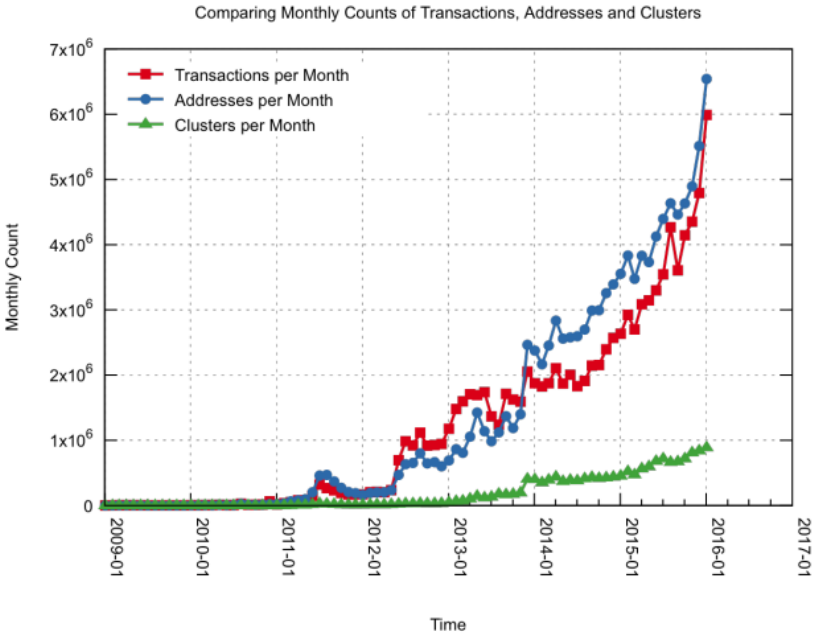


Figure 2.4: Growth of addresses, clusters and transaction per month

The article in the IEEE Transactions on Systems journal offers interesting insights.
First of all, two heuristics are proposed:

1. *Multi-input heuristic*: as stated by Nakamoto himself, it is inevitable that the link between addresses is known in multi-input transactions. Ex: transaction T1 has A and B as inputs while T2 has B and C as inputs. On the basis of this heuristic, the addresses A-C belong to the same entity. This heuristic could overestimate the cluster size, i.e., a multi input transaction in which the assumption that the transaction is made by a single person is incorrect [16];

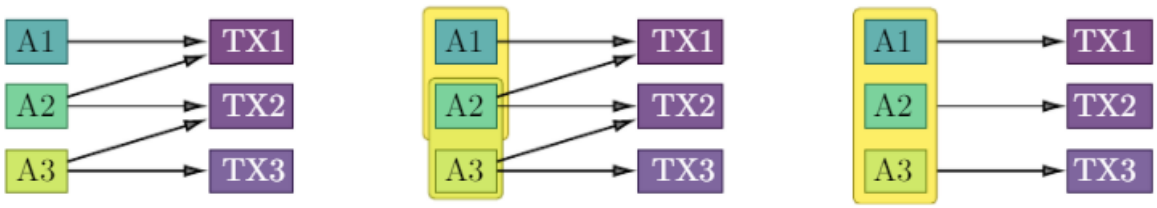


Figure 2.5: Change address mechanism

2. *Change address heuristic*: it is based on the assumption that a transaction will have to be spent in its entirety. Let's suppose Bob has 50 BTC and wants to pay 0.5 BTC to Alice, he will create a transaction with two outputs to which he will pay 0.5 BTC to each (without considering the fees), one belonging to Bob while the other will be the exchange one. Bob's two addresses must be clustered together, but the problem is that it is not known which of the two outputs is the exchange address.

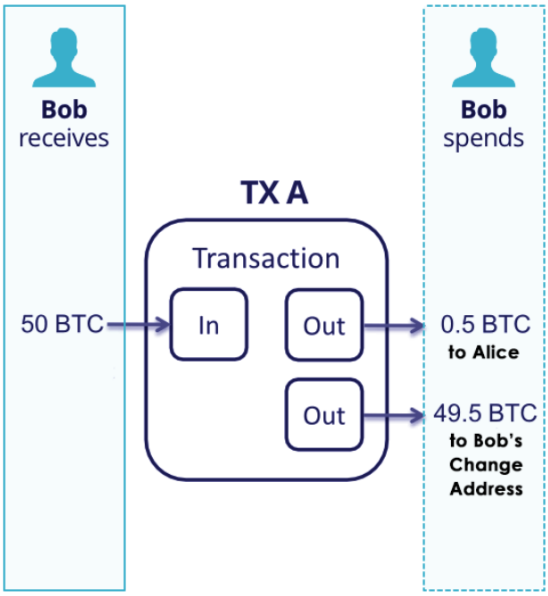


Figure 2.6: Result after using multi input heuristic

The study continues by identifying some characteristic patterns reported verbatim:

- *Peel Transaction*: regardless of the input addresses, there will always be two at the output. The first belongs to the receiving party, while the second to the sender and is used as a change;
- *Sweep Transaction*: multi input in one output;
- *Distribution Transaction*: some addresses in input and more than three in output;
- *Relay Transaction*: one input and one output;

- *Self-Spending Transaction*: an input address also appears as an output;
- *Peeling Chain Transaction*: a sequence of peel transactions.

Unlike many other works, the authors do not use a graph representation system, but a tabular form. They believe it is less computationally expensive and allows for better pattern detection.
The highest effectiveness is obtained by pre-processing the data using the two initial heuristics and then trying to create more solid clusters by identifying the proposed patterns.

Further insights into the two aforementioned heuristics are available in the doctoral thesis of Dr. Let us deepen the two heuristics through the doctoral thesis T. Neudecke. [23]
Multi-input heuristic (defined H1) is quite intuitive and effective since a transaction with multiple input addresses needs to be signed with the private key corresponding to the public key of all inputs. Assuming the transaction is created from a single address, all inputs go to the same person. The cluster determined by a transaction with the above characteristics is defined as Ct = input (t). The author believes it can also be used for the management of CoinJoin.

About the change address heuristic, discussed above, the author believes it is essential to add some conditions to try to identify which is the exchange address:

1. It is the first time that the address oj (t) is used;
2. Transaction t is not a coin generation transaction;
3. The transaction has no self-change addresses (addresses that appear both in input and output);
4. Condition 1 is valid for one address only or j (t).

Of course, despite the new conditions, the new change address heuristic (H2) can lead to false positives and false negatives. Ex: two new outputs or a transaction with two beneficiaries of which none is a change address.
In the thesis some exceptions are proposed for H2.
It is not clear whether the first two following observations are still valid.
In a transaction t there is no exchange address if there is an output address that:

- It has already received exactly one input (H2a);

- It has already been used in a self change transaction (H2b);
- The address does not appear in transactions subsequent to t, except once as an input (H2c);

Other variants of heuristics are as follows:

- HV: if a transaction has only one output with a value less than all of its inputs, then that output is most likely an exchange address;
- HG: Clusters grow slowly, but steadily. Merging two clusters that are already large following a transaction:

  > HG_k: If updating Π (partition of all clusters) with Ct would cause the largest affected partition in Π to grow by more than a constant number of k addresses, then set Ct = $\emptyset$".

The results are shown below:

Heuristics	# Cluster	∅Size	Max Size	#Cluster w/ Size 1
H1	88 m	2.24	12 m	65 m
H1+H2	46 m	4.25	92 m	29 m
H1+H2a	51 m	3.89	87 m	32 m
H1+H2b	63 m	3.10	66 m	40 m
H1+H2c	48 m	4.13	85 m	30 m
H1+HG$_{10}$	146 m	1.34	0.1 m	123 m
H1+HG$_{100}$	121 m	1.62	0.25 m	97 m
H1+HG$_{1000}$	108 m	1.83	1 m	84 m
H1+HG$_{10000}$	104 m	1.88	8 m	81 m
H1+HV	72 m	2.71	76 m	62 m

Figure 2.7: Results of different heuristic

In all hypotheses, there is a cluster with a size between 100k and 92kk of addresses since it contains the addresses of the Mt.Gox service, a super known cluster [15].

2.3.1 DiLeNa Contribute

DiLeNa is a tool for the social network analysis that allows to explore different blockchains: Bitcoin, Ethereum, LiteCoin, DodgeCoin ande Ripple.
The tool is splitted in two part:

- *Download*: it's used the service SoChain. It, through its API, allows to download all data relating to the blockchain such as: information on blocks, information on individual transactions or details on a specific address. In the listing belowe, for example, it's possible to see a part of all details of a single address.


```
{
    "status": "success",
    "data": {
        "network": "BTC",
        "address": "36FaKKQiU8MaLcz3eDBbJsQSx5ify4hqDY",
        "balance": "1.82705298",
        "received_value": "1.82705298",
        "pending_value": "-1.82705298",
        "total_txs": 2,
        "txs": [
            {
                "txid": "177768fbc0dad71c1a29f8eec2d52d05e119a36
   fee427926ea9487fdd1c7a8e8",
                "block_no": null,
                "confirmations": 0,
                "time": 1614869194,
                "outgoing": {
                    "value": "1.82705298",
                    "outputs": [
                        {
                            "output_no": 0,
                            "address": "3GfvEicou8mT4awzq3
   xJKdFQgU2FJNH25w",
                            "value": "0.83965530",
                            "spent": null
                        },
                        {
                            "output_no": 1,
```

```
27                              "address": "1GZaU7gKdygnXHWE7AdX4
   JxBBWtNtG3uSe",
28                              "value": "0.98722530",
29                              "spent": null
30                          }
31                      ]
32                  }
33              },
```

Listing 2.1: Address information downloaded with DiLeNa

- *Analysis*: the tool uses the downloaded data to calculate: number of nodes, number of edges, clustering coefficient, degree distribution (global, in and out), same metrics for the main components etc.

```
{
   {
   "loaded_graph":{
      "global":{
         "nodes_number":57,
         "edges_number":53,
         "clustering_coefficient":0.02631578947368421,
         "degree_distribution_tot":{
            "1":27,
            "4":3,
            "3":6,
            "2":20,
            "9":1
         },
         "degree_distribution_in":{
            "1":22,
            "2":11,
            "3":3,
            "0":21
         },
         "degree_distribution_out":{
            "0":30,
            "2":18,
            "1":8,
            "9":1
         }
      },
      "main_component":{
         "nodes_number":12
```


Listing 2.2: DiLeNa's analysis

About the possibility to identify cluster and implement the heuristics discussed below, we're only interesting at the download part.

Change Address Heuristic

The change address heuristic tries to associate addresses that are part of the same transaction in a single cluster. As in the listing 2.3, the method takes in input a list

of nodes, i.e. addresses, generated in the download phase. The variable node is an array composed from a number that identifies the id of the transaction and in index higher than 0 contains all the addresses of the transaction.
For every address, all the transaction are downloaded using a GET request.

```
def changeAddressHeuristic(node):
  changeTransaction = False
  address = []
  cluster = []
  for i in range(len(node)):
    date = node[i][0]
    address.append(node[i][1])
    for j in range(1, len(node[i])):
      outputNode = node[i][j]
      tx = requests.get('https://sochain.com/api/v2/address/'+blockchain+'/' +
    outputNode).json()
```

Listing 2.3: First part of change address heuristic method

As seen before, we have several variants of heuristics that improve the standard one.
First of all, there is a check to verify if the first transaction ever of the node is equals to the date of the transaction we are considering. If the result is false, obviously the node can't be a change address.

- *H2 variant*: the first thing to check is that the date of the last transaction made by the address does not coincide with the date of the transaction we are examining. Once this check has been made, it is also necessary to verify that the number of transactions is equal to 1. If both conditions are true, then the transaction is considered to be made up of addresses belonging to the same cluster and these addresses are added to the *cluster* variable;
- *H2C variant*: the check on the date remains unchanged with respect to the variable H2. The only difference is that the second check must verify that the total number of transactions carried out by the address inn examined is equal to two. in fact, as described in the previous section, the address no longer appears in further transactions, except one, of which it is an input. In the code, referring also at the listing 2.1, an address is in input if has the value "outgoing" in the details of its data.

```
#H2
""" if tx['data']['txs'][-1]['time'] == date and tx['data']['total_txs'] == 1 and
    changeTransaction == False:
          address.append(outputNode)
```

```
          changeTransaction = True """
      #H2c
      if tx['data']['txs'][-1]['time'] == date and tx['data']['total_txs'] == 2 and
    changeTransaction == False and 'outgoing' in tx['data']['txs'][0]:
        address.append(outputNode)
        changeTransaction = True
      if tx['data']['txs'][-1]['time'] == date and tx['data']['total_txs'] == 2 and
    changeTransaction == True:
        address.clear()
        break
      if changeTransaction == True and j+1 == len(node[i]):
        cluster.append(address)
  return cluster
```

Listing 2.4: Split into H2 and H2c change address heuristic

It's possible to simulate the heuristic as in the images below.The result consists in a cluster of size 32, exactly the same number of addresses involved in all the transaction belonging to the downloaded time frame. Of course, it means that in all the transaction, no address respected the conditions imposed by the heuristics, so each address is cataloged as a cluster.

Result of Change Address Heuristic

Cluster: 32
3Kb2LZoPQZdzq2QNzi92xtFyFeJdQrtLdV
3BneVB41iC5fUq4WToqJkZPYEhdJQ4ommB
LUBLjrkTF6DymFrHN8Sjc7pz2uUq4bjhAa
La8KnhT49RUVVMReFXapyhSV3QfK1BWzHn
LZmmfe9NgHLHrWJy23PFACrWSvmN1Gcd7U
LM7Gqdm2UK3JBGreRUV6g7DEhQp4WobTwG
LdN9beP9S8PA4zKgjso8wJtfu2u2SmW5T3
LZQiCHqJFdFY2aN9uoFfdQCChqYNAFKSML
LLDUKtgXrtqWg9eqhtuFGKyKeJpc7Y4HGh
LS8UAtwFANc75zTAUp2L39vDcRsW17BH8W
LMEwhFpb7rjM67cxWqy3uFk8NsqNCERj9V
LNgtmaAttjbrkSWFzxK1nVh5xDzRxB63up
LeMGku1mwEg3hXE4C4YQ9rgb8EjEw2CFUM
LRoUpFXT1B5ypLCdT4BLeTVPNBYs6cVD6b
LZQ85C25FrVDHJPyAqunWCXLmhKsc6AaYt
LXD36VW9FkMnCg38wJfePpUPXHazbPXEan
LPp6ZLz533T1ozHaKY2ZEvMgYSuZsH8Zpd
LZE1WNDwLLSht14XGjnKVnphxVd1ghz3FL

La4Hw7NLPBr5TJ28MeYWmTje6jrKZStgXe
LSDdQAz7UP7wASApTKCNpM5CqAeLZRytoL
LPiED5c4kGecpt5iF8ETiidweXEPHsn6SN
LYjqUBLf5M7oaDMbyGhRcWirDpPqSGYLBp
LRWn4Jf7cN2b4EQTGU2eV59ivS1DvQ2rXU
LVLXg5hX2rKaeLstY61oEr6AscSfyzUnwe
LKJSMDiZYgoJLSXvfSQKuMwzqgB2kbdjxT
LSyNHeqEvBggEYr36DMMyzZ6MgSVPa595C
LL5sheYfabGDfEFp7i2iMwCYygJMSoSYMt
3BJ8Xhy58Cy1DeepSBCkfX23KsXhuPDNXx
37HkXCvAsRDqay6kBphrU7CUXMSZF47Esk
LXxbby2QHSuLZs15YPAqLowW5akatgAZdL
3NZTFp9yp8Um3vbvoFJSJ8Ah8zSaWGnAqq
3HJTMREsDJoC4gYHrkrB181gyhGn5Zcr5n

```
./main.sh -file sochain -blockchain ltc -start "2020-04-01-00:00:00" -end "2020-04-01-00:01:00" -res 'res/ltc.net'
```

Figure 2.8: Simulation of change address heuristic

Multi-input heuristic

The multi-input heuristic, probably, due to mixer and CoinJoin, it is the heuristic that has suffered the most from its effectiveness. Its implementation is trivial and it was agreed to implement it anyway so as to have a starting point for the study of a new heuristic again efficient against mixer. The implementation doesn't require a multitude of code, but just two built-ins of NetworkX.

```
def to_edges(node):
    it = iter(node)
    last = next(it)
    for current in it:
        yield last, current
        last = current

def multiInputHeuristic(node):
    G = networkx.Graph()
    for part in node:
        # each sublist is a bunch of nodes
        G.add_nodes_from(part)
        # it also implies a number of edges:
        G.add_edges_from(to_edges(part))
    return G
```

Listing 2.5: Implementation of multi-input heuristic

Noting the difficulty of identifying a time interval that would allow to find transactions that respected the conditions of the heuristic, it was decided to make an ad hoc simulation to demonstrate the correct functioning of the implemented method. The result is exactly as expected, with two clusters that include all the addresses belonging to the same transaction. The limits, again, are evident, especially after discussing mixer and coinjoin.

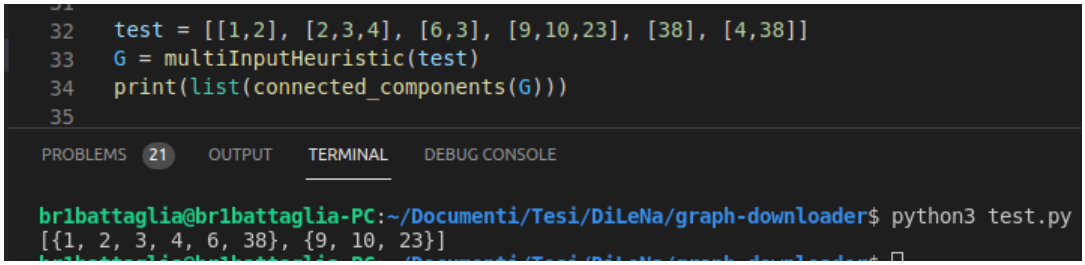


Figure 2.9: Simulation of the multi-input heuristic

Chapter 3

Bitcoin Simulator

As seen in the previous chapters, the problems to be faced are many. Coinjoint, mixer and address clusters are some of the conditions to contend with.
A problem that has not been mentioned, but not of minor importance for this, is without a doubt the cost of the operations to be carried out within the blockchain. It takes well over ten minutes to download thirty seconds of Bitcoin transaction data. Furthermore, much of that data would never be used.
It is for this reason that the simplest option appears to develop a blockchain transaction simulator. This simulator, in fact, does not need to represent the blockchain in the basic structure, but it can be limited to the highest layer, at the level of transactions.
So, the goal is not to have a network with its ledger and its blocks, but a simulator that simply stores information about the transactions that bind the nodes.

3.1 Metrics

In order to have the most accurate and realistic simulation possible, it was necessary to collect the basic Bitcoin's metrics so that we could, subsequently, design and write the simulator code.

In the presentation article of DiLeNa tool [11] there are data relating to in-degree and out-degree per node:

- *In-degree*:
  - 26% of the nodes never received any amount;
  - 63% of the nodes received cryptocurrencies only once;
  - 5% of the nodes received cryptocurrencies twice;

  - 4% of the nodes or less received cryptocurrencies more than five times.
- *Out-degree*:
  - 22% has zero outgoing transactions;
  - 25% has one outgoing transactions;
  - 50% has two outgoing transactions;
  - 3% has more than two outgoing transactions.

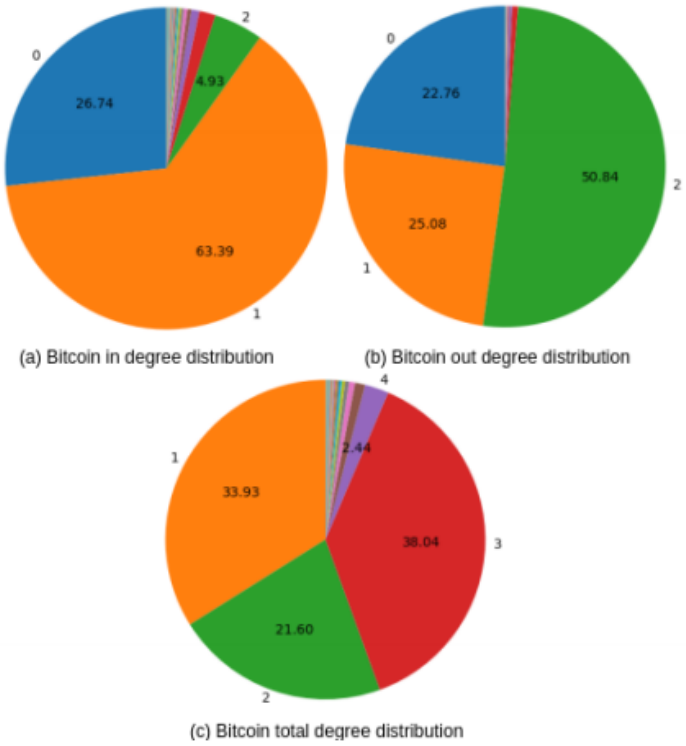


Figure 3.1: In, out and total degree of Bitcoin

In figure 3.1, it is possible to see graphically the summary of the numbers just reported.
Since these data refer to 2010, it was considered appropriate to carry out further research to verify if and how Bitcoin transactions have evolved over the years. In this regard, it is interesting a study [26] that analyzes the in-degree and out-degree values for each single year, starting from 2009 to end in mid-2021. For lack of completeness, 2021 will be discarded from the data in question.
By checking the values of the two metrics, it turns out that the in-degree fluctuates between 1.54 and 2.4, while the out-degree between 1.47 and 2.7, depending on

the year in question. Therefore, the values in DiLeNa's article, although dating back to 2010, still fall within the parameters of the most recent studies, even if with values tending more towards the low end than the high one. This last statement most likely finds its explanation in the increase in the size of the blockchain over the years.
Regarding the *multi input transactions*, the data is not available. However, it is possible to hypothesize the percentage starting from that of the coinjoin transactions. Longhash - a platform that collects data from multiple blockchains in order to visualize their evolution - claims that coinjoint transactions amount to about 4.5%. It is therefore presumable that the total of multi-input transactions is around 6-8% [8].
Of great importance, in order to verify the platform, are the information contained within "Dissecting bitcoin blockchain: Empirical analysis of bitcoin network (2009–2020)" [26] and "The Anti-Social System Properties: Bitcoin Network Data Analysis" [3]. Both studies come to the conclusion that Bitcoin follows a *power law* liquidity distribution and that it has the characteristic of the shrinking diameter [3], which seems to be a consequence of the increase in the arcs that make the components increasingly connected to each other.
The alpha value of the power law distribution is fundamental which, as reported in "Scaling properties of extreme price fluctuations in Bitcoin markets" [28], is between 2 and 2.5.
A question that needs to be answered in order to understand the size of Bitcoin is, of course, that relating to the amount of daily transactions. As reported by `https://www.blockchain.com/explorer`, it is around two hundred thousand transactions of which twenty thousand with values over one hundred thousand dollars.

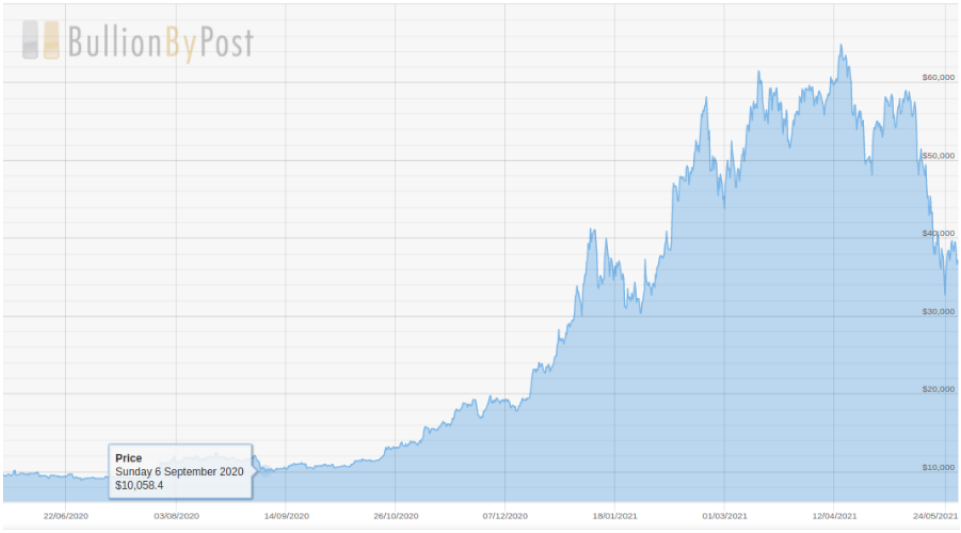


Figure 3.2: Growth in the economic value of bitcoin over the years

As for the average value, it is appropriate to refer to the last six months, given that it is the period in which Bitcoin has grown the most on the stock exchange. As of 5/26/2021, Bitcoin had an average value of $40.5k. This is equivalent to saying that a transaction of over $100k equals approximately 2.5 bitcoins. So, about 10% of transactions have values equal to or greater than 2.5 bitcoins. The *average transaction value* ranges from 400k to 500k dollars.
For the remaining transactions, however, it is appropriate to use the value given by the median for each block, which is the value of the most common transaction within each block. The value in the last six months is 0.02-0.03 BTC [20].

3.1.1 Metrics implementation

The whole simulator is written in Python and several built-ins of the NetworkX package have been employed.
In listing 3.1, it is possible to see how the first check takes place on the power law parameter. When the simulator is started, for each arc connecting two nodes, the value exchanged in bitcoin is calculated. This value is added to the sequence parameter that is fed to the built-in powerlaw.Fit(). If the value is less than 1.9 or greater than 2.5, the simulator recursively proceeds to process a new graph and create new transactions.

```
1 def calculateMetrix(DG,sequence):
```

```
  sumOut = sumIn = singleIn = singleOut = doubleIn = doubleOut = zeroIn = zeroOut =
     totOut = totIn = 0
  fit = powerlaw.Fit(sequence)
  print(fit.alpha)
  print("--------------")
  if(1.9<fit.alpha and fit.alpha > 2.6):
    return False
```

Listing 3.1: Power law fitting function to check the coherence with Bitcoin

If the result is included in the parameters previously illustrated, as shown in listing 3.2, the number of incoming and outgoing transactions is checked for each node. This value is added to the designated variable to account for nodes with zero, one, two, or more transactions so that, at the end of the process, it is possible to match the in-degree and out-degree numbers listed in section 3.1.

```
else:
    for i in range(1,len(DG)):
      #out degree
      totOut = totOut + DG.out_degree(i)
      if DG.out_degree(i) ==1 :
        singleOut = singleOut+1
      elif DG.out_degree(i) == 2:
        doubleOut = doubleOut+1
      elif DG.out_degree(i) == 0:
        zeroOut = zeroOut+1
      elif DG.out_degree(i) >2:
        sumOut = sumOut+1
      #in degree
      totIn = totIn + DG.in_degree(i)
      if DG.in_degree(i) == 1 :
        singleIn = singleIn+1
      elif DG.in_degree(i) == 2:
        doubleIn = doubleIn+1
      elif DG.in_degree(i) == 0:
        zeroIn = zeroIn+1
      elif DG.in_degree(i) >2:
        sumIn = sumIn+1
```

Listing 3.2: Check of the number of inbound and outbound transactions for each node

Once the calculation of the transactions has been completed, we proceed with the calculation of the average degree both of the entire graph and of that which in listing 3.3 is defined as mainComponents. These are the connected components of the network. To calculate the connected components it is necessary to transform the digraph into a undirect graph and create a subgraph, called MC, with the mainComponents.

```
#degree medio main component
    G = nx.to_undirected(DG)
    mainComponents = sorted(nx.connected_components(G), key=len, reverse=True)
    MC = G.subgraph(mainComponents[0])
```

Listing 3.3: Average degree of main components

A further calculation can be performed on some metrics such as the *average shortest path length*, both for the main components and for the entire network. Obviously, to have realistic data, it would be necessary to simulate a network with a number of nodes close to that of Bitcoin. For this reason, this section is limited to reporting the code used for the calculation, but it is recommended not to take these values as certain.
In listing 3.4, it is possible to see the implementation. For what concerns the components already connected, it is quite enough to use the built-in of NetworkX. For the entire graph, on the other hand, it is necessary to calculate the average shortest path length of each node, add it within a variable and, finally, divide the total by the number of nodes in the network.

```
print("ASPL MC: "+str(nx.average_shortest_path_length(MC)))
    S = [DG.subgraph(c).copy() for c in nx.weakly_connected_components(DG)]
    for index,value in enumerate(S):
      try: aspl
      except NameError: aspl = 0
      aspl = aspl + nx.average_shortest_path_length(value)
      if index == (len(S)-1):
        aspl = aspl/len(S)
    print("ASPL: "+str(aspl))
    return True
```

Listing 3.4: Average shortest path length of main components and the entire network

3.2 Simulator's code

The code starts by creating a digraph through NetworkX and, subsequently, the fillGraph () method is invoked - fulcrum of the simulator initialization process - so that all the connections between the nodes can be added following the previously reported percentages.
As seen in the previous section, if the calculateMetrix method returns a true value, the program generates the entire network and saves it in gexf format. If the return value is false, through a recursive call, a new network is generated until all the previously seen metrics are respected.

```
44 def createGraph(n):
45   DG = nx.MultiDiGraph()
46   DG.add_nodes_from(range(1,n))
47   fillGraph(DG)
48   result = mt.calculateMetrix(DG,sequence)
49   if(result):
50     nx.write_gexf(DG, "bitcoin.gexf")
51   else:
52     sequence.clear()
53     createGraph(n)
54   return DG
55
56
57   def fillGraph(DG):
58   x = init(DG)
59   #ghostInputNode = x[0]
60   singleInputNode = x[1]
61   doubleInputNode = x[2]
62   multiInputNode = x[3]
63   #ghostOutputNode = x[4]
64   singleOutputNode = x[5]
65   doubleOutputNode = x[6]
66   multiOutputNode = x[7]
67   #multiInpuntTransaction = x[8]
68   maxNumTransaction = x[9]
```

Listing 3.5: Initialization of the creation process for the simulated network

As anticipated, the heart of the generation process is the fillGraph () method. In fact, it initially has the task of extracting the data from the init method. As reported in listing 3.6, init has the task of randomly selecting nodes in the network according to the percentages of in-degree and out-degree present in DiLeNa's presentation article and adding them to lists from which, subsequently, will be taken

to generate transactions between nodes.
The process is basic: the set of all nodes is taken, nodes are chosen according to the percentages seen previously and, finally, the list just generated is eliminated from the total list of nodes. Everything is done for nodes with zero, one and two transactions, both inbound and outbound.
This management of the sets allows us to easily modify the tool if in-degree and out-degree analyzes are to be carried out on more recent Bitcoin data. In the absence of them, and verified that they still fall within the indicative values of Bitcoin, the simulator has been developed using the percentages seen above.
The nodes were chosen randomly in order to make the simulation as realistic as possible.

```
69  def init(DG):
70    node = []
71    global toList
72    toList = list(DG)
73    multiInpuntTransaction = random.sample(toList, int(0.07*len(toList)))
74    #print(multiInpuntTransaction)
75    #sys.exit()
76    for i in range(len(toList)):
77      node.append([])
78      node[i].append(toList[i])
79      node[i].append(0)
80
81    outDegreeNode = node[:]
82    inDegreeNode = node[:]
83    singleOutputNode = []
84    doubleOutputNode = []
85    singleInputNode = []
86    doubleInputNode = []
87
88    ghostInputNode = random.sample(inDegreeNode, (int(0.26*len(node))))
89    inDegreeNode = diff(inDegreeNode, ghostInputNode)
90    singleInputNode = random.sample(inDegreeNode, (int(0.63*len(node))))
91    inDegreeNode = diff(inDegreeNode, singleInputNode)
92    doubleInputNode = random.sample(inDegreeNode, (int(0.05*len(node))))
93    inDegreeNode = diff(inDegreeNode, doubleInputNode)
94    for i in range(len(inDegreeNode)):
95      inDegreeNode[i][1] = 3
96
97    ghostOutputNode = random.sample(outDegreeNode, (int(0.22*len(node))))
98    outDegreeNode = diff(outDegreeNode, ghostOutputNode)
99    singleOutputNode = random.sample(outDegreeNode, (int(0.25*len(node))))
100   outDegreeNode = diff(outDegreeNode, singleOutputNode)
```

```
101   doubleOutputNode = random.sample(outDegreeNode, (int(0.5*len(node))))
102   outDegreeNode = diff(outDegreeNode, doubleOutputNode)
103   maxnumTransaction = sum((len(singleOutputNode),(2*len(doubleOutputNode)),(8*len(
        outDegreeNode))))
104   return ghostInputNode,singleInputNode,doubleInputNode,inDegreeNode,
        ghostOutputNode,singleOutputNode,doubleOutputNode,outDegreeNode,
        multiInpuntTransaction, maxnumTransaction
```

Listing 3.6: Lists of nodes creation

The process continues by choosing both a node that sends the bitcoins and one (or more in the case of multi-input transactions) that receives them. In listing 3.7, how the sender is chosen can be seen.

```
105 while(True):
106     while(True):
107       data = chooseSender(singleOutputNode, doubleOutputNode, multiOutputNode)
108       if(data[0] != None and data[1] != None and data[2] != None):
109         break
110     sender = data[0]
```

Listing 3.7: End of the process

The chooseSender method receives as a parameter the three lists of nodes set up for sending bitcoins.
Variables *sender, index* and *currentList* are initialized as *None*.

- Sender: the node from which the transaction starts;
- Index: how many arcs have to start from sender;
- currentList: the list from which the sender node has to be selected.

Subsequently, since the total list of nodes has been divided into three different lists of nodes that perform transactions, a random number between 1 and 3 is chosen. If the number is 1 or 2, it's checked that the singleOutputNode or doubleOutputNode lists are not empty and, if not, the list from which to take the sender node is copied into currentList. The index is set as the newly generated random number and the sender is randomly chosen from the currentList.
The only big difference is if randomOutput equals three. In this case, the value of index is randomly chosen between a number between 3 and 8.
If the selected list is empty, a recursive call is proceed until a list is chosen.
The return values are the same as initialized at the beginning of the function.

```
111
112 def chooseSender(singleOutputNode, doubleOutputNode, multiOutputNode):
```

```
  sender = None
  index = None
  currentList = None
  randomOutput = random.randrange(1,4)
  if(randomOutput == 1 and len(singleOutputNode) !=0):
    currentList = singleOutputNode
    sender = random.choice(currentList)
    index = 1
  elif(randomOutput == 2 and len(doubleOutputNode) !=0):
    currentList = doubleOutputNode
    sender = random.choice(currentList)
    index = 2
  elif(randomOutput == 3 and len(multiOutputNode) !=0):
    currentList = multiOutputNode
    sender = random.choice(currentList)
    index = random.randrange(3,8)
  else:
    chooseSender(singleOutputNode, doubleOutputNode, multiOutputNode )
  return sender, index, currentList
```

Listing 3.8: Sender choice

As seen in the previous section, to make the simulator as realistic as possible, it is necessary to generate imports with those of the transactions present on Bitcoin. Relying on the data illustrated after the figure 3.1, the amount, identified by the *amount* variable, will be between 0.00001 and 2.5 bitcoin (with 8-digit precision) if the number of transactions carried out is less than or equal to 90% of the maximum number of possible transactions linked to the number of nodes and percentages seen in section 3.1; otherwise it will be between 2.6 and 14, always with 8-digit precision.

These numbers are not random.
The minimum amount chosen is given by the fact that it coincides with the most famous services for buying or selling bitcoins, the limit is precisely given by the value of 1000 Satoshi or so [34] [33] [21] [29]. Remember that 1000 Satoshi equals to 0.00001000 BTC.
The value 2.5, as explained above, is the equivalent of transactions with value equals to $100k, so higher than the 10% of all transactions per day.
The number 14, on the other hand, is an arbitrarily large number whose sole objective is to place an upper limit on the import generator. In fact, since these are already very high figures and already being part of the smallest percentage of Bitcoin transactions, this upper limit is sufficient to trigger an alarm bell and carry out checks.

Subsequently, a timestamp is randomly chosen that can be from the date of creation of Bitcoin up to the current date always expressed in seconds starting from the so-called epoch.
Any association between nodes is random. After selecting the sender, it is necessary to choose a receiver. The connection is created using the addTransaction() method and the auxiliary value of receiver is increased thanks to which the process can keep track of the incoming transactions of each node.
As with the receiver, the sender is also removed from the its list.
It should be noted that self change transactions have been excluded by checking that the sender is different from the receiver, as they would make the graph more complicated without providing useful information for the purpose of the dissertation.
When the three lists of nodes predisposed to send bitcoins are empty, the process ends.
A key role is certainly that of the addTransaction method, which takes care of adding transactions after the sender and receiver have been selected. Before seeing its implementation in detail in listing 3.11 and 3.12, the method of selecting the receiver node is illustrated.

```
132
133 if(numTransaction <= int(maxNumTransaction*0.9)):
134       amount = round(random.uniform(0.00001,2.5),8)
135     else:
136       amount = round(random.uniform(2.6,14),8)
137     index = data[1]
138     currentList = data[2]
139     for i in range(index):
140       timestamp = random.randint(1620134991,calendar.timegm(time.gmtime()))
141       while(True):
142         receiver = chooseReceiver(singleInputNode, doubleInputNode, multiInputNode)
143         if(receiver != 0 and receiver != sender):
144           break
145       addTransaction(DG, sender[0],receiver[0], amount, timestamp)
146       receiver[1] = receiver[1]+1
147       if(receiver in singleInputNode and receiver[1] == 1):
148         singleInputNode.remove(receiver)
149       elif(receiver in doubleInputNode and receiver[1] == 2):
150         doubleInputNode.remove(receiver)
151     currentList.remove(sender)
152     if(len(singleOutputNode) == 0 and len(doubleOutputNode) == 0 and len(
    multiOutputNode) == 0) :
```

```
153       break
```

Listing 3.9: Core of fillGraph() method

As for the sender, the parameters passed to the method are the three lists of nodes prepared for sending the cryptocurrency. It is randomly generated a number between 1 and 3 because the list of total node has been divideed in three different list that can recevive criptocurrency following the percentage reported above. The node is initialized to the value 0. Depending on the value of the *rnd* variable, a list of nodes predisposed to receive the coin is chosen and, after checking that that list is not empty, a node is randomly selected from the designated list.
Finally, the chosen node is returned.
The implementation of addTransaction shown in figures 3.13 and 3.14 resumes below.

```
154 def chooseReceiver(singleInputNode, doubleInputNode, multiInputNode):
155   rnd = random.randrange(1,4)
156   node = 0
157   if rnd == 1 and len(singleInputNode) != 0:
158     node = random.choice(singleInputNode)
159   elif rnd == 2 and len(doubleInputNode) != 0:
160     node = random.choice(doubleInputNode)
161   elif rnd == 3 and len(multiInputNode) != 0:
162     node = random.choice(multiInputNode)
163   return node
```

Listing 3.10: Definition and implementation of chooseReceiver() method

Since the graph will be drawn using NetworkX functions, it was believed that having the color distinction between transactions with a large number of Bitcoins and standard ones can serve as a graphical and visual aid. The first part of addTransaction, in fact, deals with verifying the amount and, based on it, choosing the color to associate with the transaction. For transactions with a large amount, the color red was chosen and gray for all the others.

```
164 def addTransaction(DG, sender, receiver, amount, timestamp, color = False):
165   global numTransaction
166   if(color == False):
167     if(amount>0.1):
168       color = '#FF0000'
169     else:
170       color ='#74889a'
171   else:
172     color ='#1d812c'
```

Listing 3.11: Color selection of the arch in the graph

The remaining part of the method must manage different cases:

- *One to one transactions*: transactions in which there is only one sender node and one receiver node;
- *Batched transactions*: transactions in which the sender is only one, but there are multiple receivers;
- *Multi input transactions*: transactions in which there are multiple sender nodes and only one receiver node;
- *Multi input and multi output transactions*: transactions in which there are both sender and receiver nodes.

The management of transactions with multiple nodes, regardless of whether they are incoming or outgoing, is entrusted to for loops that scroll through the list of received nodes.
When an arc is added, some properties previously expressed are associated with it: *color of the arc, amount expressed in bitcoin and timestamp*.

```
#one to one transaction
  if isinstance(sender, int):
    if isinstance(receiver, int):
      DG.add_edge(sender,receiver, value = amount, date = timestamp, edge_color =
    color)
      sequence.append(amount)
      numTransaction += 1
    else:
      #batched transaction
      for j in range(len(receiver)):
        DG.add_edge(sender,receiver[j], value = amount, date = timestamp,
    edge_color = color)
        sequence.append(amount)
        numTransaction += 1
  elif isinstance(sender, str):
    if isinstance(receiver, str):
      DG.add_edge(sender,receiver, value = amount, date = timestamp, edge_color =
    color)
    else:
      #batched transaction
      for j in range(len(receiver)):
        DG.add_edge(sender,receiver[j], value = amount, date = timestamp,
    edge_color = color)
  else:
    #multi input transaction
    for i in range(len(sender)):
      if isinstance(receiver, int):
```

```
196         DG.add_edge(sender[i],receiver, value = amount, date = timestamp,
    edge_color = color)
197         numTransaction += 1
198         sequence.append(amount)
199       else:
200         #multi input and multi output transaction
201         for j in range(len(receiver)):
202           DG.add_edge(sender[i],receiver[j], value = amount, date = timestamp,
    edge_color = color)
203           numTransaction += 1
204           sequence.append(amount)
```

Listing 3.12: All the cases for adding a transaction between two or more nodes

An important question at this point of the dissertation is: how the algorithm knows which kind of transaction has to add in the graph? As repeated multiple times, all the code works trying to generate transaction in the more random way. The base is that there are the percentages from DiLeNa's presentation article and six lists of addresses (three for outgoing transactions and three for incoming transactions), extracted using these percentages from the list that contains all nodes.
At this point, the algorithm has to choose the sender, verify that it is within the list with nodes designated to send one, two or more transactions and choose the receiver. The latter is also chosen from three lists, each designated to contain nodes that can receive one, two or more transactions. Since many mixers allow a limited number of addresses, the decision has been to choose a value very close to that expected by these services, that is 8. This completely random choice operation continues until all the lists are empty. In fact, when a node reaches its rank, it is removed from the list to which it belongs.

Chapter 4

Results

In this section we will examine in depth the algorithm for the recognition of malicious transactions, we'll start and evaluate a simulation of harmful transactions within and to study the behavior that the algorithm takes on a second of the value assigned to a parameter inside it.

4.1 Recognition algorithm

Once the simulator is ready to generate the network, we obtain a graph with the previously expressed characteristics which appears as in figure 4.1. In this case, the number of nodes is 100.

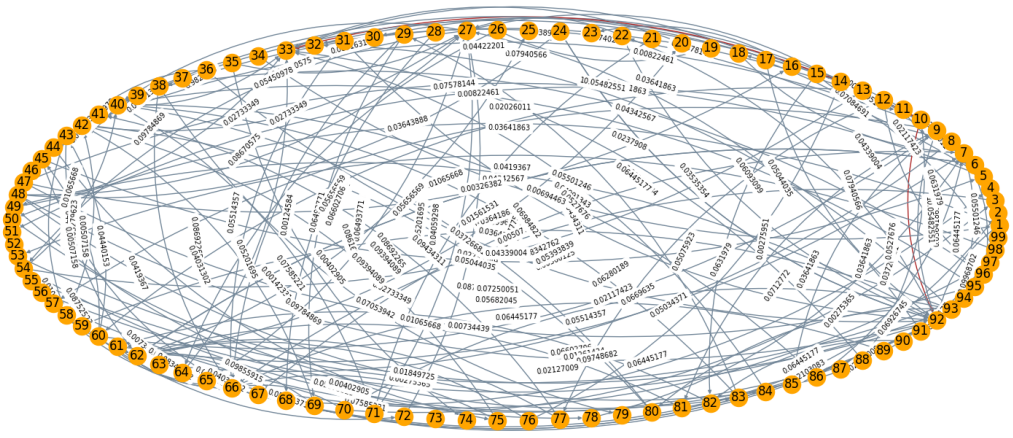


Figure 4.1: Graph of the simulated network

Having kept the graph and verified that all the metrics are respected, it is possible to move on to the actual analysis in order to identify potential dangerous transactions.
To do this, an algorithm has been conceived and designed that takes into account various elements: the *timestamp* of the transactions and the information of a node whether or not it belongs to the *cluster of addresses of some mixing service* .
The algorithm has an investigative approach, i.e. it is necessary to assume that an address is potentially dangerous in order to control its transactions. As parameters, in fact, the identifyML method takes in input the whole network and the specific node.
Of vital importance in this phase, it is to define in the ego network of the node, or that sub-graph, characterized by the border of the node. The parameter on which to work and carry out multiple tests is the one concerning the radius, that is the maximum distance, from the starting node. In a network of 100 nodes, for example, this parameter was set to 5.
Subsequently, it's important to define the frontier of the node and, for each node in the frontier, all the paths where bitcoins have traveled are calculated. Since it is expected that a node, after countless turns, wants to reacquire its bitcoins, the *all_simple_path* [1] are calculated, i.e. all the paths of the graph that have both as origin and target the node under consideration.
In other words, the algorithm calculates the network of a user-defined size and calculates all *closed walks* for the suspect nod taking into account the variables of the timestamp, the amount and the "mixer" property of the node.

```
205 def identifyML(DG, node):
206
207     path = []
208     toReturn = []
209     ego_network = nx.ego_graph(DG,node,radius = 5)
210     adjacent_node = ego_network.edges(node)
211
212     #calculate all the paths from the node to the node
213     for u, v in adjacent_node:
214         try:
215             path.extend(nx.all_simple_paths(ego_network, str(v), node))
216         except:
217             pass
218     currentTimestamp = 0
```

Listing 4.1: First part of the method to identify money laundry activity

Starting from each node within the frontier, there is the need to check that none of them are part of any mixing system. Since the simulator does not yet provide a

service that emulates the behavior of a mixer, in the simulator, the feature of being part of a mixer is a property that any node can have. In the example below, the candidate to be part of a mixer is the node 91.

```
DG.nodes['91']['mixer'] = True
```

Listing 4.2: How to set the mixer property using Network

About the mixer feature, it is done by going through each path, starting from the potentially dangerous address and finishing to it, and verifying that the "mixer" feature of every node in the path is false.
Since some nodes of a path could be part of mixing systems, regardless of the timestamp values, a path that has an address inside it belonging to a mixer is automatically considered dangerous and is reported to the user.
It is believed that, even if the timestamps do not respect a chronological order, a transaction involved in a mixing process must be subject to further verification and that the entire path can lead back to typical patterns of the mixer used by the user in his attempt to remain anonymous.

```
#if a node of the path is in a mixer cluster, then the path could be dangerous
   for i in range(len(path)):
       previousTimestamp = 0
       validPath = True
       for j in range(len(path[i])):
           try:
               mixer = ego_network.nodes[path[i][j]]['mixer']
               if mixer:
                   toReturn.append(path[i])
                   break
           except:
               pass
```

Listing 4.3: Known mixer's address

If no node belongs to a cluster of some mixer, then the algorithm proceeds to check the timestamps.
Once in the frontier, two timestamp values need be stored: the one of the current transaction and that of the immediately following transaction. All this procedure is necessary to follow the correct chronological order. This verification is entrusted to if blocks which, extracting the transaction information, verify that the second timestamp is greater than the first.
If this occurs throughout the path, the path is considered valid and is added to the list of paths to be returned to the user. Otherwise, the *validPath* variable, initially set to True, changes its state to False and the path is discarded.

If this occurs throughout the path, the path is considered valid and is added to the list of paths to be returned to the user. Otherwise, the *validPath* variable, initially set to True, changes its state to False and the path is discarded.

```
#if not, it is verified that the node have a coherent timestamp
            if(j+1) < len(path[i]) and validPath == True:
                currentTimestamp = DG.get_edge_data(path[i][j],path[i][j+1])
                if(j == 0):
                    previousTimestamp = DG.get_edge_data(node,path[i][j])

                if(previousTimestamp[0]['date']< currentTimestamp[0]['date']):
                    previousTimestamp = currentTimestamp
                else:
                    validPath = False
                    break

            if (j+2) == len(path[i]) and validPath:
                toReturn.append(path[i])
    for i in toReturn:
        print(str(i))
```

Listing 4.4: Final part of the algorithm

4.2 Simulation

Since the moment the algorithm has been implemented, it remains to verify its operation by performing more simulations.
Transactions that simulate a money laundry activity are added, but, before starting the simulation, there are some important aspect to note:

- The suspicious node is the 48;
- The sender of the first transaction coincides with the receiver of the last one;
- The timestamp of a new transaction is higher than the previous one;
- Each subsequent transaction has an increasingly lower amount due to the fees to be paid by the sender if he wants to have new recycled cryptocurrencies;
- As can be seen in listing 4.5, node 91 has been assigned the characteristic of being part of a mixer. This will be of great help in understanding if the algorithm for the detection of money laundry is working correctly and if, indeed, the paths that pass through the aforementioned node will be returned as potentially dangerous.

```
248 def addMLTransaction(DG):
249     DG.nodes['91']['mixer'] = True
250     addTransaction(DG, '48', '50', 0.058764, 1620575060, color = True)
251     addTransaction(DG, '50', '55', 0.058763, 1620575061, color = True)
252     addTransaction(DG, '55', '13', 0.058762, 1620575062, color = True)
253     addTransaction(DG, '13', '91', 0.058761, 1620575063, color = True)
254     addTransaction(DG, '91', '48', 0.056, 1620575064, color = True)
```

Listing 4.5: Simulation of a ML transaction

Now, it is possible starting the simulator in its entirety.
Initially, the simulator will generate a graph that reflects all the imposed metrics or, if it exists, it will load the entire network by extrapolating the data about nodes and arches from the .gexf file and will add the links between nodes that represent the recycling transaction.

```
255 #------------------------main block----------------------------
256 try:
257     with open('bitcoin.gexf') as f:
258         if(f):
259             DG = nx.read_gexf('bitcoin.gexf')
260             #plotGraph(DG)
261 except IOError:
262     DG = createGraph(NUMBER_OF_NODE)
263     calculateMetrix(DG, findSequence(DG))
264     plotGraph(DG)
265 
266 addMLTransaction(DG)
267 identifyML(DG, '48')
```

Listing 4.6: Simulation beginning

Our expectation is to receive as output the path we have generated using the *addMLTransaction* and, if they exist, other paths that involve node 91, even if the chronological order feature is not respected. Figure 4.2 allows to see how the algorithm has reflected the expectations. The path generated by the addMLTransaction method is present, as well as two other paths, i.e those that included the address that is part of a mixer.

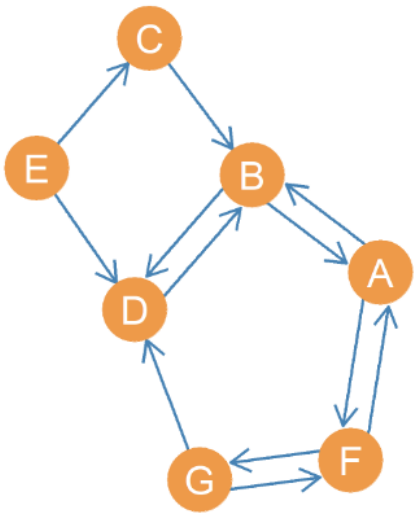
```
br1battaglia@br1battaglia-PC:~/Documenti/Tesi/BitcoinSimulator/BitcoinSimulator$ python3 main.py
['50', '55', '13', '88', '23', '91', '48']
['50', '55', '13', '91', '88', '49', '48']
['50', '55', '13', '91', '48']
```

Figure 4.2: Dangerous paths with ego network parameter equals to 5%

The goal of this simulation is to reveal how the transactions involved in money laundering, having to deliver the cryptocurrencies to the owner, are identifiable and it is possible to identify patterns based on the timestamp and the quantity of coins. Clearly the simulator is in an embryonic state and is only the first step towards what could be a much deeper analysis of the network. However, as author, I'm confident that the implemented algorithm is valid and is easily adaptable even when the feature for clustering assigned to the same physical person will be implemented. In fact, in the last scenario, we will no longer have single addresses identified by nodes, but clusters and the verification on the starting node which coincides with the arrival node will move to the starting cluster and the arrival cluster.

4.2.1 Ego network size

Having ascertained that the algorithm works and returns the expected results, it is necessary to return to a point mentioned in the previous section: the size of the ego network.
It is necessary to carry out tests and see how much this value affects the results of the algorithm and calculate the percentage of false positives that is created by not adequately balancing this parameter.
We recall, in fact, that Bitcoin and many other blockchain networks enjoy the ownership of the shrinking diameter [2]. It is none other than the characteristic of a graph of having nodes more connected as the number of arcs present in the network increases. It means that the more the arcs increase, the more the nodes are reachable in the reachability matrix.
In a graph, node B is said to be reachable by node A if there is path (of any length) that has A as the origin node and B as the destination node. In that case, it's assumed that A can reach B [30]. Therefore, this matrix will have 1s corresponding to the nodes from which it is possible to reach a node; 0 otherwise. The diagonal, of course, will be null.
In the figure 4.3, an example of a graph and its reachability matrix.



	A	B	C	D	E	F	G
A	–	1	0	1	0	1	1
B	1	–	0	1	0	1	1
C	1	1	–	1	0	1	1
D	1	1	0	–	0	1	1
E	1	1	1	1	–	1	1
F	1	1	0	1	0	–	1
G	1	1	0	1	0	1	–

Figure 4.3: Example of reachability matrix

In this section, three parameters must be considered:

- **Network size**: the number of edges or total of all edge weights [13];
- **Ego-network size radius**: subgraph of neighbors centered at a certein node within a given radius; [10]
- **Path length**: path of length n of in graph is given by a sequence of vertices v_0, v_1, ..., distinct v_n (not necessarily all) and by a sequence of edges that connect them (v_0, v_1), (v_1, v_2) , ..., (v_{n-1}), v_n). The vertices v_0 and v_n are called extremes of the path. [24].

Since the algorithm is set for scanning timestamps, regardless of the shrinking diameter phenomenon, it is evident that the probability that there are paths that randomly satisfy the time requirement is very low.
On a network of 100 nodes with a *size of 151*, by setting a *ego network parameter that corresponds to 5% of the total size of the network*, three dangerous paths were found, namely those of figure 4.2. The path that passes through multiple

nodes has a *maximum length of 7, equal to 7% of the network*.
Trying to increase the ego network value up to 10%, the results obtained are similar, but we find the presence of new paths.
The new results are available at the figure 4.4.

```
br1battaglia@br1battaglia-PC:~/Documenti/Tesi/BitcoinSimulator/BitcoinSimulator$ python3 main.py
['50', '55', '13', '88', '23', '91', '48']
['50', '55', '13', '91', '88', '49', '48']
['50', '55', '13', '91', '74', '31', '88', '49', '48']
['50', '55', '13', '91', '48']
```

Figure 4.4: Dangerous paths with ego network parameter equal to 10%

In the second case, the network has still 100 nodes and a size of 151, an ego network parameter equals to 10%. It is immediately evident that there is a new path which, unsurprisingly, is the one of greater length. In fact, the new path passes through 9 nodes, or 9% of the network. The increase of this parameter, even up to a maximum of the totality of the network, so that the ego network and the network coincide, does not provide new paths. This means that the algorithm has identified all the paths to keep an eye on.
Obviously, computational power permitting, more accurate simulations would have to be carried out with much larger network sizes. Even in those cases, however, an entire network will never meet the requirements of the algorithm, so it is believed that a parameter ranging from 2% to a maximum of 10% for the most active nodes, is ideal for avoid getting false positives.

Conclusions

The technological world moves forward and the world of crime seems to travel at even greater speeds. The ability of criminal organizations (or individuals) to adapt their malicious intent to technologies must ensure that the world of struggle and research put a stop to a race that, otherwise, would be unstoppable.
The evolution of blockchains, first simple experiments and then real economic bills, has meant that the topic is no longer within the reach of a few insiders. By now anyone has heard of cryptocurrencies, investments and, in the world of crime, the possibility of using this tool to launder money immediately became clear. If in 2009, the first bitcoin has been used for buying a pizza [4], now the intentions of cyber criminals are completely different. This dissertation does not aim to make all criminal attempts immune, but to make a long review of the technologies and practices adopted to defend against potential processes of deanonymization and aims to illustrate, in future works, those that could be of the steps to follow to fill the gap that crime has taken towards the "good guys". Since knowing the basic structure of the blockchain is crucial, in the *first chapter* has been discussed its implementation in the low level, to then move on to social network analysis, them necessary and useful for the analysis of networks in an attempt to understand what is happening inside the blockchain or how it is evolving.
Of equal importance, addressed in *Chapter two*, is the problem of mixing money. Two types of mixers were discussed, both centralized and decentralized, their weaknesses and strengths were exposed. It seems possible to be able to connect these technologies because, as they are anomalous, they leave patterns easily distinguishable from others. A further problem, treated as an extension of the DiLeNa tool, was the implementation of heuristics that would allow the clustering of blockchain addresses belonging to the same physical person.
Despite everything, the work on the real Bitcoin network has led, in *chapter three* to want and have to implement a simulator in order to be able to perform simulations based on innovative algorithms, albeit in an embryonic phase and which, therefore, do not cover all cases. The simulator is able to correctly emulate the highest layer of the network, respecting the metrics that characterize Bitcoin.

Moreover, its design foresees that a possible change of parameters is very simple and immediate.
In the *last chapter*, on the other hand, the algorithm for identifying money laundering activities was examined in depth. Since the simulator, for now, does not have methods to identify de and centralized mixers, there has been the choice to proceed in a deterministic way, exploiting some mandatory characteristics for recycling: in fact, having suspicions on a node, the closed path that starts and returns the node itself must respect a chronological sequence and an increasingly smaller amount due to the fees of the blockchain or the percentages that the mixing services withhold from the user's wallet. Furthermore, if a node of the path belongs to a cluster of addresses of a mixer, then that path is immediately considered dangerous. The simulation, giving the expected results, has shifted the question to a high focal point, namely that of how large the size of the ego network should be. The basic idea is that the larger the network becomes, the smaller the percentage of the size of the ego network must be, this is due to the shrinking diameter that leads the blockchain to be more and more connected, therefore leading to more and more reachable nodes. starting from any node. Obviously the impositions on timestamps and decreasing amounts are very stringent, so false positives would never be an excess, but it is believed that a size of 5% of the total network is enough. If it turns out to be an inadequate number, you can drag it up and always update the results.

Future works

As for future developments, it is believed that some implementations to be realized are:

- *New heuristics*: it is necessary to implement new heuristics that address the inefficiency of the current ones linked to mixing services and the problem of clusters. Reducing the size of the network is essential if the will is to significantly reduce time-related costs and if you want to have a clearer idea of the tour that the various bitcoins make, which would then move to cluster nodes and no longer to address nodes;
- *De-mixing*: mixers are the biggest obstacle to overcome, but it has seen that it is possible to study their characteristics and that, as far as decentralized mixers are concerned, the transaction pattern is easily recognizable. Implementing such systems in the simulator, in addition to clustering, would allow to walk almost hand in hand with the world of crime;

- *Supervised machine learning system*: the final step would be to train a deep neural network to analyze the blockchain network on the basis of data provided both through the simulator and through simulations actually carried out on the Bitcoin network. A system that learns automatically, no longer works in deterministic terms and constantly monitors the progress of the blocks would constitute an important brake on the world of recycling and would be an extremely useful tool for the Authorities who struggle daily against these issues;
- *Other blockchains*: if Bitcoin were no longer "secure", it would almost certainly move to lesser known blockchains. For this the whole project could and should be expanded to other blockchains. Working only on the upper layer and not on the architectural basis of the blockchain, the passage would not require all the steps suggested in the previous points and during the chapters of this dissertation.

The hope is that technological advancement will act as a means to grow, to be increasingly interconnected, to create new bonds and that, more and more, the world of crime will be eradicated and excluded from the wonders that technology offers to man.

Summary

La tesi verte sul tema dell'identificazione di attività di riciclaggio di denaro all'interno della rete Bitcoin utilizzano gli strumenti forniti dall'analisi delle reti sociali. Per quanto i termini blockchain e criptovalute siano entrati a far parte nel gergo comune, è necessario avere una conoscenza profonda di quella che è una rete blockchain e di quali indici e metriche siano utili al caso in analisi.
Il *primo capitolo*, difatti, spiega la struttura di base della blockchain, da cosa sia costituito un blocco e una transazione, di come ogni blocco sia identificato da un codice hash e di come una modifica sulla singola catena comporterebbe una violazione di integrità che verrebbe immediatamente rilevata.
Per quanto concerce la social network analysis, il capitolo tratta le differenti tipologie di grafo, il fatto che un grafo possa essere orientato o meno, connesso o non connesso e scende nel dettaglio, anche matematico, di alcuni parametri come grado, varianza ed eccentricità. Specie in-degree, ovvero numero di archi entranti in un nodo, e out-degree, ovvero numero di archi uscenti da un nodo, risulteranno di notevole importanza nelle sezioni future.
Per concludere il capitolo, viene esposto il cosìdetto Stato dell'Arte dove è illustrato come, anche se non in ambienti blockchain, l'analisi di reti sociali sia già applicata nel settore bancario per individuare operazioni di lavaggio del denaro.
La dissertazione prosegue con il *capitolo due* illustrando alcuni dei principali problemi con i quali, tale progetto, è per forza di cose costretto a scontrarsi:

- *Mixer*: sono dei servizi volti ad aumentare la pseudo-anonimicità della blockchain. Essi possono essere di due tipi:
  - Centralizzati: servizi che mischiano diverse quantità di monete ricevute da indirizzi separati e inviano una quantità casuale di bitcoin ad ogni indirizzo. Questo processo causale viene ripetuto fino a quando l'importo totale desiderato dall'attore che usufruisce del servizio non viene restituito nel portafoglio dell'utente. Altro fattore che aumenta la difficoltà, è che queste transazioni vengono inviate con un ritardo impostabile fino ad una settimana.

Poiché l'obiettivo primo è quello di aumentare il livello di privacy, ma i mixer centralizzati fanno riferimento ad un server centrale che memorizza informazioni quali IP e metadati, molti utenti preferiscono i mixer decentralizzati;

  - Decentralizzati: i mixer decentralizzati, non dipendendo da un server centrale, vanno a risolvere il problema deimixer centralizzati. Il nome più comune con cui essi vengono identificati è coinjoin.

- *Coinjoin*: L'idea è che più attori si coordinino per creare una transazione composta da più input e più output. Poiché tutti gli input sono combinati, diventa impossibile dire con certezza quale output appartiene a quale utente. Naturalmente, maggiore è il numero di utenti nel pool, maggiore è la difficoltà nel collegare l'indirizzo di input con quello di output;
  È chiaro, tuttavia, che a differenza delle operazioni di mixaggio, la pratica del coinjoin può essere implementata senza ricorrere a un servizio di terze parti, che, essendo automatizzato, può essere studiato attraverso processi di reverse engineering. La pratica del coinjoin, invece, rende tutto maggiormente filtrato.

- *Cluster*: alcuni studi dimostrano come il numero di indirizzi all'interno della blockchain, negli ultimi anni, sia cresciuto più del numero di transazioni effettuate. Questo implica che, gli utenti, per preservare la loro identità, generaro spesso nuovi indirizzi. Una delle complicazioni da risolvere è quella di capire quali indirizzi appartengano alla stessa persona fisica, al fine di raggrupparli in un unico cluster. Attualmente, nello stato dell'Arte, esistono due euristiche principali: multi-input heuristic e change address heuristic, le quali sono state implementate dall'autore stesso estendendo il tool DiLeNa presentato da uno studio dell'Università di Bologna.

Nonostante le problematiche emerse siano già molteplici, una complicanza non di poco conto è il costo a livello di tempo che le simulazioni e i test su Bitcoin richiedono. Per scaricare 30 secondi di dati sono necessari circa 10 minuti. Per questo motivo, nel *capitolo tre*, vengono illustrate e discusse tutte le metriche relative a Bitcoin con il fine di implementare un simulatore che comprenda non lo strato baso della blockchain, ma solo quello più elevato, ovvero quello necessario all'analisi di reti sociali.
Tale simulatore è stato scritto in Python e rispecchia tutte le metriche presenti nello Stato dell'Arte, quindi può considerrsi sicuramente un punto di partenza per degli sviluppi futuri.
Il *quarto capitolo* è quello dove sono illustrati i risultati in seguito a delle simulazioni nel tentativo di rilevare attività di riciclaggio all'interno del simulatore.

Ancor prima di ciò, però, è stato presentato l'algoritmo di tipo deterministico volto allo scoprire percorsi chiusi che riportavano le criptomonete allo stesso nodo di partenza. L'approccio impiegato è di tipo investigativo, difatti bisogna presupporre che un nodo sia malevolo e successivamente esplorare la sua sottorete. In merito alla suddetta sottorete, alla fine della dissertazione ci si concentra sulla dimensione che l'ego-network dovrebbe assumere.
Per quanto riguarda i **risultati**, sono quelli attesi, sia in merito alla correttezza del simulatore che ai percorsi di riciclaggio individuati dall'algoritmo.

In merito agli **sviluppi futuri**, ovviamente se ne prevedono di diversi. Innanzitutto è necessario *ideare e sviluppare euristiche più recenti* che possano contrastare i processi di mixing; occorre *studiare i mixer centralizzati* per scoprirne le varie caratteristiche al fine di identificare dei pattern all'interno della rete; per quanto riguarda i mixer decentralizzati, invece, si tratta di transazioni omogenee che per loro natura differiscono dalle transazioni standard presenti su Bitcoin, quindi sono facilmente indivisuabili. Sicuramente il fine ultimo di tutto questo lavoro sarebbe quello di *addestrare una rete neurale* tramite le simulazioni e i dati raccolti sia dal simulatore sia dalla rete Bitcoin. Un'intelligenza artificiale non deterministica e sempre attiva sarebbe di grande aiuto alla lotta contro questa parte di cyber criminalità. Per ultimo, sarebbe utile *estendere il simulatore completo ad altre blockchain*, così da lavorare in maniera ancor più efficace.

Bibliography

[1] *All simple Path*. url: https://networkx.org/documentation/stable/reference/algorithms/simple_paths.html.

[2] Israa Alqassem, Iyad Rahwan, and Davor Svetinovic. "The Anti-Social System Properties: Bitcoin Network Data Analysis". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* PP (Dec. 2018), pp. 1–11. doi: 10.1109/TSMC.2018.2883678.

[3] Israa Alqassem, Iyad Rahwan, and Davor Svetinovic. "The Anti-Social System Properties: Bitcoin Network Data Analysis". In: *IEEE Transactions on Systems, Man, and Cybernetics: Systems* (2020).

[4] *Bitcoin Pizza Day: 10 anni fa la pizza più costosa della storia di BTC*. url: https://cryptonomist.ch/2020/05/22/bitcoin-pizza-day-10-anni/.

[5] Usman Chohan. "Assessing the Differences in Bitcoin & Other Cryptocurrency Legality Across National Jurisdictions". In: *SSRN Electronic Journal* (2017).

[6] Usman Chohan. "The Double Spending Problem and Cryptocurrencies". In: *SSRN Electronic Journal* (2017).

[7] *CoinJoin: Bitcoin privacy for the real world*. url: https://bitcointalk.org/index.php?topic=279249.0.

[8] *Coinjoins as a percentage of all bitcoin payments have tripled to 4.09% over the past year*. 2019. url: https://en.longhash.com/news/coinjoins-as-a-percentage-of-all-bitcoin-payments-have-tripled-to-409-over-the-past-year.

[9] *Cosa sono le blockchain fork?* url: https://www.cmcmarkets.com/it-it/impara-come-operare-con-criptovalute/cosa-e-una-blockchain-fork.

[10] *Ego Graph*. url: https://networkx.org/documentation/stable/reference/generated/networkx.generators.ego.ego_graph.html.

[11] D'Angelo Gabiele Ferretti Stefano Serena Luca. *DiLeNA: Distributed Ledger Network Analyzer*. 2020.

[12] Andrea Fronzetti Colladon and Elisa Remondi. "Using social network analysis to prevent money laundering". In: *Expert Systems with Applications* (2017). url: https://www.sciencedirect.com/science/article/pii/S0957417416305139.

[13] *Graph Size*. url: https://networkx.org/documentation/stable/reference/classes/generated/networkx.Graph.size.html.

[14] Martin Harrigan and Christoph Fretter. "The Unreasonable Effectiveness of Address Clustering". In: *2016 Intl IEEE Conferences on Ubiquitous Intelligence & Computing, Advanced and Trusted Computing, Scalable Computing and Communications, Cloud and Big Data Computing, Internet of People, and Smart World Congress (UIC/ATC/ScalCom/CBDCom/IoP/SmartWorld)* (2016). url: http://dx.doi.org/10.1109/UIC-ATC-ScalCom-CBDCom-IoP-SmartWorld.2016.0071.

[15] Martin Harrigan and Christoph Fretter. "The Unreasonable Effectiveness of Address Clustering". In: *CoRR* (2016). url: http://arxiv.org/abs/1605.06369.

[16] Abraham Hinteregger. "Monero Cross-Chain Traceability". PhD thesis. 2018. url: https://www.researchgate.net/publication/328571978_Monero_Cross-Chain_Traceability.

[17] Younggee Hong et al. "A Practical De-mixing Algorithm for Bitcoin Mixing Services". In:

[18] *Il numero di Bitcoin nel servizio anonimo Whirlpool del portafoglio Samourai è triplicato in un mese*. url: https://it.0xzx.com/2020041273214.html#.

[19] "Index". In: *Handbook of Digital Currency*. Ed. by David Lee Kuo Chuen. Academic Press. isbn: 978-0-12-802117-0.

[20] *Median Transaction Value*. url: https://academy.bytetree.com/economic-indicators/median-transaction-value/explanation.html.

[21] *Minimum Limit for BTC Transactions*. url: https://paxful.com/support/en-us/articles/360011064140-Minimum-Limit-for-BTC-Transactions.

[22] Arvind Narayanan et al. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016. isbn: 0691171696.

[23] T. Neudecker. "Security and Anonymity Aspects of the Network Layer of Permissionless Blockchains". In: 2019.

[24] *Path Length*. url: https://it.wikipedia.org/wiki/Grafo.

[25] Andrea Piroddi. *Introduction to social network analysis*. 2019.

[26] Dhiren Patelb Pranav Nerurkarab. *Dissecting bitcoin blockchain: Empirical analysis of bitcoin network (2009–2020)*. 2020.

[27] D. Savage et al. "Detection of money laundering groups using supervised learning in networks". In: *ArXiv* (2016).

[28] "Scaling properties of extreme price fluctuations in Bitcoin markets". In: *Physica A: Statistical Mechanics and its Applications* (2018).

[29] *Sending and Receiving Bitcoin - Limits*. url: https://cash.app/help/us/en-us/31021-sending-and-receiving-bitcoin#:~:text=Sending%5C%20Bitcoin%5C%20has%5C%20a%5C%20few,or%5C%20100%5C%2C000%5C%20sats%5C%20(Satoshis).%5C&text=Learn%5C%20more%5C%20about%5C%20Bitcoin%5C%20here..

[30] *The Reachability Matrix*. url: http://olizardo.bol.ucla.edu/classes/soc-111/textbook/_book/4-6-sec-reachmat.html#sec:reachmat.

[31] *Top 15 Bitcoin Mixers In 2021*. url: https://www.europeanbusinessreview.com/top-15-bitcoin-mixers-in-2021/.

[32] *Top 7 Bitcoin Mixers in 2021*. url: https://worldfinancialreview.com/top-7-bitcoin-mixers-in-2021/.

[33] *What is the minimum amount of cryptocurrency that I can purchase?* url: https://bitflyer.com/en-eu/faq/55-4/.

[34] *What is the minimum Bitcoin transaction amount on Buy/Sell?* url: https://bitflyer.com/en-eu/faq/55-4/.

[35] Lei Wu et al. "Towards Understanding and Demystifying Bitcoin Mixing Services". In: *Proceedings of the Web Conference 2021* (2021). url: http://dx.doi.org/10.1145/3442381.3449880.