Impact of graph-based features on Bitcoin prices

Anouk van Schetsen
June 2019

Impact of graph-based features on Bitcoin prices

by

Anouk van Schetsen

to obtain the degree of Master of Science in Applied Mathematics
at the Delft University of Technology,
to be defended publicly on Friday June 28, 2019 at 15:00 AM.

Student number:	115796	
Project duration:	September 15, 2018 – June 28, 2019	
Thesis committee:	Prof. C. W. Oosterlee,	TU Delft, supervisor
	Prof. C. Kraaikamp,	TU Delft
	A. Fontanari,	TU Delft
	M. H. Oeben,	PwC

TU Delft

Abstract

Predicting the trends in Bitcoin market prices is a very challenging task due to the many uncertainties and variables influencing the market value. The market is susceptible to quick changes, causing seemingly random fluctuations in the Bitcoin price. Due to the chaotic and highly volatile nature of Bitcoin behavior, investments come with high risk. To minimize the risk involved, knowledge of the Bitcoin price movement in the future is desirable.

Different studies have shown that Machine Learning algorithms can predict, to varying degrees, the price fluctuations of Bitcoin. However, most researches do not explore the relationship between the price and other features outside the transaction network, such as market capitalization, Bitcoin mining speed, or entity behavior. Also, most of the features are extracted from the network level, which means obtaining the number of transactions, users, Bitcoins mined, etc. In this research, we focus on additional features, such as features outside the transaction network and node-based features inside the transaction network, which could improve the price prediction of Bitcoin.

The investigated features are the "fairness and goodness" measure and the "1-ARW-betweenness centrality" measure. Fairness and goodness are entity behavior measures. The goodness of a Bitcoin address captures how much this address is liked/trusted by other addresses, while the fairness of a Bitcoin address captures how fair the address is in rating other addresses' likeability or trust level. The 1-ARW-betweenness centrality is a feature based on absorbing random walks. The feature captures the extent to which a Bitcoin address has control over the money flow between different addresses.

A benchmark, based on the machine learning algorithm Random Forest with commonly used features, is used to test the impact of the additional features. The Random Forest tries to predict the sign (up-down movement) of the price per day, using data from the two previous days. Comparing this benchmark with a similar model, but then including the additional features, will gain more information about how these additional features influence the Bitcoin price.

Preface

This thesis has been submitted for the degree of Master of Science in Applied Mathematics at Delft University of Technology. The academic supervisor of this research is Kees Oosterlee, professor at the Numerical Analysis group of Delft Institute of Applied Mathematics. Also, a second supervisor, Andrea Fontanari, was appointed from the Delft University of Technology. The research has been conducted under the supervision of Marvin Oeben at the PwC office in Amsterdam.

I want to thank Kees Oosterlee, Andrea Fontanari, and Marvin Oeben for their time, valuable input and feedback during this project. Furthermore, I would like to thank Cor Kraaikamp for being part of the thesis committee.

Anouk van Schetsen
June 2019

Contents

Abstract **1**
Preface **3**

Introduction **7**

1 Background **9**
1.1 Bitcoin . . . 9
1.2 Frequently used terms and definitions . . . 10

2 Data **11**
2.1 Who-trust-whom network: Bitcoin OTC dataset . . . 11
2.1.1 Web scraping to improve the dataset . . . 12
2.1.2 Network properties . . . 14
2.1.3 Time dependence . . . 16
2.1.4 Rating analysis . . . 17
2.2 Transaction network: Bitcoin MIT dataset . . . 18
2.2.1 Construction of the Bitcoin MIT dataset . . . 18
2.2.2 Network properties . . . 19
2.3 Time-series: Bitcoin Coinbase dataset . . . 20
2.4 Combining all datasets . . . 20

3 Betweenness centrality based on absorbing random walks **21**
3.1 Economic interpretation . . . 21
3.2 Random walks . . . 22
3.3 Absorbing random walks . . . 24
3.4 The 1-ARW-betweenness centrality measure . . . 26
3.5 Algorithm . . . 28
3.5.1 Time complexity . . . 28
3.5.2 Restart probability . . . 29
3.6 Implementation on a subset of the Bitcoin MIT dataset . . . 30
3.7 Summary & Conclusion . . . 31

4 Predicting the direction of Bitcoin prices **32**
4.1 Related work . . . 32
4.2 Classification problem . . . 32
4.3 Random Forest . . . 33
4.3.1 Decision Tree . . . 33
4.3.2 Random Forest Classifier . . . 36
4.3.3 Feature importance . . . 38
4.4 Data preparation . . . 39
4.4.1 Training, Validation and Testing set . . . 40
4.4.2 Data collection . . . 40
4.4.3 Window sampling . . . 41
4.4.4 Feature Engineering . . . 42
4.4.5 Snowball Sampling . . . 46
4.4.6 Data merging . . . 48
4.5 Impact of snowball sampling on the classification model . . . 50
4.6 Implementation on the Bitcoin MIT dataset . . . 52
4.6.1 Benchmark . . . 52
4.6.2 Improvement with use of node-based features . . . 55

Conclusion **57**
Future work **58**
References **59**

A Fairness and Goodness **63**
A.1 Economic interpretation . . . 63
A.2 Fairness and Goodness in formula . . . 63
A.3 Independence . . . 65
A.4 Algorithm . . . 68
A.4.1 Convergence and Uniqueness . . . 69
A.4.2 Time complexity . . . 73
A.4.3 Choosing starting values and stopping criterion . . . 74
A.4.4 Accuracy . . . 76
A.5 Missing values . . . 77
A.5.1 Prediction based on a spectral embedding . . . 78
A.5.2 Predictive models based on inverse scoring Fairness and Goodness . . . 86
A.5.3 Accuracy of the predictive models . . . 90
A.6 Implementation on the Bitcoin OTC dataset . . . 91
A.7 Summary & Conclusion . . . 92

B Code **93**
B.1 Web scraper to construct the Bitcoin OTC dataset . . . 93
B.2 Constructing the Bitcoin MIT dataset . . . 93
B.3 1-ARW-betweenness centrality measure . . . 95
B.4 Snowball sampling . . . 95
B.5 Extracting window features . . . 97
B.6 Extracting node-based features . . . 99
B.7 Fairness and goodness measure . . . 101

Introduction

Networks (or graphs) appear as dominant structures in diverse domains, including sociology, biology, neuroscience and computer science. Graphs are mathematical structures used to model pairwise relations between objects, where the objects are called the nodes, and the relations are called the edges. In most cases, graphs are weighted and directed, as the source node transmits a property to the delivered node but not vice versa.

Well-known graphs are the transaction networks, which represent all incoming- and outgoing transactions between individual entities. Since most financial institutions protect their transaction data, limited academical research has been devoted to real-life networks. A publicly available transaction network is the Bitcoin BlockChain. Bitcoin is a cryptocurrency, which can be sent from user-to-user without the need for intermediaries. The concept is that its users mutually verify Bitcoin transactions and that all transactions are recorded in a public distributed ledger called a BlockChain.

Different studies have shown that Machine Learning algorithms can predict, to varying degrees, the price fluctuations of Bitcoin. However, most researches do not explore the relationship between the Bitcoin price and other features outside the transaction network, such as market capitalization, Bitcoin mining speed, or entity behavior. Also, most of the features are extracted from the network level, which means obtaining the number of transactions, users, Bitcoins mined, etc. These features dont capture complex relationships between the nodes in the Bitcoin transaction network. In this research, we focus on additional features, such as features outside the transaction network and node-based features inside the transaction network, which could improve the price prediction of Bitcoin.

Outline of the thesis

In chapter 1, some background information towards the Bitcoin is given. Also, the frequently used terms and mathematical definitions, inside this research are listed.

In chapter 2, different types of Bitcoin datasets are introduced; a who-trust-whom network, a transaction network, and a time-series representing Bitcoin prices. For each type of dataset, there is an explanation on how they are extracted from their sources, and some statistical properties are proven. Also, we explain how to combine all datasets to obtain a combined network containing information from all three datasets. The additional features, possible improving price prediction, are based on this combined dataset. During combining all datasets it is discovered that the who-trust-whom network can't be linked to the other datasets, and therefore the feature (fairness and goodness) based on the who-trust-whom network will be discussed in the Appendix. However, since we showed some interesting new methodologies in light of the measure, it is still included in the research and one of the most essential suggestions for further research.

In chapter 3, a feature based on absorbing random walks is introduced; the 1-ARW-betweenness centrality. The 1-ARW-betweenness centrality captures the extent to which a Bitcoin address has control over the money flow between different addresses. The algorithm of computing the measure is defined, and its time complexity and input variables are discussed. At last, the measure is implemented on a small transaction network.

In chapter 4, price prediction with the use of a Random Forest classification model is discussed. The price prediction is made daily to predict the sign (up-down) of the Bitcoin price. To test the impact of the additional features, a benchmark is constructed based on commonly used features. Comparing the benchmark with a similar model, including the additional features, will gain more information about how these additional features influence the Bitcoin price.

In the chapters, Conclusion and Future work, a conclusion of the achieved results and suggestions for further research opportunities are discussed.

In Appendix A, the feature 'fairness and goodness' is introduced. Since this measure won't contribute to the actual price prediction, it is discussed in the Appendix. Fairness and goodness are entity behavior measures, where the goodness of a Bitcoin address captures how much this address is liked/trusted by other addresses, while the fairness of a Bitcoin address captures how fair the address is in rating other addresses' likeability or trust level. The proof of the dependence of the fairness score concerning the goodness scores and vice versa is given. Also, the algorithm and some of its properties are discussed. If nodes inside the network lack any in- or outgoing edges, the algorithm can't update properly, causing missing values to occur. There is explained how to approximate these missing values. At last, the entity behavior measures are implemented on a who-trust-whom network.

In Appendix B, some codes are given which are used to prepare the data.

1 Background

1.1 Bitcoin

Bitcoin is a cryptocurrency, which can be seen as a form of electronic cash. Bitcoin transactions can be sent from user-to-user on the peer-to-peer bitcoin network without the need for intermediaries. The concept is that its users mutually verify Bitcoin transactions and that all transactions are recorded in a public distributed ledger called a BlockChain.

BlockChain. A BlockChain is a chain of blocks that stores information. The technique was originally introduced in 1991 by a group of researchers. The technique is originally intended to timestamp digital documents so that it's not possible to tamper with them. However, it went by unused until it was adapted by Satoshi Nakamoto in 2009 to create the digital cryptocurrency Bitcoin [51]. BlockChains have an interesting property: once some data has been recorded inside the BlockChain it becomes nearly impossible to change it. This is related to three concepts; hashes, proof of work, and p2p networking.

Hashes. Each block inside the BlockChain contains data, the hash of the block, and the hash of the previous block. The information that is stored inside a block depends on the type of BlockChain. The Bitcoin BlockChain saves the details about a transaction, such as a sender, receiver, amount of Bitcoin in the transaction. The hash of a block is comparable to a fingerprint, which is always unique for a specific block. Once a block is created the corresponding hash is being calculated. Changing the information inside the block will cause the hash to change as well. The block is no longer the same block. The last element inside each block is the hash of the previous block, and this effectively creates a chain of blocks. This technique makes a BlockChain so secure [24].

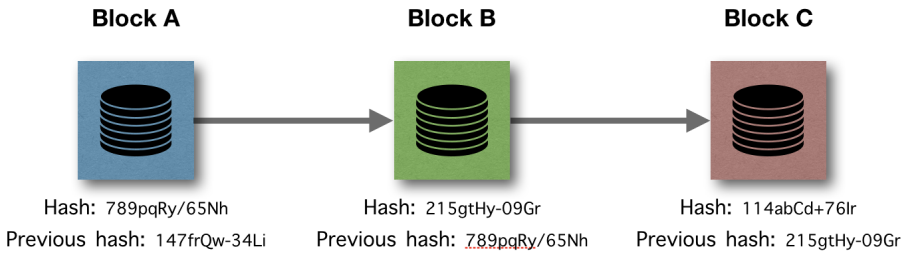


Figure 1: Example of a BlockChain with three blocks; A,B and C. Below each block its hash and previous hash is stated.

In Figure 1 an example chain of three blocks is visualized, where each block contains a hash and the hash of the previous block. So block C points to block B, block B points to block A, etc. Now let's say that somebody tampers with block B; this causes the hash of the block to change as well. It will make block C and all subsequent blocks invalid because block C no longer stores a valid hash of its previous block.

Proof of work. Hashes are not enough to prevent tampering. Computers these days are incredibly fast and can calculate hundreds of thousands of hashes per second. To avoid this, BlockChains has a property called 'proof of work'. Proof of work is a mechanism that slows down the creation of new blocks, for Bitcoins a block takes about 10 minutes to calculate the required proof of work and add a new block to the chain. Since tampering with one block will cause to recalculate the proof of work for all the following blocks, proof of work contributes to the security of the BlockChain [31].

P2P network. Another way how BlockChains secure itself is its publicity. BlockChains uses a peer-to-peer network, in which everyone is allowed to join the network. When an individual joins this

network, he or she receives a full copy of the BlockChain. With the use of this copy, anyone can verify the current BlockChain.

Anonymity. Bitcoins built-in privacy protections allow users to separate their Bitcoin addresses from their public personas completely. This causes anonymity while trading Bitcoin. While it is possible to track Bitcoin flows between addresses, it's challenging to figure out which individuals represent those addresses. Also, individuals could have multiple Bitcoin addresses, each influencing the Bitcoin transaction network in a different way.

1.2 Frequently used terms and definitions

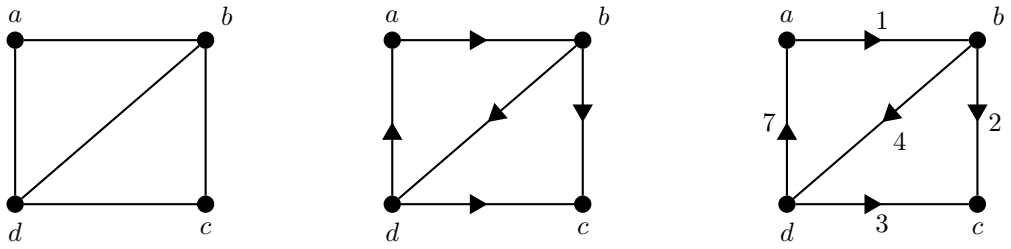
Satoshi. The unit Satoshi is the smallest tradable Bitcoin amount of 0.00000001 Bitcoin (1e-8), named after the founder Satoshi [75].

Epoch time. Epoch time is an easy way to track time as a running total of seconds. This count starts at the Unix Epoch on January 1st, 1970 at UTC. The Unix timestamp is merely the number of seconds between a particular date and the Unix Epoch. The Unix Epoch technically does not change no matter where you are located on the globe. This is very useful to computer systems for tracking and sorting dated information in dynamic and distributed applications [11].

Definition 1.1. A **graph** is a pair $G = (V, E)$ of sets such that $E \subseteq V \times V$; thus, the elements of E are 2-element subsets of V. The elements of V are called the **nodes** of the graph, and the elements of E are called the **edges** of the graph.

Definition 1.2. A graph $G = (V, E)$ is **directed** if the set of edges are directed, such that $(u, v) \in E$ doesn't imply $(v, u) \in E$.

Definition 1.3. A pair of sets $G = (V, E, w)$ is called a **weighted directed graph** if the graph is directed and the edges contain a **weight function** $w : E \to \mathbb{R}$.



(a) Graph $G = (V, E)$

(b) Directed graph $G = (V, E)$

(c) Weighted directed graph $G = (V, E, w)$

Who-trust-whom network. A who-trust-whom network (WSN) is characterized by their property of capturing like/dislike, trust/distrust, and other social relationships between entities, mostly by their edges being either negatively or positively weighted [42]. Mathematically, a WSN is a weighted directed graph $G = (V, E, w)$ with V nodes, E edges and w as the weight function over the edges displaying the social relationship between the nodes.

Transaction network. A transaction network represents all incoming- and outgoing transactions between individual addresses. Mathematically, a transaction network is a weighted directed graph $G = (V, E, w)$ with V nodes, E edges and w as the weight function. All entities inside the transaction network are the nodes V, and the made transactions are the edges E. The direction of the edges represents the flow of money and the amount of the transaction is used as weights w.

2 Data

Different sources of Bitcoin data can be used to enrich machine learning algorithms. The most commonly used data is a simple transaction network containing all information about Bitcoin transaction and their involved variables, such as the sender, receiver, and the amount of Bitcoin. In total three types of Bitcoin datasets are used in this project;

1. A who-trust-whom network.
2. A transaction network.
3. A time-series of the Bitcoin price.

A who-trust-whom network is characterized by its property of capturing like/dislike, trust/distrust, and other social relationships between entities, mostly by their edges being either negatively or positively weighted [42]. We also refer to who-trust-whom networks as weighted signed networks (WSNs). A transaction network represents all incoming- and outgoing transactions between individual addresses. At last, the time-series of the Bitcoin price lists the Bitcoin prices of the Coinbase exchange platform. It is called a time-series since it simply represents a series of price points ordered in time.

The transaction network is required from the Massachusetts Institute of Technology, which we call the Bitcoin MIT dataset. The who-trust-whom network is found on a Bitcoin trading platform OTC, and this dataset is called the Bitcoin OTC dataset. At last, the time-series of the Bitcoin is scrapped from the Coinbase exchange platform, which we call the Bitcoin Coinbase dataset accordingly.

In the upcoming sections, each type of dataset is discussed. We will elaborate on how the networks are constructed from their sources. And, some statistical tests are made towards the networks, to prove, for example, its stationarity over time. In the last section is argued how to combine all datasets to obtain a combined network containing information from all three datasets.

2.1 Who-trust-whom network: Bitcoin OTC dataset

The who-trusts-whom network is based on people who trade using Bitcoin on a platform called Bitcoin-otc[1]. Since they are anonymous, there is a need to maintain a record of users' reputation to prevent transactions with fraudulent and risky users. All transactions are managed directly between counterparties, without any intermediation from the Bitcoin-otc platform. As such, it is each responsibility to conduct due diligence on their counterparties and to complement the marketplace. The platform offers a web of trust service, based upon scoring their counterparties reputation. Members of Bitcoin-otc rate other members on a scale of -10 (total distrust) to +10 (total trust).

In 2016, Stanford University constructed and publicly shared the who-trust-whom network of the Bitcoin-otc platform. This dataset is found on the SNAP dataset inventory, under the name 'Bitcoin OTC trust weighted signed network' [77].

The SNAP dataset provided four entries per line; source, target, rating, and timestamp. *Source* notated the address id of the source, *target* the address id of the target, *rating* the source's rating for the target, ranging from -10 to +10 and *timestamp* the time of the rating, measured as seconds since Epoch. A weighted signed network $G = (V, E, w)$ is constructed from the dataset. All sources and target addresses are the nodes V in the WSN. The made transactions are the edges E, with their ratings as weights w. The source of a transaction is the starting point of the edge, whereas the target is the receiving side of the edge.

[1] `https://www.bitcoin-otc.com`

2.1.1 Web scraping to improve the dataset

The Bitcoin-otc website contains a page called the `#bitcoin-otc gpg key data`, here a list of all key user data is defined (Figure 3). This list contains fields as ID, nick, keyID, but most interesting; bitcoinaddress. The Bitcoin address is a unique string of characters and letters used to identify users in the Bitcoin BlockChain. With the use of these Bitcoin addresses, the who-trust-whom network can be connected to the Bitcoin transaction network.

id	nick	registered at (UTC)	keyid	fingerprint	bitcoinaddress
7790	#bitcoin-otc	2013-04-10 21:22:13	32EC272A73778423	19D183E30039EF31C34766A032EC272A73778423	
10942	'hightides'	2014-01-14 20:34:03			19vhqyEZQuAj7ue9NcivCfr4n2TQ734C1D
2963	1beacon	2011-11-05 12:58:09	AF65AE980C825691	BFD626E9E7E33BEFAAE1773AAF65AE980C825691	
9533	1dannyboy	2013-08-12 21:28:09			1HFCYaUgeZvNfNbL9QZDMqRaDQxHE11LXh
4771	1MjDZAW	2012-09-08 15:57:36			1MjDZAWjUfcJFeEbrFJeZwASHrFML5JPF6
4408	2fast4you	2012-08-02 01:23:45	15781F58CDB2035C	4AD605094A71F58A288798E915781F58CDB2035C	
7654	3phx	2013-04-08 20:27:19			16hoadnFtLeshvwocA8WfPvSbuhLDsfvMp

Figure 3: Screenshot of the '#bitcoin-otc gpg key data' page on the Bitcoin-otc webpage.

However, while linking SNAPs who-thrust-whom network to the transaction network, there was discovered that the identity keys used by SNAP were different than the identity keys used on the otc-website. A solution was to manually web-scrape the who-trust-whom network from the Bitcoin-otc website, creating two main advantages; more up-to-date data and the users' Bitcoin addresses.

A web scraper was build in `R`, a programming language and free software environment for statistical computing and graphics, with the use of a pre-defined package `jsonlite`. This package offers tools for working with JSON (JavaScript Object Notation) in `R` and is particularly powerful for building pipelines and interacting with a web API [78]. In the following paragraph, the outline of the scraping tool is explained. After that, some comparisons are made between the old and new who-trust-whom network, and at last, something is said about the found Bitcoin addresses.

Outline web scraping tool. For re-building the SNAP dataset, we are interested in the source, target, and timestamp of all individual ratings given on the platform. The Bitcoin-otc website stores its ratings on the `viewratingdetail` page of its users. The JSON object of this type of page can be extracted through the link:

https://www.bitcoin-otc.com/viewratingdetail.php?nick=**nickname**&outformat=json,

with the bold text being a nickname of a subscribed user of the platform. All nicknames can be extracted from the formerly named `#bitcoin-otc gpg key data` page of the website. The outline of our web scraper is pretty straightforward; it finds all ratings by looping through the rating detail page of all users.

While looping through a single rating detail page, our web scraper builds two datasets; the Bitcoin OTC dataset and the parent OTC dataset. In the Bitcoin OTC dataset the source, target, rating and timestamp (in Epoch) of each individual rating is saved, such that the dataset has the same structure as the original SNAP dataset. The parent OTC dataset provides three entries per line; id, bitcoinaddress and nick. The field *id* notates the unique identity key of a user, which matches with the data in the Bitcoin OTC dataset. And naturally, the fields *bitcoinaddress* and *nick* stands for the corresponding Bitcoin address and nickname of a user.

The full code of the web scraper is stored in Appendix B.1.

Statistics of improved dataset. Since the structure of the Bitcoin OTC dataset is similar to the SNAP dataset, a weighted directed graph can be assembled similarly: Take all sources and targets as nodes for the graph. Made transactions are the edges of the graph, with their ratings as weights. The source of a transaction is the starting point of the edge, whereas the target is the receiving side of the edge.

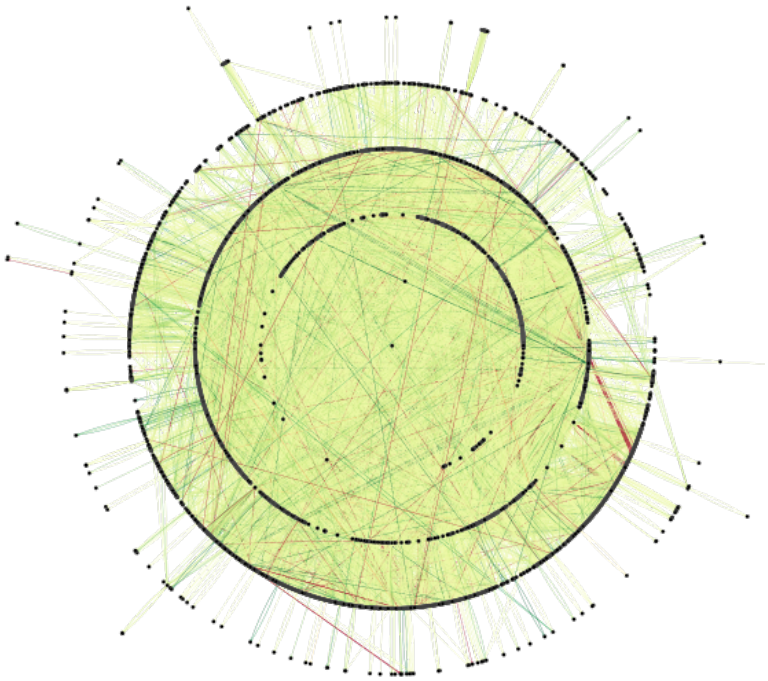


Figure 4: Visualization of a weighted directed graph, based on the Bitcoin OTC network. The graph contains 5.986 nodes and 36.108 edges. The coloring of the edges visualize their weights (on the interval [-10,10]), where red is the lowest possible value and green the highest.

In Table 1 some statistics of the newly scrapped dataset are given, and in Figure 4, the Bitcoin network is visualized, where the colors indicate the weights/ratings of the network.

Total users (nodes): 5.986	Start:	8 Nov 2010				
Total transactions(edges): 36.108	End:	9 Jan 2019				
	Median	**Mean**	**Sd**	**Skew**	**Min**	**Max**
Rating value	1	1,02	3,57	-1,49	-10	10
Number of users per month	93	170	187	1,20	2	762
Number of transactions per month	145	372	486	1,68	1	2.205

Table 1: Basic statistics of the Bitcoin OTC dataset, result of the web scraping tool.

After scrapping, there is tested if all ratings of the SNAP dataset are included in the new data. This was indeed the case. The SNAP dataset ran between 8 Nov 2010 until 25 Jan 2016, with a total of 5.681 nodes and 33.592 edges. Our newly constructed networks include additional twenty-five-hundred ratings and three-hundred users in three years. This is on average less than previous recordings, probably due to a various set of reasons such as lower user retention, caused by an up-rise of competing platforms.

2.1.2 Network properties

Complex networks, such as the who-trust-whom network, have received a tremendous amount of attention in the past decade. Due to the increased capability of computational power, large datasets can now easily be stored and investigated. Most real networks share fascinating properties, such as being 'small world' or 'scale-free'. Being small world means that in some sense, most nodes are separated by relatively short chains of edges. And, being scale-free means that their degrees are size-independent.

During network analysis, we encountered that the graph was not connected. Approximated 0,2% of the nodes are not connected to the biggest component. Since a lot of methodologies obliges connectivity, we discarded these nodes and only considered the biggest component.

Small worlds. The first fundamental network property is called the 'small-world' phenomenon [66].

Definition 2.1. Given is a network with $n \in \mathbb{N}$ nodes, let $L \in \mathbb{N}$ be the distance between two randomly chosen nodes in the network. The network is called **small-world** if L grows proportionally to the logarithm of the number of nodes n, such that;

$$L \propto \log n,$$

while the clustering coefficient of the network is relatively high [28].

In graph theory, the global clustering coefficient is a measure of the degree to which nodes in a graph tend to cluster together. By one definition, it is equal to the fraction of ordered triples of nodes a, b, c in which edges ab and bc are present that have edge ac present [14]. But what is a relatively high clustering coefficient? A randomly generated network with an equal amount of nodes and edges of the Bitcoin OTC dataset will have a clustering coefficient of approximated 0,0060. The clustering coefficient exceeding this level will be considered relatively large [40]. In the context of who-trust-whom networks, this results in a so-called 'small world' phenomenon of strangers being linked by a short chain of acquaintances. An advantage to small world networking for social ratings is their resistance to change in behavior due to sampling in nodes and edges [66].

In Table 2 is illustrated the shortest path distances between all nodes and one 'most central' node in the WSN. This most central node is chosen such that it has the highest degree. The undirected graph is investigated as well, to derive the difference between association (undirected) and influence (directed).

Shortest path	Undirected	Directed
0	1	1
1	833	799
2	2518	2162
3	2454	2747
4	167	233
5	9	13
6	-	1

Table 2: Length of shortest paths, started at the node with the highest degree.

The found clustering coefficient for the undirected graph is 0,17, and for its directed version 0,15. The relatively high value of the clustering coefficients, together with the fact that shortest path lengths are small, indicates that this graph is a small world graph (for both the undirected and directed graph).

Power-law degree sequence. The second fundamental property of real networks is the phenomenon 'scale-free'. Scale-free indicates that the number of nodes with degree $k \in \mathbb{N}$ falls off as an inverse power of k. This is called a 'power-law degree sequence'. Resulting graphs often go under the name scale-free graphs, which refers to the fact that the asymptotics of the degree sequence is independent of its size [28].

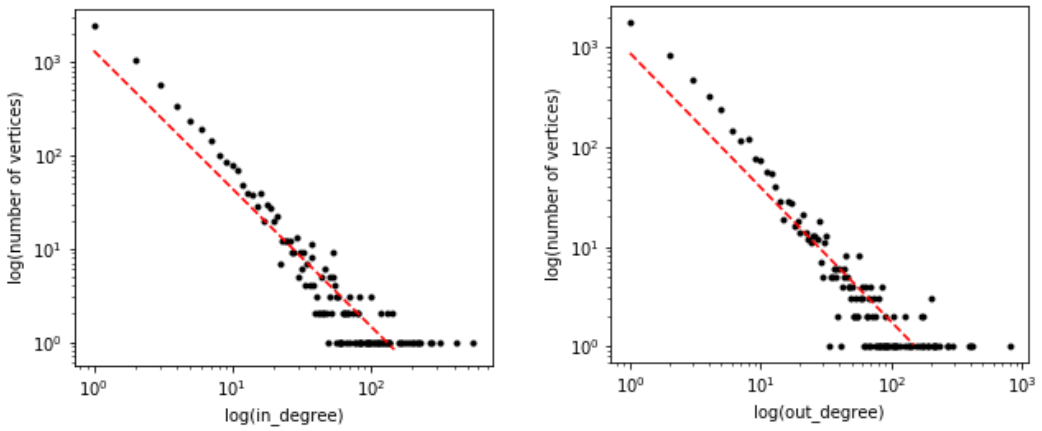
Definition 2.2. Given is a network, let $N_k \in \mathbb{N}$ be the number of nodes in the network with degree $k \in \mathbb{N}$. The network is called **scale-free** if $\exists \gamma \in \mathbb{R}$, such that;

$$N_k \sim k^{-\gamma}.$$

Taking the logarithmic power of both sides provides:

$$\log N_k \sim \gamma \log k$$

So a plot of $\log N_k \mapsto \log k$ should be close to a straight line for scale-free networks. In Figure 5, the power-law degree sequence of the WSN, based on the Bitcoin OTC dataset, is visualized. The points are in the trend of a straight line on the logarithmic scale, which proves the scale-free property of the network. The power exponent γ can be estimated by the slope of the line, and is for the in-degree and out-degree of the WSN, $\gamma_{in} \approx 1,47$ and $\gamma_{out} \approx 1,34$, respectively.



(a) Power-law of in-degree

(b) Power-law of out-degree

Figure 5: Degree sequences for the WSN based on the Bitcoin OTC dataset. It's clear that both sequences are power-law degree sequences since there exists a linear trend in both figures.

2.1.3 Time dependence

Since Bitcoin networks are always evolving and not static, network properties are time-dependent. The first transaction dates on 8 November 2010 and the last transaction on 9 January 2019. Figure 6 plots the number of users and transactions per month against time. There exists a clear peak of transactions and users in mid-2011. Around this time, several articles and media attention, e.g., Forbes, Businessweek, and Bloomberg, were focused on Bitcoins, which caused an increase in Bitcoin trading activities [4].

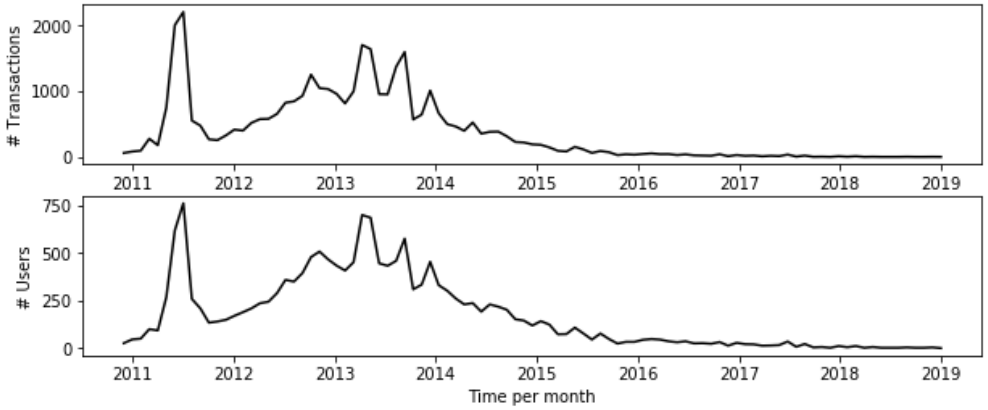
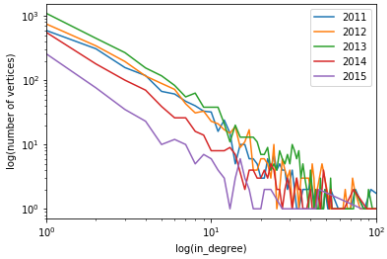


Figure 6: Number of transactions/users per month on the Bitcoin OTC platform over time.

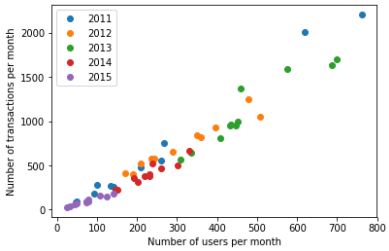
In the previous analysis, the terms small-world and scale-free were introduced. Those properties can also be investigated for different time-dependent subsets of the Bitcoin OTC dataset, to investigate the dynamics of the network even further. For every whole year, since the start of our dataset in 2010, both properties are proved.

year	γ
2011	1,47
2012	1,36
2013	1,38
2014	1,35
2015	1,37

(a) Power-law degree exponents



(b) Degree sequences per year.



(c) Users vs. transactions.

Figure 7: (a) Power-law degree exponent γ for each year in the Bitcoin OTC dataset. (b) Degree sequences for yearly Bitcoin data, it is seen that each year holds a linear trend in the data. (c) Visualization of the number of users versus the number of transactions per day.

In Figure 7b, the resulting yearly degree sequence is plotted on a log-log scale against the number of nodes, having that specific degree. The change in sequences gives insight into network usage over time. For example, at the beginning of Bitcoin trading, there were fewer fluctuations than in the end. Also, with decreasing time, the degree distribution of the network seems to be converting to scale-free

behavior. This could be a result of the large transactions and users peak in 2011 and a decrease in transactions and users afterward. For all years, there still exists a clear linear trend in the data, which implies that each year has a power-law degree sequence.

In Table 8, the clustering coefficients and the length of the shortest paths are visualized for all yearly subsets of the Bitcoin OTC dataset. These properties are used to prove if a network is small world. Due to the time constraint, the yearly sub-networks are not always connected. For unconnected components, the shortest path cannot be calculated with respect to all nodes. For this analysis, only the biggest component of each year network is considered. In Table 8 is described how large these yearly biggest components are, and their corresponding clustering coefficient and shortest paths are listed. It is seen that all yearly subsets are small-world. Only a small percentage of nodes are not in the biggest component, which makes the small world property still reliable.

year	# Users	% Not in biggest comp.	# Trans.	% Not in biggest comp.	Clustering Coefficient	Shortest Path
2011	1570	0,12%	4445	0,02%	0,11	5
2012	1833	0,31%	5803	0,05%	0,14	6
2013	2690	0,37%	8825	0,06%	0,14	6
2014	1246	2,88%	2874	0,63%	0,14	6
2015	414	3,54%	913	0,99%	0,10	4

Figure 8: Yearly biggest components statistics and their clustering coefficient and shortest paths.

2.1.4 Rating analysis

The members of Bitcoin-otc platform rate other members on a scale of -10 (total distrust) to +10 (total trust). Not all ratings are mutually given; this causes unsymmetrical behavior in the WSN. In Figure 9, the distribution of ratings between two individuals is plotted. Most ratings are mutually equal to one, and a total 93% of all ratings are positive.

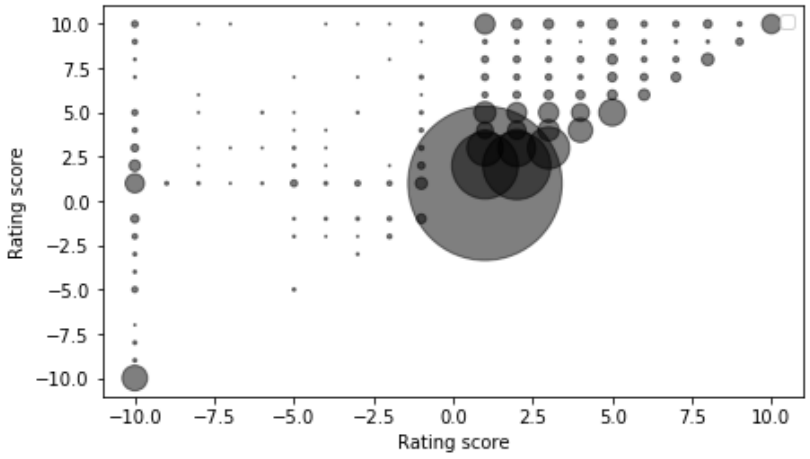


Figure 9: Distribution of ratings inside the Bitcoin OTC dataset.

2.2 Transaction network: Bitcoin MIT dataset

The Bitcoin transaction information is obtained from a third party, the Massachusetts Institute of Technology (MIT). One of their project websites[2] stores all Bitcoin transactions in the first 508241 block of the BlockChain blocks (approximately up to 9 Feb 2018). The project website stores a collection of files, containing information about the block hashes, the Bitcoin addresses, transaction IDs, etc. By using a combination of these files a Bitcoin transaction network can be constructed, which we will call the Bitcoin MIT dataset.

Since constructing the whole transaction network is too computational expensive for normal computers, we only used transaction data between 1 December 2015 and 31 July 2017. The choice of this interval will be discussed in section 2.3

2.2.1 Construction of the Bitcoin MIT dataset

The construction of the Bitcoin MIT dataset is based on combining a selection of files from the project website. This happens in the following steps:

(1) Sample all block hashes between 1 December 2015 and 31 July 2017, with the use of the file `bh.dat`, which is the output file for block hashes.

(2) Generate a transaction overview, with the use of file `tx.dat`, which is the output file for all transaction IDs. Sample only the transactions with the block hashes obtained at step (1).

(3) Only keep the one-to-one transactions in the transaction overview, section 4.4.2 states the importance of only keeping the one-to-one transactions.

(4) Derive from the files `txin.dat` and `txout.dat` all transaction details from the transactions listed in the transaction overview. These files contain the Bitcoin addresses, the unique transaction ID, and their amounts in Satoshi. Satoshi is the smallest tradable Bitcoin amount of 0.00000001 Bitcoin (1e-8) [75].

(5) Combine all the transaction inputs and transaction outputs, from step (4), with the use of the unique transaction ID. This dataset is called the Bitcoin MIT dataset, which will contain a transactionID, in-address, out-address, amount, and timestamp for every transaction.

The full code and details of the data preparation of the MIT dataset are stored in Appendix B.2. In Table 3 some statistics of the Bitcoin MIT dataset are given.

Total users (nodes): 17.340.565	Start:	1 Dec 2015				
Total transactions(edges): 15.335.142	End:	31 July 2017				
	Median	**Mean**	**Sd**	**Skew**	**Min**	**Max**
Amount	18.439	26.744	164.683	1.970	1	40.000.000
Number of Bitcoin addresses per month	912.424	934.079	258.043	-0,39	444.204	1.366.838
Number of transactions per month	745.919	717.747	204.788	-0,50	325.111	1.066.306

Table 3: Basic statistics of the Bitcoin MIT dataset, result of the web scraping tool.

[2] `https://senseable2015-6.mit.edu/bitcoin/`

From the Bitcoin MIT dataset to a weighted- directed graph. The Bitcoin MIT dataset provides 5 entries per line; transactionID, in-address, out-address, amount and timestamp. The *transactionID* is the unique index of a transactions, *in-address* notates the node id of the sending Bitcoin address, *out-address* the node id of the receiving Bitcoin address, *amount* the amount of Bitcoin transferred in Satoshi, and *timestamp* the time of the transaction, measured as seconds since Epoch.

From the Bitcoin MIT dataset, a weighted directed graph $G = (V, E, w)$ can be constructed by taking all in/out-addresses as nodes V and the made transactions as the edges E. The direction of the edges is from in-address to out-address, and their amounts are used as weights w. The graph can be constructed for different subsets of the Bitcoin OTC dataset.

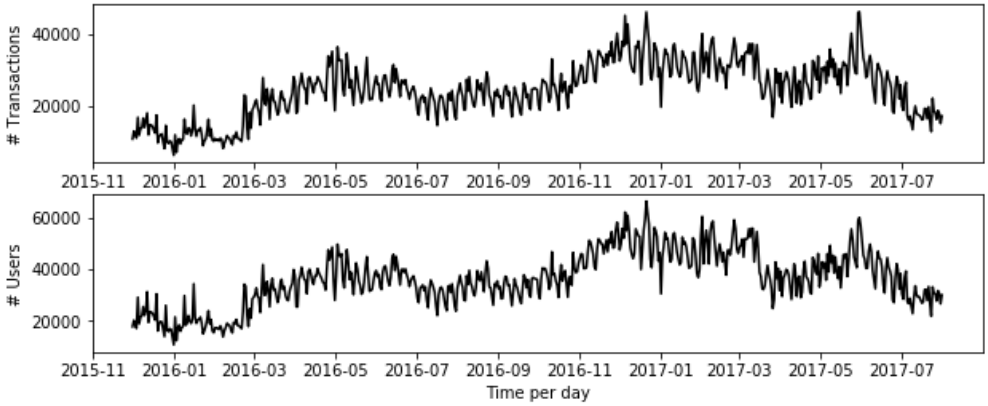


Figure 10: Number of transactions/users per day in the Bitcoin MIT dataset over time.

2.2.2 Network properties

Also for the Bitcoin MIT dataset, the property scale-free (Definition 2.2) is investigated. Being scale-free meant that their degrees are size-independent. In Figure 11, the power-law degree sequence of the weighted directed graph, based on the Bitcoin MIT dataset, is visualized. The points are in trend of a straight line on a logarithmic scale, which proves the scale-free property of the network.

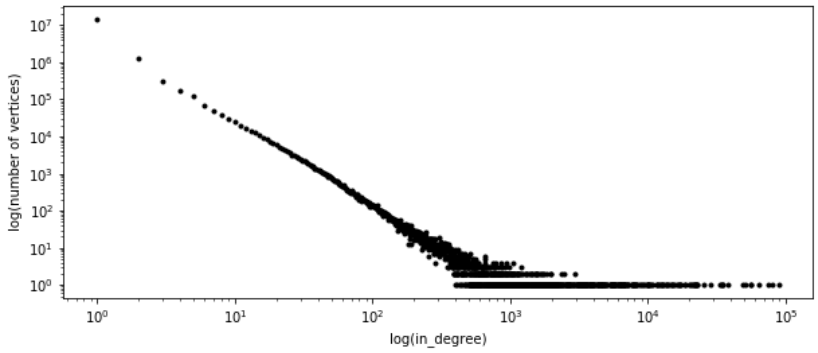


Figure 11: Degree sequences for the weighted directed graph based on the Bitcoin MIT dataset. It's clear that the sequence is a power-law degree sequence since there exists a linear trend in the figure. The power-law degree exponent is $\gamma = 1, 22$.

2.3 Time-series: Bitcoin Coinbase dataset

A time-series is used to represent the Bitcoin price, in which a time-series is just a series of data ordered in time. A complete historical listing of Bitcoin prices is used. On Bitcoincharts[3], the entire history of several Bitcoin exchanges are stored. Due to the completeness, the Bitcoin exchange platform Coinbase in EUR is chosen. Each line in the Coinbase dataset provides a timestamp in Epoch and the exchange rate in EUR.

In Figure 12, the price evaluation of Bitcoin is given, based on the price of the Coinbase exchange platform. The Bitcoin price widely fluctuates over short periods. For example, in late 2017, the Bitcoin value doubled several times, reaching nearly 17.500 Euro per Bitcoin. This period is called the Bitcoin Bubble. Afterward, in the first weeks of 2018, the price decreased massively, which caused billions of market value to disappear. Since price predictions are unreliable on high volatile time-series, the research focuses on data before the Bitcoin Bubble.

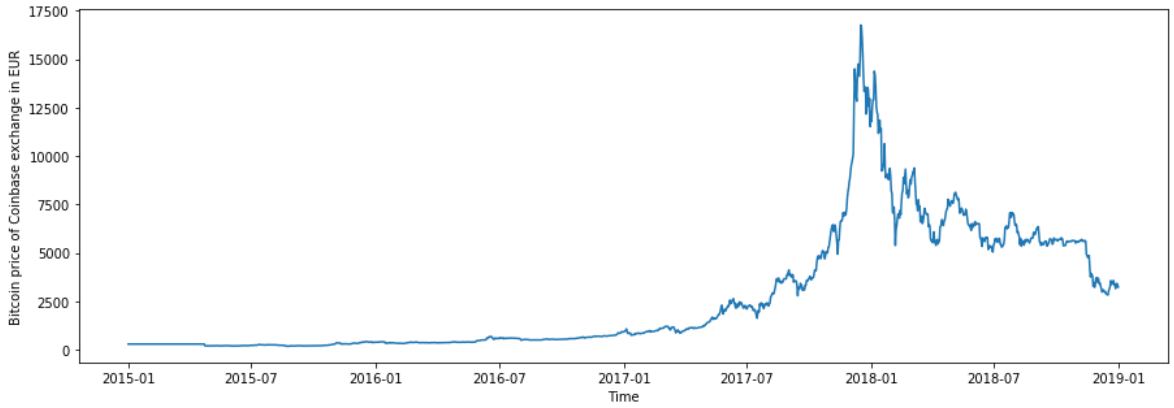


Figure 12: Price variation of the Bitcoin, based on the price of the Coinbase exchange platform.

2.4 Combining all datasets

One of the main advantages of web scrapping the who-trust-whom network directly from the Bitcoin-otc platform was the availability of the Bitcoin addresses. These Bitcoin addresses are unique for all users in the Bitcoin BlockChain. Therefore, these unique addresses are used to combine the who-trust-whom network with the transaction network. However, due to some reason, not all Bitcoin addresses are listed on the Bitcoin-otc website. Still, we managed to extract 2.990 Bitcoin addresses (appr. 50%). This amount was not sufficient enough to link the who-trust-whom network to the transaction data. Other methodologies as graph mappings could be used to link both networks together, which aren't included in this research.

Since we can't link the who-trust-whom network its investigated measure, fairness and goodness, won't contribute to the price prediction of Bitcoin. Therefore, its chapter is discussed in the Appendix. However, since we showed some interesting new methodologies in light of the measure, it is still included in the research and one of the most essential suggestions for further research.

The time-series, representing the Bitcoin price, could easily be combined with the transaction network by the timestamp.

[3] `http://api.bitcoincharts.com/v1/csv/`

3 Betweenness centrality based on absorbing random walks

Betweenness is a centrality measure of a node in a network and can be calculated as the fraction of shortest paths between pairs of nodes that pass through the node of interest. Betweenness can be seen as a measure of the influence a node has over the spread of information through the network.

In 1977 Freeman gave the first formal definition of betweenness centrality, which is based on shortest paths [19]. For every pair of nodes, in a connected graph, there exists at least one shortest path between the nodes such that the sum of the weights of edges is minimized. The betweenness centrality is defined as the number of these shortest paths that pass through the node.

However, by counting only the shortest paths, the traditional definition implicitly assumes that information spreads only along those shortest paths. In 2003, Newman proposed a betweenness measure that relaxes this assumption, including contributions from necessarily all paths between nodes [52]. This measure is based on absorbing random walks, counting how often a node is traversed by a random walk between two other nodes. In this chapter, we argue the use of this improved betweenness centrality, its mathematical properties, and application to the Bitcoin network.

3.1 Economic interpretation

Money laundering is the conversion of criminal income into assets that cannot be traced back to the underlying crime. Yury Fedotov, Executive Director of UNODC (United Nations of Office on Drugs and Crime), stated in 2009 that an estimated amount to US 2.1 trillion or 3.6 percent of GDP is laundered annually [3]. Due to the complex nature of money laundering, it is challenging to derive a single rule set capable of explicitly expressing all of the relationships in the data that correspond to illicit activities. Finding the underlying dynamics of transaction networks can help to identify potential money laundering activities and patterns.

Betweenness centrality can be regarded as a measure of the extent to which a node has control over information flowing between others. In transaction networks, we can argue that the flow of information can be seen as the flow of money. There exist multiple examples of why money flow is essential in tracking fraud and money laundering. First of all, assume that we can find the origin of illegally obtained money. Investigating possible propagation of illegally obtained money over a network can give insights into money laundering activities. It represents a sort of 'follow the money' principle. A secondary argument is the concept of opportunity. Money laundering mostly appears on a large scale, entities where high amounts of money flow through have therefore a higher potential to launder money.

Money laundering. Money laundering is conventionally divided into three phases: *placement* of funds derived from illegal activity, *layering* of those funds by passing them through many institutions and jurisdictions to disguise their origin, and *integration* of the funds into an economy where they appear to be legitimate [58].

The first stage in the progress is placement. The placement stage involves the physical movement of money, derived from illegal activities, into another form that seems less suspicious to law enforcement authorities. The second stage is called layering. The layering stage involves the propagation of illegally money from their illegal source by using complex financial structures. The goal is to hide the audit trail and cover the proceeds of the money. The third stage in the money laundering process is integration. During the integration stage, illegal profits are converted into apparently legitimate business earnings through normal financial or commercial operations.

Choice of betweenness centrality. The choice of betweenness centrality is mostly based on the second stage of money laundering. In a network in which flow is entirely or at least mainly along the shortest paths, the simple betweenness definition of Freeman [19] is sufficient. However, laundered money does not flow only along shortest paths, as criminals try to camouflage the propagation of the illegally obtained money. It makes more sense that the funds wander around more seemingly randomly, encountering multiple nodes. Therefore we would imagine that in the Bitcoin case a realistic betweenness centrality measure should include all paths in addition to the shortest ones. This can be accomplished with the use of absorbing random walks.

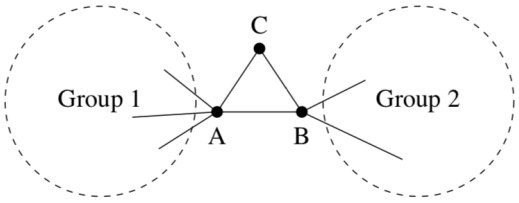


Figure 13: Example network to illustrate a limitation of the betweenness centrality based on shortest paths. Here nodes A and B will have high betweenness centrality, while node C has low betweenness centrality. Source: Newmann, 2001 [52].

Also, a betweenness measure, based on the shortest paths, will cause some limitations. Newmann described a clear example in his paper about betweenness centrality's based on random walks [52]. In Figure 13, a network is illustrated, it exists out of two large groups which are bridged by connections among just a few of their members. nodes A and B will undoubtedly get high betweenness scores since all shortest paths between the two communities must pass through them. On the other hand node, C will get a low score, since none of those shortest paths pass through it, taking the direct route instead from A to B. However, the probability that information flows to C instead of A or B exists, and therefore, should play a significant role in the information flow. "Certainly, it is possible for the information to flow between two individuals via a third mutual acquaintance, even when the two individuals in question are themselves well acquainted" [52].

3.2 Random walks

Random walks are stochastic processes that describes a path that consists of a succession of random steps on some mathematical space. A simple example of a random walk is the random walk on the integer number line, $\mathbb{Z}$, where random paths start at 0, and at each step moves -1 or $+1$ with equal probability.

Mathematically, a random walk on a graph is a special case of a Markov chain, which is a stochastic process $\{X_t\}_{t\geq 0}$ of random variables $X_1, X_2, \ldots$ (states) that satisfies the Markov property. The Markov property states that the probability of moving to the next state depends only on the present state and not on the previous states. The changes in the Markov chain are called transitions, and the probabilities associated with various state changes are called transition probabilities [20]. We will use the definition of a Markov chain to define our random walks.

Definition 3.1. Given a graph $G = (V, E)$ with $n = |V|$. A stochastic process $\{X_t\}_{t\geq 0}$ is called a **random walk** over G if there exists a stochastic matrix $\mathbf{P} \in [0, 1]^{n\times n}$ such that:

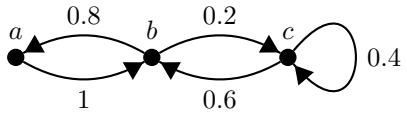
$$X_{t+1} = X_t\mathbf{P}, \forall t \geq 0. \tag{1}$$

(a) The probability that a random path will move to node $j \in V$, given that it is currently at node $i \in V$, is called the **transition probability** from node i to node j. Notation: $\mathbf{P}(i, j) \in [0, 1]$.

(b) The array $\mathbf{P}(i) = \{\mathbf{P}(i,j)\} \in [0,1]^n$ displays all transition probabilities leaving node i and is called the **transition array** for node i.

(c) The matrix $\mathbf{P} = \{\mathbf{P}(i,j)\} \in [0,1]^{n \times n}$ displays the transition probabilities on the whole graph G and is called the **transition matrix**.

A random walk is characterized by a state space $\{X_t\}_{t \geq 0}$, a transition matrix $\mathbf{P}$ containing all transition probabilities, and an initial state X_0. To clarify the definition of a random walk and its state space $\{X_t\}_{t \geq 0}$, we introduce the following directed graph with weighted edges.



Corresponding transition matrix $\mathbf{P} = \begin{bmatrix} 0 & 1 & 0 \\ 0.8 & 0 & 0.2 \\ 0 & 0.6 & 0.4 \end{bmatrix}$

The graph consists of three nodes; a, b and c and some edges, which are the transition probabilities. The state of the random walk at time t is written as a stochastic row vector X_t, which holds equation (1), and represents the distribution of the random walk over nodes a, b and c at time t. The first input of the row vector X_t denotes the probability that a random path is in node a at time t, and the second and third input the probability of a random path being in nodes b and c at time t respectively. Each state X_t is a sample space and therefore the sum of its vector elements is equal to one.

A random walk should always have an initial state X_0, from which the future states can be calculated. Assume the initial space $X_0 = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix}$, which means that the random walk is with probability one in node A. For our random walk $\{X_t\}_{t \geq 0}$ this will mean:

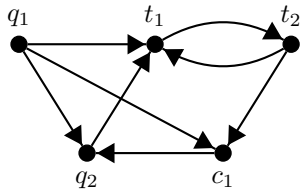
$$
\begin{aligned}
X_0 &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \\
X_1 = X_0\mathbf{P} &= \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 0.8 & 0 & 0.2 \\ 0 & 0.6 & 0.4 \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \\
X_2 = X_1\mathbf{P} &= \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 & 0 \\ 0.8 & 0 & 0.2 \\ 0 & 0.6 & 0.4 \end{bmatrix} = \begin{bmatrix} 0.8 & 0 & 0.2 \end{bmatrix} \\
&\vdots \\
X_t &= X_0(\mathbf{P})^t
\end{aligned}
$$

Note that the entries of X_t sum up to one for every time step. The two terms 'random walk' and 'random path' are used. The main difference is that a random walk is the set of possible outcomes of the random variable X_t at time t. Whereas a random path is a observed path over the nodes, in the above example we could say that $\{S_t\}_{t \geq 0} = \{a, b, c, b, c, c, \dots\}$ is a random path. Random paths are not always unique, while random walks are unique for a fixed set of transition matrix $\mathbf{P}$ and initial state X_0.

3.3 Absorbing random walks

An absorbing random walk, in short ARW, is a random walk with a fixed type of transition matrix. To establish the transition probabilities for an absorbing random walk, we need to look at its fundamental behavior. First, the clear definition of an absorbing random path is given. Second, we discuss the three types of probability distributions which form the basis of the overall transition probabilities for absorbing random walks. Third, we define these transition probabilities in formulas. And last, we derive the expected length of an absorbing random path.

Absorbing random paths. Given a graph $G = (V, E)$, with a subset of query nodes $Q \subseteq V$ and central nodes $C \subseteq V$. A general random walk describes a path over the nodes of V, where the path can be seen as a series $\{S_t\}_{t\geq 0}$ such that S_{t+1} is a node chosen under the transition probabilities of S_t. An absorbing random path on the nodes of V proceeds by the following discrete steps; starting at a query node $q \in Q$, at each step moving to a different node following the edges in G, until it arrives at some node in C, where it terminates.



Random path	Absorbing?
$\{q_1, t_1, t_2, c_1\}$	yes
$\{t_1, t_2, c_1, q_2\}$	no
$\{q_2, t_1, q_1, q_2\}$	no
$\{q_2, t_1, t_2, c_1\}$	yes

Figure 14: An absorbing random path is always a random path; however, not all random paths are absorbing random paths. The difference is illustrated in the above table for the illustrated graph. We assumed that $Q = \{q_1, q_2\}$ is the set of query nodes and $C = \{c_1\}$ the set of central nodes.

The overall transition probabilities, for an absorbing random walk, are based on three kinds of distributions; a starting distribution, a restart probability, and neighbor-propagation distributions.

Starting distribution. The starting distribution $\mathbf{s}$ corresponds to the choice of query nodes at the beginning of a absorbing random path or after restart. Due to the fact that (re)starts can only occur in query nodes, we can state that the starting distribution is zero for nodes outside of Q.

Definition 3.2. Given a graph $G = (V, E)$ and a set of query nodes $Q \subseteq V$. The probability distribution $\mathbf{s}$ is called the **starting distribution**, such that:

$$\mathbf{s}(v) = \begin{cases} p_v \in [0,1] & \text{if } v \in Q \\ 0 & \text{if } v \notin Q \end{cases} \quad \text{and} \sum_{v \in Q} p_v = 1, \forall v \in V$$

Usually, there is no reason to distinguish between query nodes, a uniform starting distribution for the starting probability is then most suiting, i.e., $\mathbf{s}(q) = 1/|Q|$ if $q \in Q$ and zero otherwise. While in some cases the starting distribution can be used to mark the importance of specific query nodes.

A clear example, where the choice of a non-uniform distribution is most suiting, is tracking cash deposits. Say, node A makes a cash deposit of 50 euros and node B of 950 euros, intuitively the cash propagation of node B is more interesting. We could choose $s(A) = \frac{1}{20}$ and $s(B) = \frac{19}{20}$ as the starting probability to trace the propagation of all cash over the transaction network, where node B is of higher importance. In general, the choice of the starting distribution should always correspond to the original purpose of the betweenness centrality measure.

Restart probability. For absorbing random walks, a restart probability α is introduced. When a path is 'restarted', it proceeds to a query node under the starting probability. This transforms our random walker into some sort of random jumper over the network; this is in the interest of generality and

allows for adjustable importance of query nodes in the measure. When restarting, the path proceeds to a query node selected randomly according to the starting distribution.

It is crucial to observe that the betweenness centrality of each node may in general change as the restart probability changes [56]. A higher restart probability decreases the likelihood of following long paths of links without taking a random jump, and thus increase the contribution to a node's betweenness centrality [6]. Generally, a restart probability $\alpha > 0$ is chosen, since this ensures invertibility of the transition matrix.

Neighbor-propagation distributions. An absorbing random path moves at each step to a different node following the edges in G, before terminating as it reaches a central node. The up-following node of the random path over the edges is chosen according to a neighbor-propagation distribution $\mathbf{n}_v$. This distribution is characterized by the (outgoing) edges of its node.

Definition 3.3. Given a graph $G = (V, E)$. The probability distribution $\mathbf{n}_v$ is called the **neighbor-propagation distribution** for node $v \in V$, such that $\forall i \in V$:

$$\mathbf{n}_v(i) = \begin{cases} p_i \in [0,1] & \text{if } i \in N(v) \\ 0 & \text{if } i \notin N(v) \end{cases} \quad \text{and} \quad \sum_{i \in N(v)} p_i = 1,$$

where $N(v)$ is the set of neighbor nodes of v.

Note that if a node doesn't have any outgoing edges, the absorbing random path restarts under the starting probability. This assumption is made to prevent an absorbing random path to be stuck in a sink node for a long time.

Transition probabilities in formulas. We distinguish two-state scenarios for an absorbing random path; the path is either absorbed or not. First, assume that the random path is absorbed, which happens if the random path is situated in any node $c \in C$. By definition of an absorbing path, the walk cannot leave this node. Therefore, $\forall c \in C$, $\mathbf{P}(c, c) = 1$ and $\mathbf{P}(c, j) = 0$, if $j \neq c$. Second, consider that the random path is not absorbed yet. The walk either moves to a neighbor node under the neighbor-propagation distribution $\mathbf{n}_v$ or restarts, with probability α, to a query node $q \in Q$ under the starting distribution $\mathbf{s}$.

Definition 3.4. Given a graph $G = (V, E)$, a set of query nodes $Q \subseteq V$, a set of central nodes $C \subseteq V$, and an absorbing random walk on graph G with respect to (Q, C). For the absorbing random walk, a starting distribution $\mathbf{s}$, neighbor-propagation distributions $\mathbf{n}_v$ $\forall v \in V$, and a restart probability $\alpha \in \mathbb{R}$ are given.

1. The set $T \subseteq V$ is called the set of **transient nodes** of an absorbing random walk, such that $T = V \backslash C$.

2. The probability that the random walk will move to node $j \in V$, given that it is currently at node $i \in V$, is called the **ARW transition probability** from node i to node j and notated by $\mathbf{P}(i, j) \in \mathbb{R}$, such that $\forall i, j \in V$:

$$\mathbf{P}(i,j) = \begin{cases} 0 & \text{if } i \in C \text{ and } j \neq i \\ 1 & \text{if } i \in C \text{ and } j = i \\ (1-\alpha)\mathbf{n}_i(j) + \alpha \mathbf{s}(j) & \text{if } i \in V \backslash C = T \end{cases} \tag{2}$$

3. The matrix $\mathbf{P} = \{\mathbf{P}(i, j)\} \in \mathbb{R}^{n \times n}$ displays the ARW transition probabilities on the whole graph G and is called the **ARW transition matrix** for a random walk, such that:

$$\mathbf{P} = \begin{pmatrix} \mathbf{P}_{TT} & \mathbf{P}_{TC} \\ \mathbf{0} & \mathbf{I} \end{pmatrix},$$

where $\mathbf{P}_{TT}$ is the $|T| \times |T|$ sub-matrix of $\mathbf{P}$ that contains the transition probabilities between transient nodes; $\mathbf{P}_{TC}$ is the $|T| \times |C|$ sub-matrix of $\mathbf{P}$ that contains the transition probabilities from transient to absorbing nodes.

Expected length of an absorbing random path. The absorbing random walk is similar to an absorbing Markov chain with $|T|$ transient states, $|C|$ absorbing states, and a transition matrix $\mathbf{P}$. A basic property of a Markov chain is the expected number of visits to a transient state j, starting from a transient state i. The probability of transitioning from i to j in exactly t steps is the (i, j)-entry of $\mathbf{P}_{TT}^t$. Summing this for all possible values of t (from 0 to ∞, where the sum is finite due to the presence of absorbing states), gives the expected number of times the Markov chain is in state j, given that the chain started in chain i [59]. For absorbing random walks, this can be translated in the sum being the expected length of the walk in node j, given that the walk started in node i. The collection of these lengths is called the fundamental matrix of the absorbing random walk.

Definition 3.5. Given a graph $G = (V, E)$, a set of query nodes $Q \subseteq V$, a set of transient nodes $T \subset V$, an absorbing random walk on graph G with respect to (Q, T), and its ARW transition matrix $\mathbf{P}$. The matrix $\mathbf{F} \in \mathbb{R}^{|T| \times |T|}$ is called the **fundamental matrix** of the absorbing random walk, such that;

$$\mathbf{F} = \sum_{t=0}^{\infty} \mathbf{P}_{TT}^t = \left(I_{|T|} - \mathbf{P}_{TT}\right)^{-1}.$$

Each entry of the fundamental matrix represents the expected length of the walk in node j, given that the walk started in node i. Summing these expected lengths for all possible values of j, provides the overall expected length of an absorbing random walk that started in node i, before it gets absorbed. Due to the possibility that the absorbing random walk starts at an absorbing node (and being absorbed immediately), the expected length of a walk starting in an absorbing node is equal to zero.

Definition 3.6. Given a graph $G = (V, E)$, a set of query nodes $Q \subseteq V$, a set of central nodes $C \subseteq V$, an absorbing random walk on graph G with respect to (Q, C), and its fundamental matrix $\mathbf{F}$. The vector $\mathbf{L} \in \mathbb{R}^n$ is called the **expected length of a absorbing random walk**, such that;

$$\mathbf{L} = \begin{pmatrix} \mathbf{F} \\ \mathbf{0} \end{pmatrix} \mathbb{1}^{|T|},$$

where $T \subset V$ are the transient nodes of the absorbing random walk.

3.4 The 1-ARW-betweenness centrality measure

Different choices for the neighbor-propagation distributions, restart probability, and starting distribution influence the length of an absorbing random walk. This length is the critical factor that determines the ARW betweenness centrality. The choice of the distributions depends on the goal of the measure and the structure of the transaction network.

Consider one more time the example of tracking cash deposits. We want to investigate the cash propagation of these deposits over a transaction network. As query nodes, the origins of the cash deposits would be a logical choice, whereas its importance (amount) the starting probability.

In this research, we will focus on a simple case of the ARW-betweenness centrality, which we call the 1-ARW-betweenness centrality. This measure uses a single central node, for which the expected length of the absorbing random walk is calculated. The central node is equal to the node for which the betweenness centrality is calculated. The 1-ARW-betweenness centrality can be regarded as a measure of how fast all possible money flows towards this single central node inside the network. In definition 3.7 the assumptions towards the absorbing random walks, on which the 1-ARW-betweenness centrality is based, are given.

Definition 3.7. (1-ARW-betweenness centrality assumptions) Given a graph $G = (V, E)$ with $n = |V|$, a restart probability α and a single central node $c \in V$. For the absorbing random walk, on which the 1-ARW-betweenness centrality is based, we assume:

1. All nodes, except the central node, are query nodes, i.e. $Q = V \backslash \{c\}$.
2. The neighbor-probability distribution, from node i to node j ($\forall i, j \in V$), is notated by $\mathbf{n}_i(j)$ and defined as:

$$\mathbf{n}_i(j) = \begin{cases} A(i,j)/A_i^{sum} & \text{if } A_i^{sum} > 0 \\ \frac{1}{n} & \text{if } A_i^{sum} = 0 \end{cases} \quad \text{with } A_i^{sum} = \sum_{j=1 \in n} A(i,j).$$

   A is the adjancency matrix of graph G.
3. A uniform starting distribution $\mathbf{s}(i) = \frac{1}{n}$, $\forall i \in V$.
4. The ARW transition probabilities and ARW transition matrix are constructed by Definition 3.4.

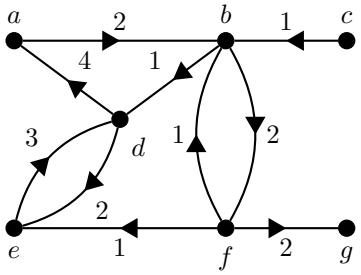
The expected length of an absorbing random walk is a vector $\mathbf{L} \in \mathbb{R}^n$ where each input denotes the expected length of the absorbing random walk, starting in that specific node (Definition 3.6). Since all nodes are query nodes, taking the average of all inputs of $\mathbf{L}$ will result in an expected length of the absorbing random walk, not knowing in which node the random walk started.

Definition 3.8. Given a graph $G = (V, E)$ with $n = |V|$, and a restart probability α. The **1-ARW-betweenness centrality** L_{arw} of a node $u \in V$ is defined as:

$$L_{arw}(u) = \frac{1}{n} \sum_{i=1}^{n} \mathbf{L}(i),$$

where $\mathbf{L} = \{\mathbf{L}(i)\} \in \mathbb{R}^n$ is the expected length of an absorbing random walk with central node u under the 1-ARW-betweenness centrality assumptions (Definition 3.7).

Note that the measure may be normalized by dividing through the number of pairs of nodes not including u, which for directed graphs is $(n-1)(n-2)$.



n	$L_{arw}(n)$
A	0,09
B	0,06
C	0,24
D	0,08
E	0,12
F	0,12
G	0,27

$$\mathbf{N} = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{1}{3} & 0 & \frac{2}{3} & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \frac{2}{3} & 0 & 0 & 0 & \frac{1}{3} & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & \frac{1}{4} & 0 & 0 & \frac{1}{4} & 0 & \frac{1}{2} \\ \frac{1}{7} & \frac{1}{7} & \frac{1}{7} & \frac{1}{7} & \frac{1}{7} & \frac{1}{7} & \frac{1}{7} \end{bmatrix}$$

Figure 15: An example network to explain 1-ARW-betweenness centrality. Left: A visualization of the transaction network. Middle: The corresponding 1-ARW-betweenness centrality scores. To calculate the scores, the 1-ARW algorithm (Algorithm 1) is used, which will be explained further on. Right: The matrix representation of the ARW neighbor-probability distributions.

The graph in Figure 15 illustrates a small weighted transaction network. The weights of the network correspond to the transaction value between two nodes, which are expressed in the adjacency matrix. The matrix on the right is a matrix representation $\mathbf{N}$ of the neighbor-probability distributions where

the i^{th} input represents the node of the i^{th} letter in the alphabet. Node A (row 1) has one outgoing edge such that the ARW moves to node B under probability one. Node B (row 2) has two outgoing edges, to both D and G. Since the weights are expressed in the adjacency matrix, the ARW moves from node B to D under probability $\frac{1}{3}$ and to G with a probability of $\frac{2}{3}$. Since node G (row 7) has no outgoing edges, the ARW restarts by assumption under a uniform starting probability.

This example reflects very well the behavior and construction of the 1-ARW-betweenness centrality measure. nodes A, B and D are most likely to have high cash flow and therefore low betweenness centrality. In the table of Figure 15, all 1-ARW-betweenness centrality scores are listed.

3.5 Algorithm

In the upcoming section, the algorithm to compute the 1-ARW-betweenness centrality for each node in the transaction network is presented. First, we present the algorithm and show its exponential time complexity. Second, we argument the choice of the restart probability α.

Algorithm 1 1-ARW-betweenness centrality algorithm

Require: A graph $G = (V, E)$, and a restart probability α.
Ensure: The 1-ARW-betweenness centrality score for all nodes in V

1:
2: **for** $\forall u \in V$ **do**
3: Define $C = \{u\}$ as the set of central nodes
4: Define $Q = V \backslash \{u\}$ as the set of query nodes
5: Let $\mathbf{P}_{TT}$ be the ARW transition matrix of the transient nodes $T = V \backslash C$ under the 1-ARW-betweenness centrality assumptions (Definition 3.7).
6: Calculate the fundamental matrix $\mathbf{F} = (I - \mathbf{P}_{TT})^{-1}$
7: Calculate the expected length of the ARW $\mathbf{L} = \begin{pmatrix} \mathbf{F} \\ \mathbf{0} \end{pmatrix} \mathbb{1}^{|T|}$
8: Calculate the 1-ARW-betweenness centrality $L_{arw}(u) = \frac{1}{|V|} \sum_{i=1}^{|V|} \mathbf{L}(i)$
9:
10: **return** The 1-ARW-betweenness centrality score $L_{arw}(u)$, $\forall u \in V$

Outline Algorithm. Algorithm 1 describes the 1-ARW-betweenness centrality algorithm. The algorithm requires a graph G and a restart probability α. In subsection 3.5.2 the choice of the restart probability is argued, and in Appendix B.3 the Python code can be found.

The algorithm is a for-loop over all nodes $v \in V$, where for each node individually the 1-ARW-betweenness centrality score is calculated. Inside the for-loop, the algorithm initially starts by choosing the central nodes as the to-calculated node v, and the query nodes as all remaining nodes (line 5 & 6). For these sets of central and query nodes, the ARW transition matrix is determined under the 1-ARW-betweenness centrality assumptions (Definition 3.7, line 5). From the transition matrix, the fundamental matrix and the expected length of the absorbing random walk are calculated. At last, in line 8 the 1-ARW-betweenness centrality is calculated for node v.

3.5.1 Time complexity

Time complexity quantifies the amount of time taken by an algorithm to run as a function of the length of the input [61]. Time complexity is important since it can make a big difference as to whether the 1-ARW-betweenness centrality is practical for large graphs. In line 6 of Algorithm 1 a matrix inverse to determine the fundamental matrix of the absorbing random walk. With use of the Gauss-Jordan elimination, the time complexity of matrix inversion is equal to $\mathcal{O}(n^3)$, with n the number of nodes in

a graph [15]. The notation $\mathcal{O}(n^3)$ is the upper limit of the computation time growth [50], such that the algorithm performs to the third power of input value n (worst case).

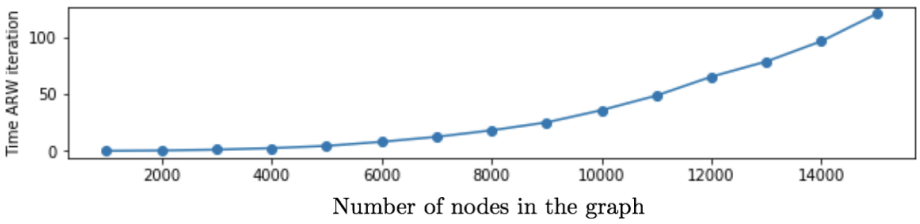


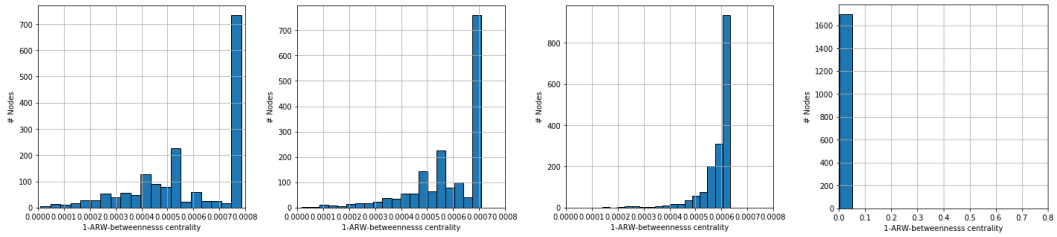
Figure 16: Time complexity of calculating the 1-ARW-betweenness centrality for a single node. As the number of nodes in a graph increases, the time grows exponentially.

In Figure 16, the time complexity of calculating the 1-ARW-betweenness centrality of a single node is visualized. The x-axis denotes the number of nodes in a graph. As the number of nodes increases, the time grows exponentially, as suspected. This causes the 1-ARW-betweenness centrality being impractical while dealing with large graphs. While using such a measure in problems as price prediction, the obstacle can be overcome by sampling the original graph in smaller subsets. This will be discussed and used in section 4.4.5.

3.5.2 Restart probability

Besides a graph, the 1-ARW-betweenness centrality algorithm requires a restart probability α. There is expected that the betweenness centrality of each node may in general change as the restart probability changes [56]. A higher restart probability decreases the likelihood of following long paths of links without taking a random jump, and increases the contribution to a node's betweenness centrality [6].

To test the influence of the restart probability a graph is constructed based on the Bitcoin MIT dataset. Since its time complexity depends on the number of nodes in the graph, a small subset of the Bitcoin MIT dataset is used. In total 2.517 transactions were observed between 26 April 2016 and 28 April 2016, where only the biggest graph component is considered. In the graph a total of 1.634 unique Bitcoin addresses were listed.



(a) $\alpha = 0.1$. (b) $\alpha = 0.5$. (c) $\alpha = 0.85$. (d) $\alpha = 1$.

Figure 17: The distribution of the 1-ARW-betweenness centrality scores for alternating values of α.

In Figure 17, the distribution of the 1-ARW-betweenness centrality scores for alternating values of α. One of the most interesting figures is Figure 17d, here all nodes have equal centrality. This is caused by the fact that the absorbing random walks restarts in every step, such that the absorbing random walk is distributed equally over all nodes in the network. In Figure 17a there exists more nodes with a higher centrality, this is caused by the idea that the random walker gets 'stuck' in a node

without outgoing edges. The choice of the restart probability is mainly based in which ratio we want the random walker to follow long paths without restarting. From Literature a most common chosen restart probability was equal to $\alpha = 0.85$, which is visualized in Figure 17c [6].

3.6 Implementation on a subset of the Bitcoin MIT dataset

The Bitcoin MIT dataset represents a Bitcoin transaction network. The source contained all Bitcoin transactions in the first 508241 blocks (approximately up to 9 Feb 2018). The Bitcoin MIT dataset provides 5 entries per line; transactionID, in-address, out-address, amount and timestamp. The *transactionID* is the unique index of a transactions, *in-address* notates the node id of the sending Bitcoin address, *out-address* the node id of the receiving Bitcoin address, *amount* the amount of Bitcoin transferred in Satoshi, and *timestamp* the time of the transaction, measured as seconds since Epoch.

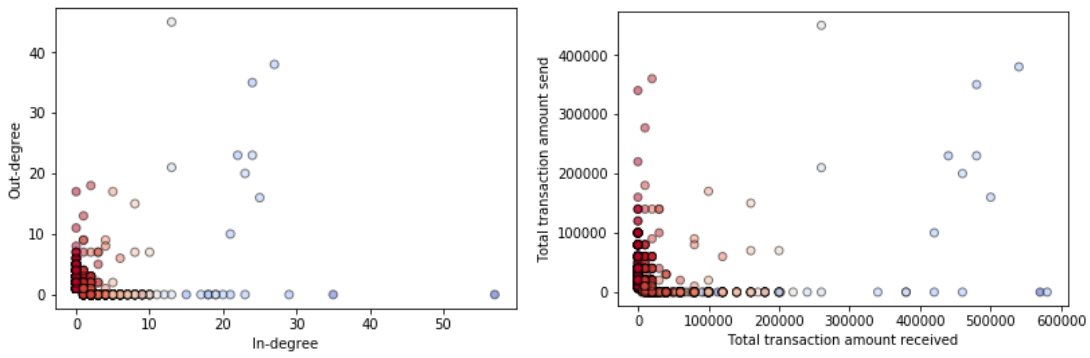
We can construct a weighted directed graph $G_{mit} = (V, E, w)$ by taking all in/out-addresses as nodes V and the made transactions as the edges E. The direction of the edges is from in-address to out-address, and their amounts are used as weights w. The graph is constructed for a small subset of the Bitcoin MIT dataset, to reduce the computation time of the 1-ARW-betweenness centrality. Only data between 26 April 2016 and 28 April 2016 is used, where only the biggest graph component is considered. The constructed graph contained 1634 nodes and 2517 transactions. For more detailed information on the Bitcoin MIT dataset, we refer to subsection 2.2.

In Table 4 some statistics of G_{mit} are visualized.

Description		**Degree**			
Number of unique Bitcoin addresses	1.634	Average in/out degree	1,54		
Number of transactions	2.517	Max in-degree	0	Max out-degree	45
		Min in-degree	0	Min out-degree	51

Table 4: Statistics of G_{mit}.

In Figure 18, the in/out- degree, and the total received/send transaction amounts in Satoshi for each node are visualized. The coloring in the figures is the 1-ARW-betweenness centrality, where the blue indicates high centrality and red indicates low centrality. Interesting is to see that both the degrees as the transaction amounts are somehow correlated with the 1-ARW-betweenness centrality.

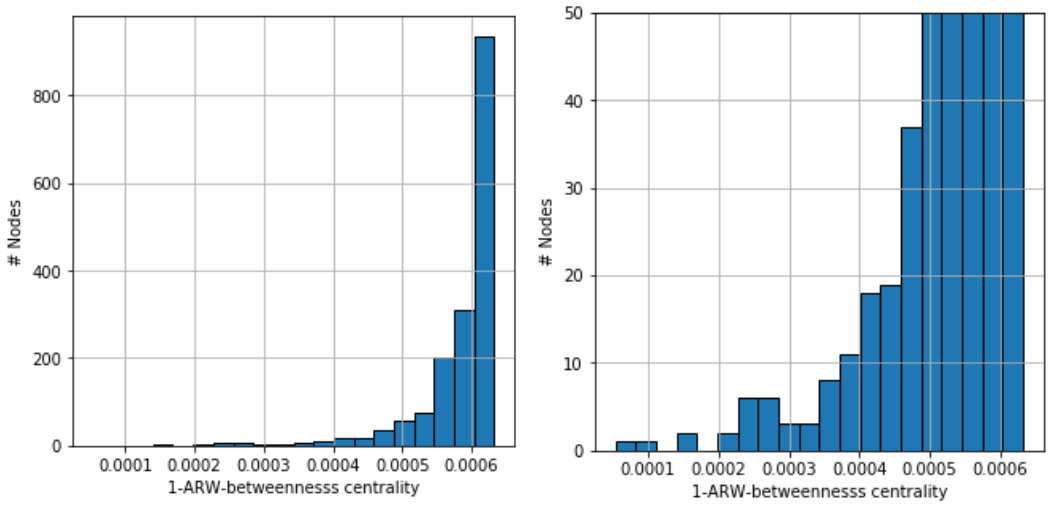


(a) In/out- degree.

(b) Received/send transaction amount.

Figure 18: A 1-ARW-betweenness centrality coloring on the in/out- degree and the total received/send transaction amounts in Satoshi. The blue indicates high centrality and red indicates low centrality.

In Figure 19 the final 1-ARW-betweenness centrality scores are visualized.



(a) Normal axis.

(b) Zoomed-in y-axis.

Figure 19: 1-ARW-betweenness centrality on the Bitcoin MIT dataset

3.7 Summary & Conclusion

General node betweenness centrality is calculated as the fraction of shortest paths between node pairs that pass through the node of interest. However, by counting only the shortest paths, the definition implicitly assumes that information spreads only along those shortest paths. Laundered money doesn't always flow along the shortest paths, as criminals try to camouflage the propagation of the illegally obtained money. It makes more sense that the money wanders around more randomly, encountering multiple nodes. Therefore we would imagine that in the Bitcoin case a realistic betweenness centrality measure should include all paths in addition to the shortest ones. This can be accomplished with the use of absorbing random walks.

An absorbing random walk is a special type of random walk. A general random walk describes a path over the network nodes, where the path can be seen as a series such that the successive node is chosen under the transition probabilities of the random walk. An absorbing random path over the network nodes proceeds by the following discrete steps; starting at a query node, at each step moving to a different node following the edges of the network, until it arrives at some central node, where it terminates.

The on absorbing random walks based measure; 1-ARW-betweenness centrality, is introduced. Its calculation is mostly based on the expected length of the absorbing walk for a single central node. The 1-ARW-betweenness centrality can be regarded as a measure of how fast all possible money flows towards this single central node inside the network.

The time complexity for the 1-ARW-betweenness centrality is of importance since it can make a big difference as to whether the measure is practical for large graphs. It is shown that the computation time grows exponentially, with respect to the number of nodes in a network.

4 Predicting the direction of Bitcoin prices

Predicting the trends in Bitcoin market prices is a very challenging task due to the many uncertainties and variables influencing the market value. Because of this, the market is susceptible to quick changes, causing seemingly random fluctuations in the Bitcoin price. Due to the chaotic and highly volatile nature of Bitcoin behavior, investments come with high risk. In order to minimize the risk involved, knowledge of Bitcoin price movement in the future is desirable [39].

4.1 Related work

In literature the application of different machine learning algorithms, on Bitcoin prices, is already researched with the use of Random Forests [48], Bayesian neural networks [34], long short-term memory neural networks [49], support vector machine [22], and other algorithms [23][26]. These studies were able to anticipate, to different degrees, the price fluctuations of Bitcoin.

Most recent and significant work include the paper of Greaves et al. [22] and Zhao et al. [48], both in 2015. Graeves et al. investigated the predictive power of 'BlockChain network based' features on the future price of Bitcoin. As a result of feature engineering and machine learning algorithms, as support vector machines and artificial neural networks, they obtained an sign (up-down) Bitcoin price movement classification accuracy of roughly 55 percent. They concluded that limited predictability was seen in BlockChain data alone, as the price is technically dictated by exchanges whose behavior lies outside of the realm of the BlockChain. Another significant research from Zhao et al. (2015) focused on identifying daily trends in the Bitcoin market [48]. They used 25 features and a custom algorithm, that leverages both Random Forests (RF) and generalized linear models (GLM). Their results had up to 56 percent accuracy in predicting the sign (up-down movement) of future price changes using 10 minutes time intervals. They observed that RF gave a better accuracy rate as compared to GLM. This is because RFs use non-parametric Decision Trees, so outliers and non-linear separability of the data are less impactfull.

Most current work, however, does not explore or disclose the relationship between the Bitcoin price and other features outside the transaction network, such as market capitalization, Bitcoin mining speed or entity behavior. Also, most of the features are extracted from the network level, that means extracting the number of transactions, users, Bitcoins mined, etc. These features don't capture complex relationships between the nodes in the Bitcoin transaction network. We focus on exploring additional features, such as features outside the transaction network and node-based features inside the transaction network, to improve the quality of the prediction.

4.2 Classification problem

Traders are more likely to buy Bitcoins whose value is expected to increase in the future. While on the other hand, traders are more likely to refrain from buying Bitcoins, whose value is expected to fall in the future. Therefore knowing the direction of the Bitcoin price (increase/decrease) is sufficient for a profitable strategy in trading Bitcoins [72].

We distinguish two classes for the Bitcoin price movement:

0 The Bitcoin price descends or stays equal.
1 The Bitcoin price rises.

By using the above classes, the prediction of the Bitcoin price movement can be seen as a classification problem. Classification refers to the problem of identifying to which set of classes (sub-populations) a new observation belongs [55]. The classification model is based on historical data containing observations whose class membership (0 or 1) is known. For the purpose of the research, we will aim to predict the direction of Bitcoin prices on a daily basis.

A daily classification is chosen due to the nature of additional features used in the classification model (subsection 4.4.4). One of the most well-known learning methods for classification is the Random Forest method, which is based on a assemble of Decision Trees. A common problem, in machine learning on networks, is that some features become impractical for large graphs. Therefore, the classification model will rely on graph sampling to reduce computation time (subsection 4.4.5)[43]. Since graph sampling can be seen as a 'pre-sampling' stage of a Random Forest, the choice of the Random Forest learning method is substantiated (subsection 4.5).

4.3 Random Forest

In 2001, Breimann proposed an ensemble learning method for classification called Random Forest (RF) [5]. Random Forest appears as an assemble of Decision Trees, where a simple majority vote over their outcomes determines the output of the Random Forest. Each tree is constructed using a different subset of observations and a different subset of the features. Decision Trees are popular, since they enable predictive models to have high accuracy, stability and ease of interpretation. However, a single Decision Tree tends to overfit the training data. Random Forests are a way of averaging multiple Decision Trees, trained on different parts of the training set, with the goal of reducing variance [25].

4.3.1 Decision Tree

A Decision Tree (DT) is a flowchart-like structure made of three elements; internal nodes, branches and leafs. Each internal node represents a splitting rule on the feature set, each branch the outcome of a splitting rule, and each leaf a classification label. The paths from root to leaf are called the classification rules of the Decision Tree [37]. The classification rules of the tree divides the feature space (set of possible feature values) into multiple regions, called the region partitions [32]. Since we use a classification model for predicting the direction of Bitcoin prices, we only consider classification Decision Trees.

Introduction to Decision Trees. In Figure 20 an example of a Decision Tree is visualized. The Decision Tree consist of a series of splitting rules, starting at the root node of the tree. An example of such a series is the following: The root node assigns a observation, not meeting the splitting rule `Number of transactions > 1.000`, over the left branch, where it reaches a new node. This node gives the observation classification 1, if the splitting rule `Mean transaction value > 50.000` is met. The outcome of these series will be at the leftmost leaf in Figure 20.

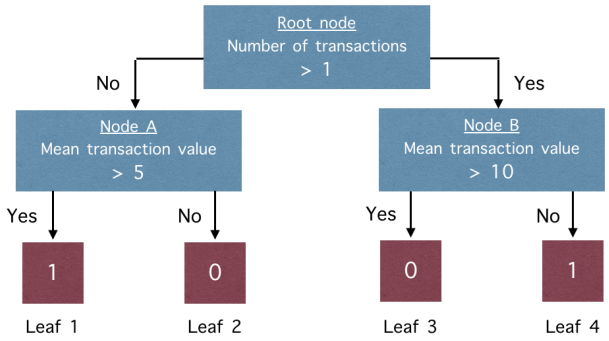


Figure 20: Example of a Decision Tree with classification labels; 1 and 0. The blocks (blue) represents the splitting rules on certain features, determining which branch (yes or no) will be followed. The example splitting rules are simple yes-no operations, they could also be more complex statements.

Definition 4.1. The space $\mathcal{Z} = (z_1, z_2, \ldots, z_M) \subseteq \mathbb{R}^M$ is called the **feature space** of a Decision Tree with $M \in \mathbb{N}$ features, such that the random variable z_i represent the possible outcomes of the i^{th} feature, $\forall i = 1, \ldots, M$.

Definition 4.2. Let $\mathcal{Z}$ be the feature space of a Decision Tree, the set of spaces $\mathcal{R} = \{r_1, r_2, \ldots, r_J\}_{J \in \mathbb{N}}$, is called a **partition** of the feature space, if $r_1, r_2, \ldots, r_J$ are distinct and non-overlapping spaces, such that:

$$\mathcal{Z} = r_1 \cup r_2 \cup \cdots \cup r_J \text{ and } r_i \cap r_j = \emptyset, \forall i, j \in \{1, 2, \ldots, J\} \text{ with } i \neq j.$$

A single space $r_i \subseteq \mathcal{Z}, i \in \{1, 2, \ldots, J\}$ is called a **region** for the partition, and the function $f_{\mathcal{R}} : \mathcal{Z} \to \mathcal{R}$ is called the **splitting rule** of the partition.

The example Decision Tree considers 3 splitting rules, and therefore also 3 partitions. Define the initial feature space as $\mathcal{Z} = [0, 5] \times [0, 20] \subseteq \mathbb{R}^2$, with the interval $[0, 5]$ representing the possible outcomes of the number of transactions and the interval $[0, 20]$ the possible outcomes of the mean transaction value. At the root node we make the first partition $\{r_A, r_B\}$ for feature space $\mathcal{Z}$, such that $r_A = [0, 1] \times [0, 20]$ and $r_B = [1, 4] \times [0, 20]$. At node A the next partition $\{r_1, r_2\}$ for feature space r_A is made, such that $r_1 = [0, 1] \times [5, 20]$ and $r_2 = [0, 1] \times [0, 5]$. A similar partition $\{r_3, r_4\}$ is made for node B. From the partitions of node A and B we will end up with a total of 4 regions, $\{r_1, \ldots r_4\}$, representing a overall partition over the initial feature space $\mathcal{Z}$ (Figure 21).

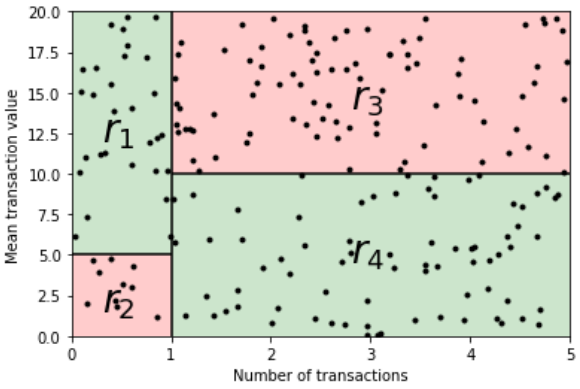


Figure 21: The partition $\mathcal{R} = \{r_1, \ldots, r_4\}$ of feature space $\mathcal{Z} = [0, 5] \times [0, 20] \subseteq \mathbb{R}^2$ according to the Decision Tree in Figure 20. The green color in a region represents classification value 1 and red represents classification value 0.

Growing a Decision Tree. Growing a classification tree is a recursive process of making partitions of the regions of a previous partition, starting with a initial feature space. The partitions are made according to a splitting rule, which optimizes a error function over the classification values of the training set. Naturally we like to assign training observations with similar classification to the same region. A typically used error to measure the quality of a region is called the Gini index [32].

Definition 4.3. Let $\mathcal{Z}$ be a feature space. The set of two paired variables;

$$(X, Y) = \{(\mathbf{x}_i, y_i)\}_{i \leq N},$$

is called a **training, validation or testing set** with $N \in \mathbb{N}$ observations and $K \in \mathbb{N}$ classes. We call $X \in \mathbb{R}^{N \times M}$ the collection of feature observations, with N rows of **feature observations** $\mathbf{x}_i \in \mathcal{Z}$, $\forall i \leq N$. And $Y \in \{0, 1, \ldots, K-1\}^N$ the collection of classifications, with **classifications** $y_i \in \{0, 1, \ldots, K-1\}$, $\forall i \leq N$.

Definition 4.4. Consider a training set (X, Y) of feature space $\mathcal{Z}$, with N observations and K classes. Let $r \subseteq \mathcal{Z}$ be a region of the feature space, the variable $G_r \in [0, 1]$ is called the **Gini index** of region r, such that:

$$G_r = \sum_{k=1}^{K} p_{rk}(1 - p_{rk}),$$

and,

$$p_{rk} = \frac{1}{N} \left| \{y_i \in Y : y_i = k\} \right|, \forall k = \{0, 1, \ldots, K-1\}$$

The variable $p_{rk} \in [0, 1]$ represents the **proportion of training observations** having the k^{th} class for region r, such that $\sum_{k=1}^{K} p_{rk} = 1$.

The Gini index measures the total variance across K possible classes, where it is trivial that the Gini index is small if all p_{rk}'s are close to zero or one. By minimizing the sum of the Gini indexes over all regions in a partition, we can find the optimal splitting rule. Other well-known errors measures are the classification error and the cross-entropy [32]. However, we use the Gini index since this approach favors large partitions over a small subset of features, which is most optimal when dealing with a small number of classes [9]. In Algorithm 2, the algorithm for training a Decision Tree is given.

Algorithm 2 Training a Decision Tree, with use of the Gini index.

Require: The number of regions J in a partition and a training set (X, Y) of feature space $\mathcal{Z}$, with N observations and K classes.
Ensure: A Decision Tree classifier $f : \mathcal{Z} \to \{0, 1, \ldots, K-1\}$.

Define $p_k = |\{y_i \in Y : y_i = k\}|/N$, $\forall k = 0, 1, \ldots, K-1$.
Define the Gini index, $G = \sum_{k=1}^{K} p_k(1 - p_k)$.
if $G > 0$ **then**
    Make a partition $\mathcal{R} = \{r_1, r_2, \ldots, r_J\}$, such that $\sum_{r \in R} G_r$ is minimized.
    Define a set of classifiers $\{f_{r_1}, f_{r_2}, \ldots f_{r_J}\}$, such that f_{r_j} is trained by a Decision Tree with training set $\{(\mathbf{x}_i, y_i) \in (X, Y) : \mathbf{x}_i \in r_j\}$ of feature space r_j, $\forall r_j \in \mathcal{R}$.
    Define the classifier $f : \mathcal{Z} \to \{0, 1, \ldots, K-1\} : z \mapsto \{f_r(z) : z \in r\}$.
else
    Define the classifier $f : \mathcal{Z} \to \{0, 1, \ldots, K-1\} : z \mapsto \{k : \max_k(p_k)\}$.

return f

The trained Decision Tree by Algorithm 2 will usually be very large, since the algorithm keeps making partitions until a 'pure' region arises (only observations in the region with similar classifications, or with similar feature observations). Such trees are susceptible for overfitting, a solution would be to limit the number of partitions made, leading to a tree with a bounded number of nodes [60]. We call this limit the maximum depth of the Decision Tree.

Implementation in Python. During the implementation a fixed Python package for Decision Trees is used. In Python the `scikit-learn` machine learning library includes the class `DecisionTreeClassifier`, which includes as the name states a Decision Tree Classifier. The class includes multiple parameters, which influence the behavior of the Decision Tree. The parameter `criterion` represents the function to measure the quality of a partition, for which we choose "gini". Another parameter is `max_depth`, which represents the maximum depth of the tree. If none is given, the nodes of the Decision Tree are expanded until all leaves are pure [73]. The choice of the parameter `criterion` is discussed in subsection 4.6.

4.3.2 Random Forest Classifier

A Random Forest is a assemble of Decision Trees, where a different subset of the training set is used to train each tree, and the Random Forest classifier is based on the majority vote of all Decision Trees of the Random Forest [5].

Constructing a training set for each Decision Tree. Assume we would like to construct a Random Forest consisting of $B \in \mathbb{N}$ Decision Trees. Each Decision Tree uses a training set based on random sampling both the observations and features of the original training set. Given is a training set (X, Y) of feature space $\mathcal{Z}$, with N observations, $M = |F|$ features and K classes. For each Decision Tree $b = 1, 2, \ldots, B$, a training set (X_b, Y_b) of feature space $\mathcal{Z}$, with $n \in \mathbb{N}$ observations and $m \in \mathbb{N}$ features, is constructed with use of the following steps:

(1) Randomly sample a subset $(X_b, Y_b) \subseteq (X, Y)$ of n observations.

(2) Randomly sample a subspace $\mathcal{Z}_b \subseteq \mathcal{Z}$ of m features, such that $\mathcal{Z} \backslash \mathcal{Z}_b = \{z_{b_1}, z_{b_1}, \ldots, z_{b_{M-m}}\} \subseteq \mathbb{R}^{M-m}$. Where the set $\dot{B} = \{b_1, \ldots, b_{M-m}\}$ represents the index of the non-selected features in the original features space $\mathcal{Z}$.

(3) Set $\mathbf{x}_{b_j} = \mathbf{0}$, $\forall b_j \in \dot{B}$.

A training set is constructed by sampling a subset of n observations (step 1) and a subset of m features (Step 2). To maintain a similarly structured feature space for every training set, the inputs of the non-selected features are set to zero (step 3). Similar structure, inside the feature space, is needed to compare the Decision Tree Classifiers for similar input data. For example, if we did not maintain structure, the feature space sampled of Decision Tree A could differ from the feature space of Decision Tree B. Since both Decision Trees have a equal number of features, a observation z can be predicted on both Decision Trees. However, due to feature sampling (step 2), the feature observations in z represents different features than those in Decision Trees A and B, making the prediction invalid.

A clarification regarding the sampling is given in Figure 22.

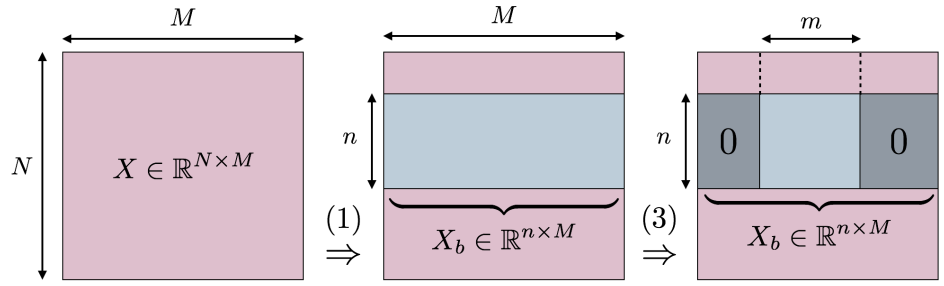


Figure 22: Example of sampling a training set (X_b, Y_b), with n observations and m features, from the initial training set (X, Y), with N observations and M features. For visualization purposes the feature observations X are expressed in matrix form, with N rows of observations $\mathbf{x}_i \in \mathbb{R}^M$.

In Figure 22, the blue fields are the actual sampled values, where the grey areas are values set to zero to ensure similar training data structure for each Decision Tree. Similar structure is needed to compare the Decision Tree Classifiers for similar input data. Note that in the example the sampled observations lay next to each other, this is not mandatory, but used to simplify the visualization.

For all samples the Decision Tree classifiers $f_1, \ldots, f_B$ are determined. After training, the prediction for a single observation is made by taking a majority vote over the predictions of all individual Decision Trees in the Random Forest.

Definition 4.5. Let (X, Y) be a training set of feature space $\mathcal{Z}$, with N observations and K classes. Define the classifier $\hat{f} : \mathcal{Z} \to \{0, 1, \ldots, K-1\}$ as the **Random Forest classifier** of Random Forest with B Decision Trees, such that:

$$\hat{f}(x) = \texttt{round}\left(\frac{1}{B}\sum_{b=1}^{B} f_b(x)\right),$$

with $\{f_1, f_2, \ldots, f_B\}$ being a set of Decision Tree classifiers.

Algorithm 3 Training a Random Forest

Require: The number of Decision Trees B and a training set (X, Y) of feature space $\mathcal{Z}$, with N observations and K classes.

Ensure: A Random Forest classifier $\hat{f} : \mathcal{Z} \to \{0, 1, \ldots, K-1\}$.

1:
2: **for** $b \in \{1, \ldots, B\}$ **do**
3: Sample a subset $(X_b, Y_b) \subseteq (X, Y)$ of n features for feature space $\mathcal{Z}$.
4: Sample a subspace $\mathcal{Z}_b \subseteq \mathcal{Z}$ of m features.
5: Define $\mathcal{Z} \backslash \mathcal{Z}_b = \{z_{b_1}, z_{b_2}, \ldots, z_{b_{M-m}}\}$, with $\dot{B} = \{b_1, \ldots, b_{M-m}\}$.
6: Set $\mathbf{x}_{b_j} = \mathbf{0}$, $\forall b_j \in \dot{B}$.
7: Define the classifier f_b, trained by a Decision Tree with training set (X_b, Y_b).
8:
9: Define Random Forest Classifier $\hat{f} = \texttt{round}(\sum_{b=1}^{B} f_b / B)$.
10: **return** $\hat{f}$

4.3.3 Feature importance

After training the classification model uses the feature importance as a measure to clarify what the actual impact of the features were on the classifiers. While predicting the Bitcoin price the feature importance will give more insights in how our additional features for the Random Forest improves the price prediction. The feature importance of a Random Forest is the average feature importance of all Decision Trees, which is again based on the partition importance inside the Decision Tree [74].

Definition 4.6. Let (X, Y) be a training set with N observations in a feature space $\mathcal{Z}$, with a partition $\mathcal{R} = \{r_1, r_2, \ldots, r_J\}_{J \in \mathbb{N}}$. The variable $q_{\mathcal{R}} \in \mathbb{R}$ is called the **partition importance** of partition $\mathcal{R}$, such that:

$$q_{\mathcal{R}} = G_{\mathcal{Z}} - \sum_{j=1}^{J} \frac{|\{x_j \in X : \mathbf{x}_j \in r_j\}| \cdot G_{r_j}}{N},$$

with $G_{\mathcal{Z}}$ the Gini index of the feature space $\mathcal{Z}$, and G_{r_j} the Gini index of a region $r_j \in \mathcal{R}$, $\forall j \leq J$.

The feature importance of a Decision Tree is calculated by the partition importance of partitions which apply to the feature, divided by the partition importance of all partitions. The higher the value the more important the feature. In Figure 23, the visualization makes it more clear on how the feature importance is calculated.

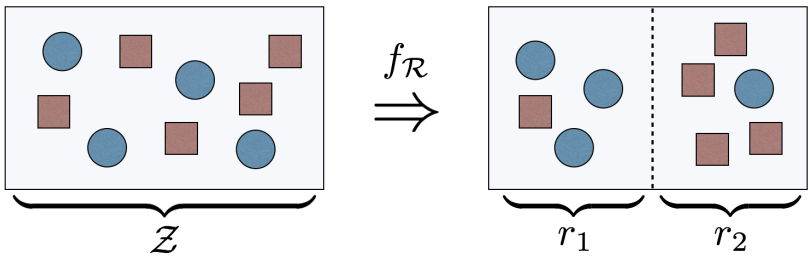


Figure 23: Example of partition importance of partition $\mathcal{R} = \{r_1, r_2\}$ and $N = 9$ observations. It can be calculated that $G_{\mathcal{Z}} \approx 0.49, G_{r_1} \approx 0.38$ and $G_{r_2} \approx 0.16$. Resulting in a partition importance of $q_{\mathcal{R}} = 0.23$

Definition 4.7. Let b be a Decision Tree with a set of partitions $\dot{\mathcal{R}} = \{\mathcal{R}_1, \mathcal{R}_2, \ldots, \mathcal{R}_{\dot{k}}\}_{\dot{k} \in \mathbb{N}}$ and M features. The vector $\mathbf{q}_b = \{\mathbf{q}_b(i)\}^M$, with $\mathbf{q}_b(i) \in [0, 1]$ $\forall i \leq M$, is the **Decision Tree feature importance** of Decision Tree b, such that $\forall i \leq M$:

$$\mathbf{q}_b(i) = \frac{\sum_{\mathcal{R} \in \dot{\mathcal{R}}_i} (q_{\mathcal{R}})}{\sum_{\mathcal{R} \in \dot{\mathcal{R}}} (q_{\mathcal{R}})}, \text{ with } \dot{\mathcal{R}}_i = \{\mathcal{R} \in \dot{\mathcal{R}} : \text{ partition } \mathcal{R} \text{ applies to feature } i\}.$$

The feature importance of a Random Forest will be the average of the feature importance of all Decision Trees in the Random Forest.

Definition 4.8. Given is a Random Forest with B Decision Trees and M features. The vector $\mathbf{q} \in [0, 1]^M$ is the **Random Forest feature importance** such that;

$$\mathbf{q} = \frac{1}{B} \sum_{b=1}^{B} \mathbf{q}_b, \tag{3}$$

with $\mathbf{q}_b \in [0, 1]^M$ the feature importance of Decision Trees $b = \{1, 2, \ldots, B\}$.

Implementation in Python. During the implementation a fixed Python package for the feature importance of a Decision Tree is used. The function `feature_importances_` of the class `DecisionTreeClassifier` gives the feature importance as described.

4.4 Data preparation

The data preparation for the classification model is quite complex, caused by both the size and computational time constraints of the Bitcoin network. In Figure 24 a flowchart is visualized to illustrate the main steps in the data preparation. We distinguish four stages; data collection, window sampling, snowball sampling and data merging.

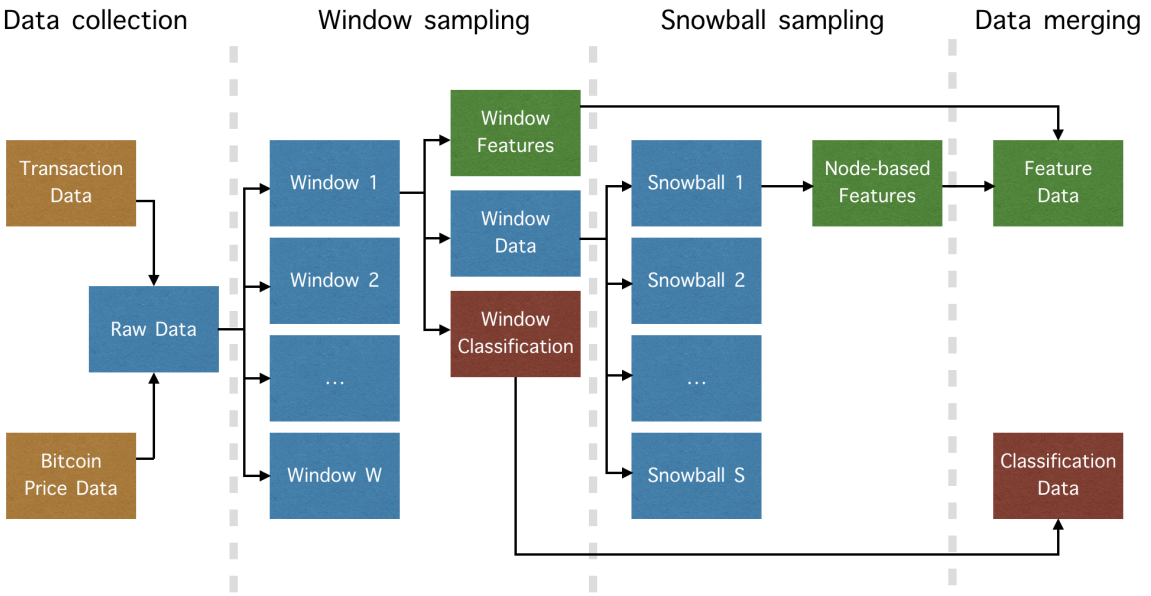


Figure 24: Data preparation can be divided into four stages; Data collection, Window sampling, Snowball sampling and Data merging.

In the data collection stage data is collected from different sources. For the classification model we use two sources: Bitcoin transaction data from MIT and Bitcoin prices from Coinbase (subsection 4.4.2). In the window sampling stage we divide the data in different time windows, which are used as the main observations in the classification model. From each window/observation the corresponding classification and set of features are extracted (subsection 4.4.3). The features are divided into two families; window features and node-based features, where window features are calculated on the overall window and node-based features on node level (subsection 4.4.4). Since node-based features are computationally expensive, the snowball sampling stage is introduced. In the snowball sampling stage different subsets from the window data is taken to compute the different node-based features (subsection 4.4.5). At last, in the data merging stage, all generated data is united in two datasets; one containing all feature data and one containing the classification data. Together they form a pair which is used to train, validate or test the classification model (subsection 4.4.6).

Note that if we use the classification model to predict future data, the labels will be unknown. However, the same data preparation stages will be followed, where the feature data is used as input for the classification model (i.e. the red blocks in the flowchart disappear).

4.4.1 Training, Validation and Testing set

In machine learning, three datasets are commonly used in creation of a classification model. The model is initially fit on a *training set*, that is a set of observations used to fit the parameters of the model [33]. Secondly, the fitted model is used to predict the classifications for the observations in a second dataset called the *validation set*. Finally, the *testing set* is a set used to provide an unbiased evaluation of the final model, fit on the training set and affected by the validation set [7]. We will assume the following training, validation and training sets:

Training set	Transaction and price data from the year 2016.
Validation set	Transaction and price data from the first half year of 2017.
Testing set 1	Transaction and price data from December 2015.
Testing set 2	Transaction and price data from July 2017.

The mathematical definition of a training, validation and testing set is defined in Definition 4.3.

4.4.2 Data collection

For price prediction we used two datasets: one dataset containing Bitcoin transaction information and another dataset containing the Bitcoin prices.

Transaction data. The Bitcoin transaction information is obtained from a third party, the Massachusetts Institute of Technology (MIT). One of their project websites stores all Bitcoin transactions in the first 508241 block of the BlockChain blocks (approximately up to 9 Feb 2018). Through data preparation, which is described in section 2.2, the Bitcoin MIT dataset is constructed.

The Bitcoin MIT dataset contains information about Bitcoin transactions, including the unique transaction ID, the Bitcoin addresses involved, the amount and timestamp of all transactions. The amount is measures in Satoshi, which is the smallest tradable Bitcoin amount of 0.00000001 Bitcoin (1e-8) [75] and the timestamp measures in seconds since Epoch.

A subset of the Bitcoin MIT dataset is taken, such that its timestamps fall inside the training, validation or testing period. For example, if we would like to have the Bitcoin MIT dataset for training, we sample all transaction with their timestamp locating in 2016. The Bitcoin MIT dataset is represented in a weighted directed graph, where each node is a Bitcoin address and each edge is a transaction. Bitcoin mining is represented in the data as a transaction from one user to itself, and is hence manifested in the graph as a self-loop. Transactions involving multiple senders and multiple receivers are neglected due to untraceability of the many-to-many relations [2]. This representation of the data allows us to explore various properties of the graphs and also use them as features in price prediction. More information on how the graph is constructed from the Bitcoin MIT dataset, is found in section 2.2.

Bitcoin prices. In addition, a complete historical listing of Bitcoin prices is used. On Bitcoincharts the entire histories of several Bitcoin exchanges are stored. Due to its completeness, the Bitcoin exchange platform Coinbase in EUR is chosen. For more information about the retrieved Bitcoin price, see section 2.3.

4.4.3 Window sampling

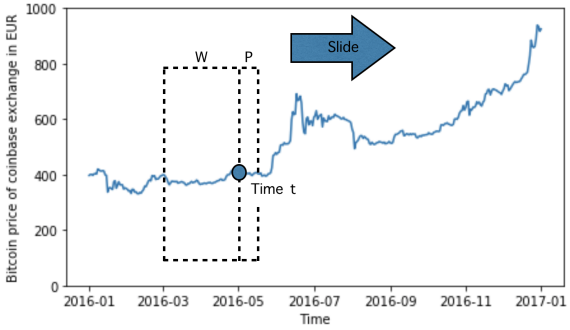
The development of Bitcoin prices over time can be seen as a time-series, which is simply a series of data points ordered in time. To predict future data points, all previous data points or a subset of these points can be used. Our methodology uses a rolling window prediction, where the data is divided in multiple 'windows', where each window predicts one data point.

Argumentation. When predicting time-series, a key assumption is that the model's features are of equal quality over time. Since the Bitcoin market is highly volatile, it may cause outliers and noise in the features. A common technique to assess the stability of a model's features is to compute features over a rolling window of a fixed size through the sample. If the features are truly stable over the sample, then the estimates over the rolling windows should not too large deviations. If the features change drastically at some point during the sample, then the rolling estimates should capture this instability [69].

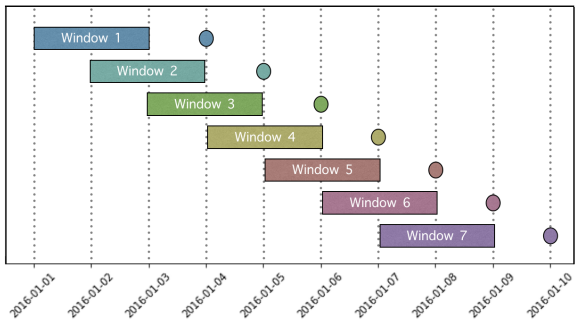
Another argument to use rolling windows is the 'short memory' seen in price prediction. Rolling windows only take into account information of a certain time interval prior to the prediction time. And since Bitcoin prices are mostly influenced by recent developments in the market, recent data is more predictive than older data [22].

Window construction. A single window is constructed in the following way: At time t we estimate the price direction of Bitcoin at the prediction time $t+p$, using data from a fixed interval w before time t. We call w the window size, and p the prediction interval. For all sliding windows, used to train the classification model, the length of time intervals p and w are fixed. In Figure 25a the representation of a single window is made. In Figure 25b is visualized how multiple windows propagate over the time horizon and how the prediction points lay in respect to their windows.

Figure 25



(a) Visualization of a single window, with w the window size and p the prediction interval.



(b) Example of 7 windows (squares) and their prediction points (circles)

For the classification model, assume a window size equal to 2 days ($w = 2$ days). A daily classification is chosen due to the nature of some features, which will be discussed in the upcoming subsection, and therefore we set $p = 1$. Together the training, validation and testing dataset will have a total of 609 prediction points. In Figure 26 some window statistics with respect to the training, validation and testing sets are given.

	Training set	Validation set	Testing set 1	Testing set 2
Raw data taken from	2016	first half of 2017	Dec 2015	Jul 2017
Number of prediction points / windows	366	181	31	31
Number of Bitcoin addresses	9.725.068	6.609.888	470.182	820.555
Number of transactions	8.865.561	5.490.963	408.501	570.117
First prediction point	04-01-2016	04-01-2017	04-12-2015	04-01-2017
Last prediction point	03-01-2017	03-07-2017	03-01-2016	03-08-2017

Figure 26: Some window statistics for the classification model.

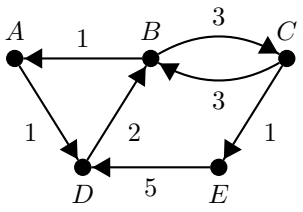
4.4.4 Feature Engineering

Feature engineering is the art of extracting useful patterns from data to make it easier for machine learning algorithms to perform their prediction. In 2015, Dettmers even stated that feature engineering can be considered "the most important skill to achieve good results for prediction tasks" [12]. In addition, Wind concluded, while investigated the behaviour of top performers in Kaggle data mining competitions, that feature engineering is a subjective process requiring a lot of domain knowledge (2014) [67].

The classification model tries to predict the direction of Bitcoin prices on a daily basis. As described in the window sampling stage, for each prediction data 2 days prior to the time of prediction is used. Therefore all features are computed using only information available two days prior to the time of prediction.

Window vs. node-based features. Features for networks can be divided into two families: Window features and node-based features. Window features are calculated over the whole network, and give output on the network level. For example, the number of Bitcoin transactions or the average transaction value in the network. Node-based features can be calculated on both network-, neighborhood- or node-level, and have output on node level. Examples are the in/out degree, or the PageRank of a node.

Node-based features are not commonly used in machine learning algorithms, since their computation is quite expensive. However, node-based features captures more complex relationships between nodes, and since the Bitcoin price is influenced by the interaction between users, these features could prove important. In Figure 27 a visualization is made to show the difference between window and node-based features.



(a) A graph G.

Description	
Number of nodes	5
Number of edges	7
Avg in/out degree	1.4
Mean in/out degree	1
Avg edge weight	3.2

(b) Window features.

Node	in-degree	out-degree	PageRank
A	1	1	0.11
B	2	2	0.40
C	1	2	0.27
D	2	1	0.19
E	1	1	0.03

(c) Node-based features.

Figure 27: Difference between window and node-based features on a graph G.

The choice of a daily prediction is mostly based on the nature of these node-based features, which only make sense in large enough connected graph components for a single window.

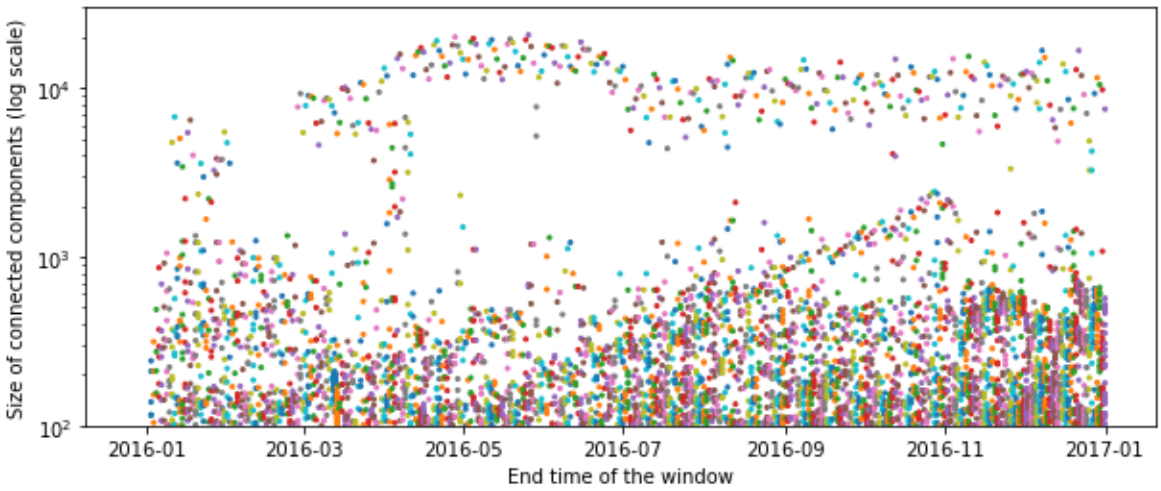


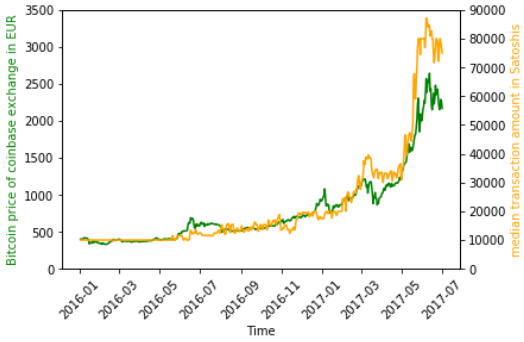
Figure 28: The distribution of the number and size of components over the training set. Windows of 2 days are used, and a log-scale for the y-axis to ease visualization.

In Figure 28 can be seen that windows of two days provided a significant size of connected components, and therefore the window size should be at least equal to two days. By increasing the window size, the number of training points in the classification model fell. To obtain enough accuracy a daily classification was chosen. Another advantage of daily classification is that the daily seasonality is preserved in the classification model.

Again, with the use of node-based features we aim not only to extract important information for the classifiers, but also to gain a general idea of how the dynamics between addresses in the Bitcoin market affect its classification. The biggest downside was that node-based features are computational expensive, for example the 1-ARW-betweenness centrality measure grows exponentially over its network size (Section 3.5.1). To avoid the growth of computation time, each window is sampled through a method called "Snowball Sampling" (subsection 4.4.5).

Feature Selection. Choosing which features to use is an important aspect of any regression or classification model optimization. Reasons are; dealing with a too large feature set will considerably increase training time. In addition, machine learning algorithms can suffer from decreased accuracy if the number of features is significantly higher than the optimal number [41]. And the last and most important reason is that too many features tend to create more variance, causing overfitting [57].

Many of the features are somewhat correlated with the price of Bitcoin. In Figure 29a, the median transaction amount is plotted against the current price, where some correlation is visible. Whereas, in Figure 29b, there exists nearly no correlation between the price and the alpha constant of power-law degree sequence (Definition 2.2).



(a) The median transaction amount per day in Satoshis.

(b) The alpha constant of the power-law degree sequence of the daily transaction network.

Figure 29: The behavior of two different features in comparison with the price of Bitcoin over time.

Before training, we can calculate the mutual information between each feature and the classification already before training. Equation (4) calculates the mutual information I of two jointly discrete random variables X and Y [10]. Here X is a single column of feature data and Y the classification data. The greater the mutual information I, the more informative the feature.

$$I(X;Y) = \sum_{y \in Y} \sum_{x \in X} p_{(X,Y)}(x,y) \log \left(\frac{p_{(X,Y)}(x,y)}{p_X(x) p_Y(y)} \right) \tag{4}$$

Here $p_{(X,Y)}$ is the joint probability mass function of X and Y, and p_X, p_Y are the marginal probability mass functions of X and Y respectively. The mutual information is calculated with help of the `normalized_mutual_info_score` function of the `sklearn.metrics` class in Python. Some features proved uninformative, and had mutual information scores barely above the change of a 'coin flip'. These features will be neglected while training the classification model, to reduce computation time.

Table 5 illustrates the mutual information of all described features in the training set. The informative-found features are characterized by an asterisk (*).

Table 5

Window features*	I
Current Bitcoin price*	0.660
Number of active Bitcoin addresses*	0.659
Number of transactions*	0.661
Mean transaction value*	0.669
Median transaction value*	0.796
Average in/out node degree*	0.659
Median node degree	0.500
Alpha constant of power-law degree sequence*	0.659
Percentage new addresses*	0.659
Percentage transactions performed by new addresses*	0.659

Node-based features	I
In degree	0.501
Out degree	0.501
1-ARW-betweenness centrality*	0.765
PageRank*	0.762
Closeness centrality	0.521
Total Bitcoin passing through	0.512
Netto Bitcoin flow (received minus sent)	0.509

The mutual information I only represents the one-to-one relation a feature has on the Bitcoin price. However, a subset of features with low mutual information can still have a large influence on the Bitcoin price due to their underlying covariance. Consider two highly correlated features, implying high covariance, which each focuses on a different aspect of the Bitcoin price. The mutual information of the two features combined should be higher than their individual mutual information, and therefore could be informative enough for the classification model. In Figure 30 the covariance matrix of both window and node-based features is visualized.

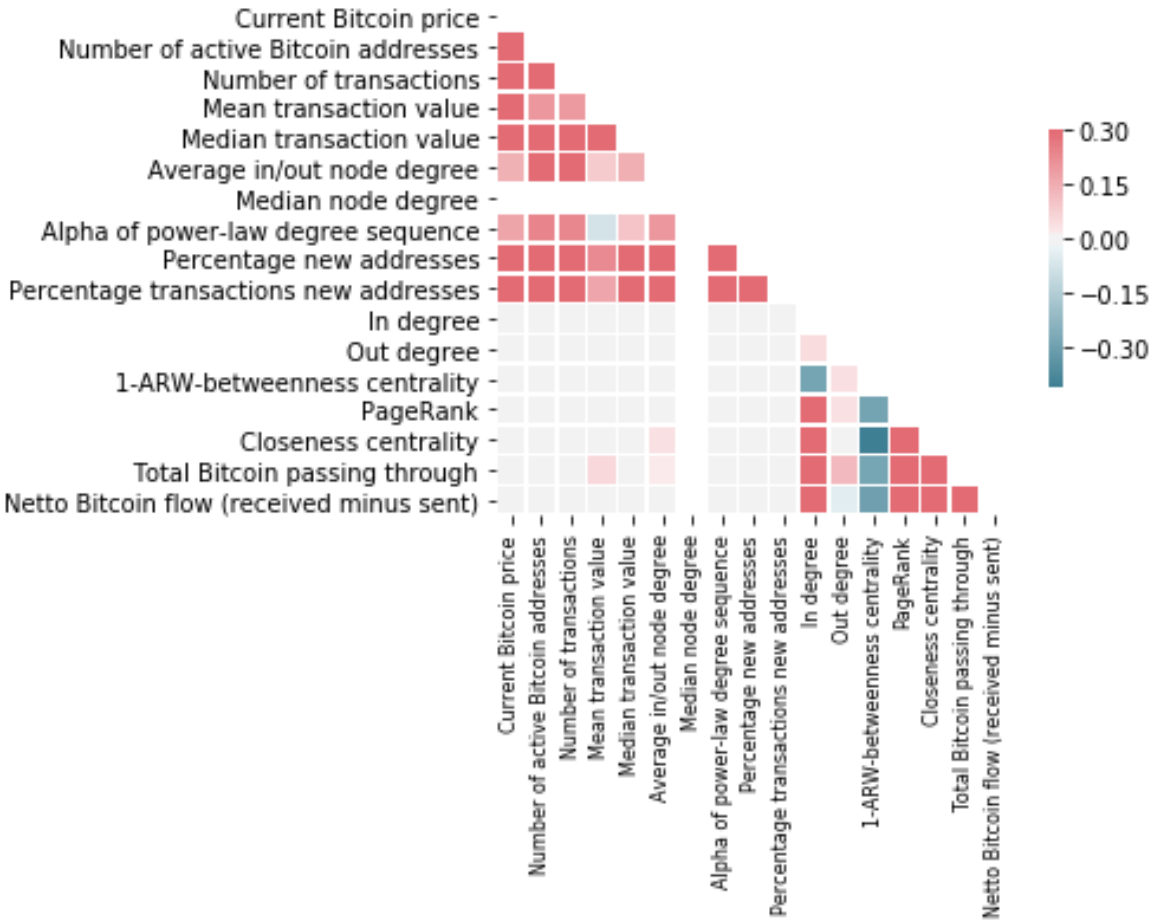


Figure 30: Covariance matrix of features.

In Figure 30 can be seen that the covariance between window and node-based features are small. This is caused by less variety in the window feature observations with respect to the node-based features. The row and column representing the median node degree is barely visible, since the values of this feature is equally for all observations in the training set, which makes the feature insignificant. Furthermore, there exists no high covariance between the already low-informative features, which would influence the mutual information of the two features combined.

The code for retrieving the window features is stored in Appendix B.5, and for the node-based features in Appendix B.6.

4.4.5 Snowball Sampling

A common problem in network analysis is the fact that several interesting network features become impractical for large graphs [43]. To reduce computation time, graph sampling is essential. In our research we choose a method for graph sampling called 'Snowball Sampling'.

Snowball Sampling is a variant of the Breadth First Search Algorithm, which starts at an arbitrary node, and explores all of the neighbor nodes at the present depth prior to moving to the nodes at the next depth level [62]. In addition, Snowball Sampling has a limit on the number of neighbors $|\mathcal{N}|$ that are added to the sample [76].

The choice of Snowball Sampling is based on the property of getting highly dense connected components in the samples, which only make sense if we use measures as betweenness, closeness, etc. [30]. In 2018, Ashish Aggarwal from MIT uploaded on a public github multiple graph sampling algorithms, including a Snowball Sampling algorithm [76]. We copied a part of the code, and included some extra properties needed for our research, the updated code can be found in Appendix B.4

Algorithm 4 Snowball Sampling Algorithm

Require: A graph $G = (V, E)$, the desired size of the snowball graph N_s, the minimal size of the connected components N_c and the max number of sampled neighbours per iteration $|\mathcal{N}|$.
Ensure: A snowball graph G_s

Define $G_c = (V_c, E_c)$ as the graph of all connected components in G larger or equal than N_c.
if $|V_c| \leq N_s$ **then return** G_c

Choose a arbitrary root node $v \in V_c$.
Define the sets $S = \{v\}$ and $D = \emptyset$.
do
    if $S \backslash D = \emptyset$ **then**
        if $N_s - |S| < N_c$ **then break.**
        Choose another root node $v \in V_c \backslash D$
        Update $S = S \cup \{v\}$
    Choose a arbitrary node $s \in S \backslash D$
    Sample, if possible, $|\mathcal{N}|$ unique neighbours from s, call them $V_{\mathcal{N}}$

    if $|\mathcal{N}| \leq |V_{\mathcal{N}}|$ **then** $D = D \cup \{s\}$
    if $|S \cup V_{\mathcal{N}}| \leq N_s$ **then**
        Update $S = S \cup V_{\mathcal{N}}$
    else
        Sample $N_s - |S|$ nodes from $V_{\mathcal{N}} \backslash S$, call them $\hat{V_{\mathcal{N}}}$
        Update $S = S \cup \hat{V_{\mathcal{N}}}$
while $|S| < N_s$

Define $E_s = \{(s_1, s_2) | (s_1, s_2) \in E_c \text{ and } s_1, s_2 \in S\}$
Define $G_s = (S, E_s)$ as a snowball graph of G.

return G_s

Outline Algorithm. Algorithm 4 describes the Snowball Sampling Algorithm. As input for the Algorithm a graph $G = (V, E)$ is given and three parameters: the desired size of the snowball graph N_s, the minimal size of the connected components N_c and the maximum number of sampled neighbours per iteration $|\mathcal{N}|$. In the first step of the algorithm the overall graph is filtered such that only connected components of size larger than N_c remain (Line 1). If the number of nodes in G_c is smaller than the desired snowball graph size, the filtered graph G_c is the target snowball graph (Line 2). In the Algorithm we consider two sets; S and D, in which S lists all the nodes in the snowball graph and D the nodes which are already sampled from (Line 4).

In each iteration the Snowball Sampling Algorithm adds $|\mathcal{N}|$ neighbours of an arbitrary node in the snowball graph, starting at a root node v. This comes with some obstacles; (1) If the root note v lies in a component smaller than the desired snowball graph, we need to add the whole component and sample a new root node from the remaining nodes in V_c (lines 7, 9 & 10). (2) While choosing a new root node the availability in the snowball graph should be larger or equal than N_c. Otherwise we will add connected components smaller than N_c to the snowball graph (line 8). (3) While adding neighbours the size of the snowball graph can be exceeded, in line 15 it is checked whether this occurs, after which the Algorithm terminates (line 18 & 19).

There are to two reasons why the size of the obtained snowball graphs can be smaller than the desired size N_s. First, after sampling out the minimal size of the connected components, the number of nodes in graph G_c is smaller than the desired snowball graph (line 2). And second, after choosing a new root node there is no space left in the snowball graph for a new component of minimal size N_c (line 8).

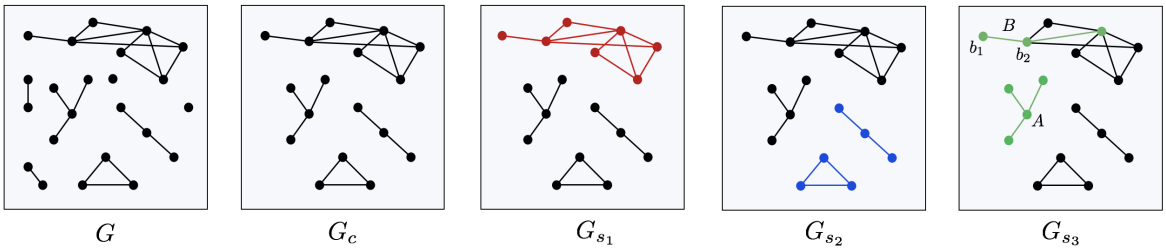


Figure 31: Three valid examples of snowball graphs s_1, s_2 and s_3, each sampled from the original graph G. The snowball graphs are sampled with use of Algorithm 4 and parameters $N_s = 7$, $N_c = 3$ and $|\mathcal{N}| = 2$.

Let's look at Figure 31, snowball graph G_{s_1} is the most simple snowball graph, where all nodes originate from the same connected component.

Snowball graph G_{s_2}, only has 6 nodes not the desired size $N_s = 7$. Here obstacle (1) and (2) occur. (1): The size of both components is smaller than the desired size N_s and therefore the entire components will be added to the snowball graph. (2): The remaining availability in the snowball graph is equal to 1, which is smaller than the minimal size of a connected components N_c, after which the algorithm terminates.

Snowball graph G_{s_3} is a combination of obstacles (1) and (3), we assume that from component A the initial root node was sampled. (1): The size of the initial component (denoted by A), is smaller than the desired size N_s and therefore added to the snowball graph. (3): Assume node b_1 was the root node of component B. In iteration 1, the node b_2 was added to the snowball graph. In iteration 2, we should add $|\mathcal{N}|$ neighbours of b_2 to the graph. However by adding those neighbours, the desired size N_s is exceeded. To reach the desired size, only one neighbour of b_2 is added to the snowball graph.

4.4.6 Data merging

Feature and classification observations are both generated in the window- and snowball- sampling stages of the data preparation. In the last stage, the data merging stage, all generated observations are collected and transformed into the right format. Since many assumptions are made throughout the earlier stages, Table 6 sketches an overview of all assumptions and notations for the classification model.

Description	Notation	Training set	Validation set	Testing set 1	Testing set 2
Raw data taken from	-	2016	first half of 2017	Dec 2015	Jul 2017
Number of Bitcoin addresses	$\|V\|$	9.725.068	6.609.888	470.182	820.555
Number of transactions	$\|E\|$	8.865.561	5.490.963	408.501	570.117
prediction interval	p	1 day	1 day	1 day	1 day
Window length	w	2 days	2 days	2 days	2 days
Number of windows	W	366	181	31	31
First prediction point	-	04-01-2016	04-01-2017	04-12-2015	04-07-2017
Last prediction point	-	03-01-2017	03-07-2017	03-01-2016	03-08-2017
Number of classes	K	2	2	2	2
Total number of features	M	11	11	11	11
Number of window features	M_w	9	9	9	9
Number of node-based features	M_n	2	2	2	2
Number of snowball graphs per window	S_w	50	50	50	50
Maximal number of neighbourhood sampling	$\|\mathcal{N}\|$	100	100	100	100
Maximal size of snowball graph	N_s	1000	1000	1000	1000
Minimal size of connected component in snowball graph	N_c	100	100	100	100

Table 6: Overview off all assumptions made during data preparation.

The output of the data merging stage should be applicable to a Random Forest classification model. Definition 4.3 defines a training, validation and testing set such that they can be used as inputs for a Random Forest, which is desirable. It concludes that:

$$(X, Y) = \{(\mathbf{x}_i, y_i)\}_{i \leq N},$$

is called a training, validation or testing set, with $N \in \mathbb{N}$ observations, $K \in \mathbb{N}$ classes and a feature space $\mathcal{Z}$. With $X \in \mathbb{R}^{N \times M}$ being the collection of feature observations, with N rows of **feature observations** $\mathbf{x}_i \in \mathcal{Z}$, $\forall i \leq N$. And $Y \in \{0, 1, \ldots, K-1\}^N$ the collection of classifications, with **classifications** $y_i \in \{0, 1, \ldots, K-1\}$, $\forall i \leq N$.

Constructing feature observations $\mathbf{X}$**.** Assume the set X represent all generated feature observations in the data preparation stage. In both the window- and snowball- sampling stage feature data is generated.

In the window sampling stage, a feature space $\mathcal{Z}_w$ of M_w window features is considered, such that their feature observations have dimension $\mathbf{x}_i \in \mathbb{R}^{M_w}$. In the Snowball Sampling stage, the features space $\mathcal{Z}_n$ has M_n node-based features, such that their feature observations have dimension $\mathbf{x}_i \in \mathbb{R}^{M_n}$. Each window has multiple snowball graphs, causing a large amount of feature observations in the node-based

feature space. All node-based feature observations are linked to the window feature observation of the window on which their snowball graph is sampled. This enables a joint feature space $\mathcal{Z} = (\mathcal{Z}_w, \mathcal{Z}_n)$ of M features, such that $\mathbf{x}_i \in \mathbb{R}^M$.

In total there are $W \cdot S_w \cdot N_s$ feature observations in set X. This is derived from having W windows, having each S_w snowball graphs of N_s nodes, causing a total of $W \cdot S_w \cdot N_s$ node-based feature observations. Mathematically,

$$X = \{(\mathbf{x}_w, \hat{\mathbf{x}}_{w_{s_i}}) \in (\mathcal{Z}_w, \mathcal{Z}_n) : w = w_{s_i}\} \tag{5}$$

with w_{s_i} being the window w from which the snowball graph s_i is sampled, and the set of snowball graphs $S = \{s_1, s_2, \ldots, s_{W \cdot S_w}\}$.

***Constructing classifications* $\mathbf{Y}$.** Assume the set Y represents all generated classification data. In the window sampling stage, a total of W classifications $y_i \in \{0, 1\}$ are generated. Since X and Y are paired sets, the window feature observations should correspond with the window classifications of the same window. Extending Equation (5) under this assumption, gives us:

$$(X, Y) = \{(\mathbf{x}_w, \hat{\mathbf{x}}_{w_{s_i}}, y_w) \in (\mathcal{Z}_w, \mathcal{Z}_n, \{0, 1\}) : w = w_{s_i}, s_i \in S\} \tag{6}$$

with again w_{s_i} being the window w from which the snowball graph s_i is sampled, and the set $S = \{s_1, s_2, \ldots, s_{W \cdot S_w}\}$ of snowball graphs.

Example. Assume the feature spaces $\mathcal{Z}_w, \mathcal{Z}_n \subseteq \mathbb{R}^2$, the number of classes $K = 2$, the number of windows $W = 2$, with each $S_w = 2$ snowball graphs with $N_s = 2$ nodes. On basis of the assumptions there exists two window feature observations $\mathbf{x}_w \in \mathcal{Z}_w$ and classifications $y_w \in \{0, 1\}$, define them as:

$$\mathbf{x}_1 = \begin{bmatrix} 150 \\ 26 \end{bmatrix} \text{ with } y_1 = 1, \text{ and } \mathbf{x}_1 = \begin{bmatrix} 265 \\ 12 \end{bmatrix} \text{ with } y_1 = 0.$$

In each window w four node-based feature observations $\hat{\mathbf{x}}_{w_{s_i}} \in \mathcal{Z}_n$ are made on independent sampled snowball graphs $S = \{s_1, \ldots, s_{W \cdot S_w}\}$, such that:

$$\hat{\mathbf{x}}_{s_1} = \underbrace{\left\{ \begin{bmatrix} 0.543 \\ 0.369 \end{bmatrix} \right.}_{\text{node 1}}, \underbrace{\left. \begin{bmatrix} 0.987 \\ 0.489 \end{bmatrix} \right\}}_{\text{node 2}}, \ldots, \hat{\mathbf{x}}_{s_{W \cdot S_w}} = \left\{ \begin{bmatrix} 0.365 \\ 0.369 \end{bmatrix}, \begin{bmatrix} 0.478 \\ 0.458 \end{bmatrix} \right\}$$

Through combing all feature observations and classifications the set (X, Y) is constructed, which is visualized in Figure 32.

Window w	Snowball graph s_i	Node-based feature 1	Node-based feature 2	Window feature 1	Window feature 2	Classification
1	s_1	0.543	0.369	150	26	1
1	s_1	0.987	0.489	150	26	1
1	s_2	0.245	0.785	150	26	1
1	s_2	0.365	0.851	150	26	1
2	s_3	0.158	0.951	265	12	0
2	s_3	0.789	0.168	265	12	0
2	s_4	0.365	0.369	265	12	0
2	s_4	0.478	0.458	265	12	0

Feature observations X | Classifications Y

Figure 32: Example of the data merging stage.

4.5 Impact of snowball sampling on the classification model

The classification model will be impacted by the snowball sampling. Usually the observations in a set (X, Y) are independent, where each observation has one set of features observations and a single classification. With the introduction of snowball sampling, all node-based features inside a single snowball graph become dependent, while the node-based features between different snowball graphs are independent. In Figure 33 a example is given.

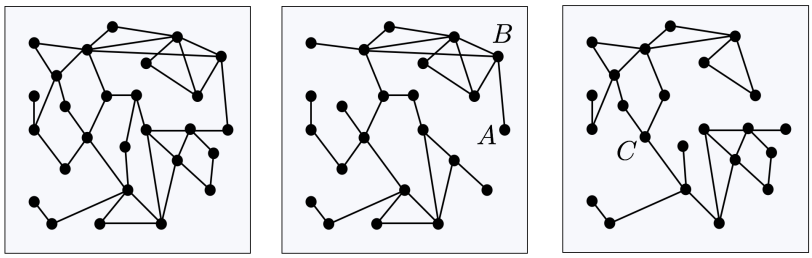


Figure 33: Two snowball graphs s_1 and s_2, sampled from the original graph G. Changes in the 1-ARW-betweenness centrality of node A imply changes in the centrality of its neighbour node B, making the two observations dependent. On the other hand, changes in the 1-ARW-betweenness centrality of node A in snowball graph s_1 will not have a influence on the centrality of node C, originated in another snowball graph s_2.

Snowball Sampling can be seen as a 'pre-sampling' stage of a Random Forest and to overcome the problem of feature dependence, a subset of whole snowball graphs are used to train and predict the Decision Trees of the Random Forest.

Training. In Algorithm 5 the adjusted Random Forest training algorithm, trained on whole snowball graphs, is stated. Note that the output from this Algorithm is a set of Decision Tree Classifiers and not a single Random Forest Classifier. Further on we will see why this is necessary.

Algorithm 5 Training a Random Forest, based on snowball graphs

Require: The number of Decision Trees B and a training set (X, Y) of feature spaces $\mathcal{Z}_w$, $\mathcal{Z}_n$ and $\{0, 1\}$, with $W \cdot S_w \cdot N_s$ observations and the set $S = \{s_1, s_2, \ldots, s_{W \cdot S_w}\}$ of snowball graphs.
Ensure: A set of Decision Tree Classifiers $\{f_1, f_2, \ldots, f_B\}$

1:
2: **for** $b \in \{1, \ldots, B\}$ **do**
3: Sample a subset $\hat{S} \subseteq S$ of s snowball samples.
4: Define the set $(X_b, Y_b) = \{(\mathbf{x}_w, \hat{\mathbf{x}}_{w_{s_i}}, y_w) \in (\mathcal{Z}_w, \mathcal{Z}_n, \{0, 1\}) : w = w_{s_i}, s_i \in \hat{S}\}$,
5: which is a subset of the initial training set (X, Y).
6: Sample a subspace $\mathcal{Z}_b \subseteq (\mathcal{Z}_w, \mathcal{Z}_n)$ of m features.
7: Define $(\mathcal{Z}_w, \mathcal{Z}_n) \backslash \mathcal{Z}_b = \{z_{b_1}, z_{b_2}, \ldots, z_{b_{M-m}}\}$, with $\dot{B} = \{b_1, \ldots, b_{M-m}\}$.
8: Set $\mathbf{x}_i(b_j) = 0$, $\forall \mathbf{x}_i(b_j) \in X_b$ with $b_j \in \dot{B}$.
9: Define the classifier f_b, trained by a Decision Tree with training set (X_b, Y_b).
10:
11: **return** $\{f_1, f_2, \ldots, f_B\}$

The impact of the snowball sampling is seen in line 3 & 4, where a subset of the snowball graphs $\hat{S}$ is used as a requirement for the sampled training set (X_b, Y_b). The variable w_{s_i} is the window w from

which the snowball graph s_i is sampled. The Python code which is used to train, validate, and test the Random Forest is not included in the Appendix. For more information about the Random Forest Python code please contact the author.

Predicting. The prediction of the classification model, based on snowball graphs, is also made for whole snowball graphs. The classification of a snowball graph is equal to the majority vote of the classifications of all nodes in the snowball graph. A observation has a single window with (possible) multiple snowball graphs, its classification can be determined by taking the majority vote of the classifications of all snowball graphs.

Definition 4.9. Let (X, Y) be a training set of feature space $\mathcal{Z}_w, \mathcal{Z}_n$ and $\{0, 1\}$, with $W \cdot S_w \cdot N_s$ observations and the set $S = \{s_1, s_2, \ldots, s_{W \cdot S_w}\}$ of snowball graphs. Define the classifier $\hat{f} : \mathcal{Z} \to \{0, 1, \ldots, K-1\}$ as the **Random Forest classifier, based on snowball graphs** of Random Forest with B Decision Trees, such that:

$$\hat{f}(x) = \texttt{round}\left(\frac{1}{S}\sum_{i=1}^{S}\texttt{round}\left(\frac{1}{B}\sum_{b=1}^{B} f_b((\mathbf{x}_w, \hat{\mathbf{x}}_{w_{s_i}})\right)\right).$$

with $\{f_1, f_2, \ldots, f_B\}$ the set of Decision Tree Classifiers, trained in Algorithm 5.

Twice a majority vote is taken, Initially for each snowball graph, and secondly over the collection of snowball graphs, i.e. a window.

4.6 Implementation on the Bitcoin MIT dataset

The Bitcoin MIT dataset represents a Bitcoin transaction network. The source contained all Bitcoin transactions in the first 508241 blocks (approximately up to 9 Feb 2018). Since constructing the whole transaction network is too computationally expensive for normal computers, we only used the transaction data between 1 December 2015 and 31 July 2017.

The Bitcoin MIT dataset provides five entries per line; transactionID, in-address, out-address, amount and timestamp. The *transactionID* is the unique index of a transactions, *in-address* notates the node id of the sending Bitcoin address, *out-address* the node id of the receiving Bitcoin address, *amount* the amount of Bitcoin transferred in Satoshi, and *timestamp* the time of the transaction, measured as seconds since Epoch.

We can construct a weighted directed graph $G = (V, E, w)$ by taking all in/out-addresses as nodes V and the made transactions as the edges E. The direction of the edges is from in-address to out-address, and their amounts are used as weights w. The graph can be constructed for different subsets of the Bitcoin MIT dataset. For more detailed information on the Bitcoin MIT dataset, we refer to subsection 2.2.

The goal of the price prediction was to see if node-based features improved the quality of prediction. Most current work only use window features in their machine learning algorithms, which does not explore the relationship between Bitcoin prices and other more complex features in the space. With the use of a benchmark, based on only the window features, we can see if the addition of node-based features indeed improves the classification model.

4.6.1 Benchmark

Benchmarks can be seen as a reference point against which something can be measured, compared, or assessed. In our case, the use of only window features is used as a benchmark to measure the influence of the node-based features.

Training. Consider a training set (X, Y), with N observations, M window features, W windows and two classes: $K = \{0, 1\}$. Since window features don't require snowball sampling, a simple Random Forest is used to train and predict the classification model (subsection 4.3.2). The observations are obtained in the window sampling stage of the data preparation (subsection 4.4) and the Random Forest is trained by Algorithm 3 (subsection 4.3.2). In Table 7 some simple statistics of observations in the training set are given.

	Classification value 1	**Classification value 0**
Description	Bitcoin price grows	Bitcoin price stays equal or decreases
Total	206	160
Percentage	57%	43%

Table 7: Classifications of the observations in the training set.

Validation. After training, the classifications of the validation set are predicted using the trained Random Forest. The difference between the predicted classifications and actual classifications are minimized by adjusting the parameters in the classification model. The optimal parameters of the classification model, for the validation set, are listed in Table 8.

Description	Notation	Value
Number of regions per partition inside a Decision Tree	J	2
Maximum depth Decision Tree	-	4
Number of observations used in each Decision Tree of the Random Forest	n	$0.60 \cdot N$
Number of features used in each Decision Tree of the Random Forest	m	$0.75 \cdot M$

Table 8: Optimal parameters for the classification model, with respect to the validation set.

After optimizing the parameters for the Random Forest, we calculate its feature importance Q for the Random Forest. More information about the feature importance can be found in subsection 4.3.3, where Equation (3) elaborates the details. The results are visualized in Figure 34. The number of active Bitcoin addresses is the most significant features impacting the classification model, whereas the median transaction value has the lowest significance.

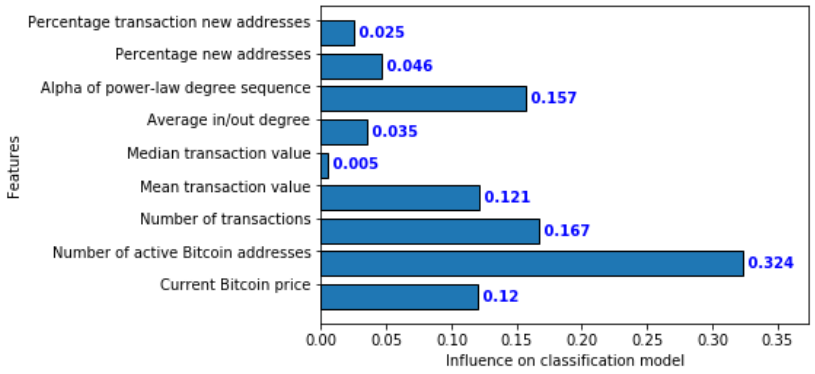


Figure 34: Feature importance of the Random Forest, trained on only window features.

In Figure 35a, the behavior of the Bitcoin price is visualized over time. The colors in the figure indicate a correct or false classification made by the classification model. In total, 64% of the observations were classified correctly.

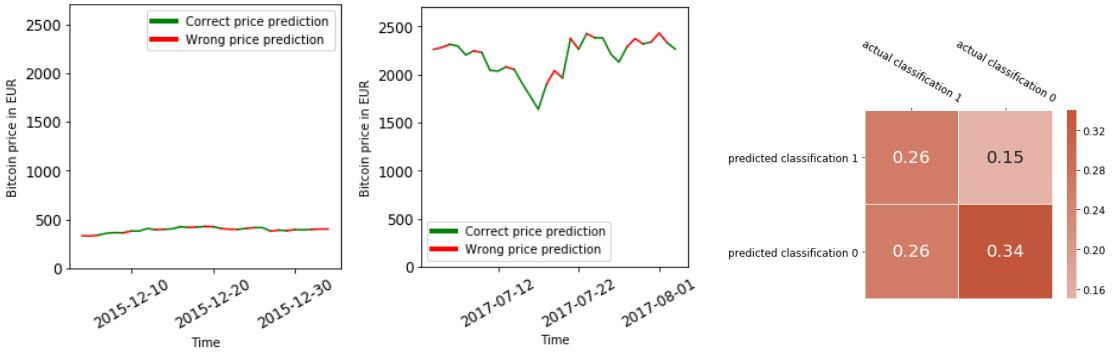
Figure 35b illustrates the distribution of the predicted classification with respect to the actual classifications. Approximated 30% of all actual observations are classified as 0, where the Bitcoin price decreases. The classification model only predicted a small portion of these observations correctly. The cause could be that the classification model is better trained in distinguishing classification value 1, instead of 0. This makes sense since a more substantial proportion of the training set has classification value 1. Another important result is that the classification model nearly predicts all observations, having a classification 1, correctly. This also indicates that the classification model is better in predicting the increase of the Bitcoin price, having classification value 1.

(a) Validation of the Random Forest.

(b) Actual vs. predicted.

Figure 35: Validation results of the Random Forest, trained on the window features. In total 64% of all observations are classified correctly.

Testing. The testing set provides an unbiased evaluation of the classification model, fit on the training set and affected by the validation set. There is chosen for two testing sets, a small interval before the training set and a small interval post the validation set. In Figure 36a, the behavior of the Bitcoin price is visualized over time. The colors in the figure indicate a correct or false classification made by the classification model. In Figure 36b, a similar chart is presented, but then for testing set 2.



(a) Test set 1.

(b) Test set 2.

(c) Actual vs. predicted.

Figure 36: Validation results of the Random Forest, trained on the window features. In total 60% of all observations are classified correctly.

Figure 36c gives the distribution of the predicted classification with respect to the actual classifications is illustrated. In total, 60% of all observations were classified correctly. This percentage is a bit lower than those for the validation set. This is caused by the fact that the parameters of the trained model are influenced by the validation set and is therefore biased. The classification model was better in predicting class 1 than class 0 in the validation set. However, in both testing sets, this is not the cause. This suspects that the classification model isn't superior in classifying class 1.

4.6.2 Improvement with use of node-based features

Similar steps are taken to train, validate, and test the Random Forest, based on both window and node-based features. The main difference is that Algorithm 5 trains the Random Forest and that our observations are dependent and of a higher dimension. To support the benchmark, similar optimal parameters are used for training the Random Forest as listed in Table 8. The statistics for the classifications of the training set are still the same as in Table 7.

Validation. In Figure 37, the new feature importance of the Random Forest is illustrated. Still, the number of active Bitcoin addresses is the most important feature. However, the node-based features PageRank and 1-ARW-betweenness centrality indicate significant importance, and therefore we suspect that the price prediction on Bitcoin will improve with the use of these new features. It is interesting to see that the order of importance of the window based features is similar as in our benchmark, which would be as expected since we don't alter essence of the window features.

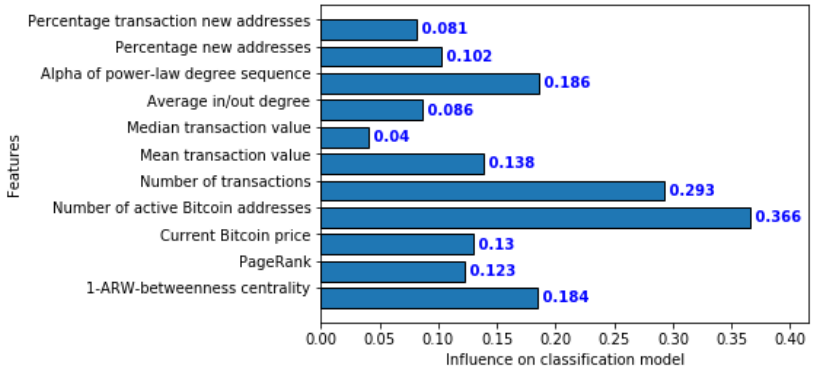


Figure 37: Feature importance of the Random Forest, trained on both window and node-based features.

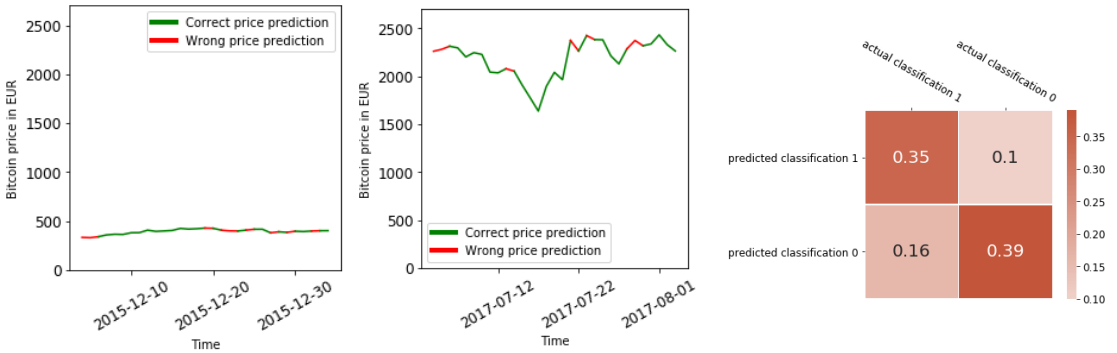
In Figure 38a the behavior of the Bitcoin price is visualized over time. The colors of the figure indicate a correct or false classification made by the classification model. In total, 73% of the observations were classified correctly. Figure 35b illustrates the distribution of the predicted classification with respect to the actual classifications. Concerning the benchmark, the classification model using node-based features is better in predicting the classification of value 0.

(a) Validation of the Random Forest.

(b) Actual vs. predicted.

Figure 38: Validation results of the Random Forest, trained on both window and node-based features. In total 73% of all observations are classified correctly.

Testing. The testing set provides an unbiased evaluation of the classification model, fit on the training set and affected by the validation set. In Figure 39a, the behavior of the Bitcoin price is visualized over time. The colors in the figure indicate a correct or false classification made by the classification model. In Figure 39b, a similar chart is presented, but then for the testing set 2. In Figure 36c, the distribution of classifications is illustrated. In total, 74% of all observations were classified correctly. With respect to the benchmark, both classes are better predicted.



(a) Test set 1. (b) Test set 2. (c) Actual vs. predicted.

Figure 39: Validation results of the Random Forest, trained both window and node-based features. In total 74% of all observations are classified correctly.

Conclusion. The Bitcoin price classification improves with the use of node-based features. The benchmark, based on solely window features, reached an accuracy of 60% correct classifications for the testing sets. After adding the node-based features, this accuracy reached a level of 74% correct classifications. This proves that indeed, the node-based features affect the Bitcoin market price and improves its classification with the use of a Random Forest.

In Figure 40 an overview is given for the training, validation, and testing sets used to train the Random Forest, based on both window and node-based features.

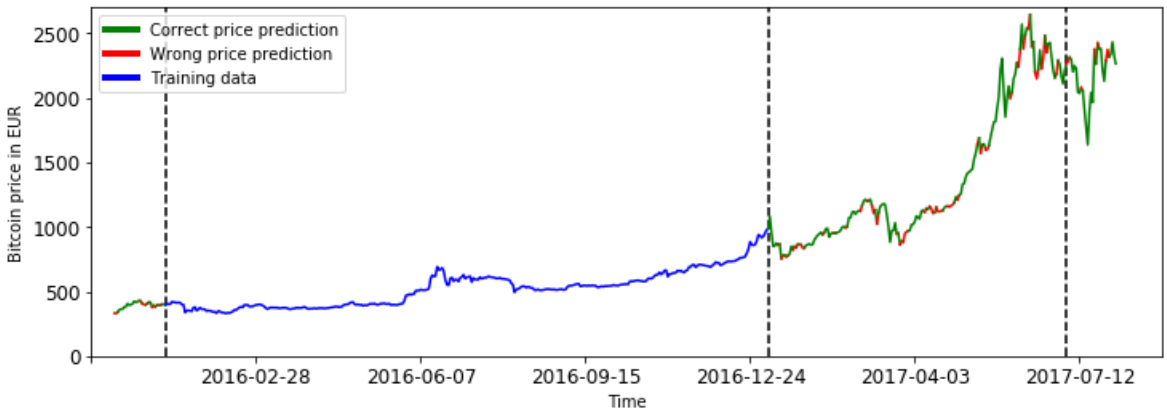


Figure 40: Overview of the evaluation of Bitcoin price, where the coloring indicated a correct or false classification mad by the classification model.

Conclusion

The machine learning algorithm Random Forest is used to predict the sign of the Bitcoin price, which is a classification problem with two classes, up or down. In this research, additional features are investigated to improve the price prediction of Bitcoin.

The researched additional features are the "fairness and goodness measure and the "1-ARW-betweenness centrality measure. Since the who-trust-whom network, on which the fairness and goodness measure was based, couldn't be linked to the transaction network, the fairness and goodness measure can't be used as an additional feature in the price prediction. The 1-ARW-betweenness centrality can be regarded as a measure of how fast all possible money flows towards a node inside the network. The time complexity for the 1-ARW-betweenness centrality is of importance since it can make a big difference as to whether the measure is practical for large graphs. It is shown that the computation time grows exponentially, to the number of nodes in a network.

The 1-ARW-betweenness centrality is a node-based feature, which means that the measure has output on node-level and can capture more complex relationships between nodes. A common problem in network analysis is that the node-based features become impractical for large graphs, as seen with the 1-ARW-betweenness centrality. Therefore the classification model will rely on graph sampling to reduce the computation time. Since graph sampling can be seen as a 'pre-sampling' stage of a Random Forest, the choice of the Random Forest learning method is substantiated.

To maintain stationarity and independence in the training data, rolling window prediction is used, such that the classification model uses a two-day window to predict a daily Bitcoin price classification. The choice of a daily forecast is mostly based on the nature of the node-based features, which only make sense in large enough connected graph components for a single window. Seen was that windows larger than two days provided a significant size of connected components. However, by increasing the window size, the number of training points in the classification model fell. To obtain enough accuracy, a daily classification was chosen.

As expected, the time complexity of the Random Forest increased massively, since most node-based features are computationally expensive and grow exponentially over its network size. To avoid the growth of computation time, a graph sampling method 'Snowball Sampling' is introduced. The graph sampling method constructed highly dense connected components of the original network, on which the node-based features were calculated. The snowball sampling had an impact on training the Random Forest since some observations in the training set became dependent. An adjusted training algorithm was introduced to overcome these problems, with the main difference that samples existing out of whole snowball graphs were used while training the individual Decision Trees inside the Random Forest.

The Bitcoin price classification indeed improves with the use of node-based features. The benchmark, based on solely window features, reached an accuracy of 60% correct classifications with respect to the testing sets. After adding two node-based features; PageRank and 1-ARW-betweenness centrality, the accuracy reached a level of 74% correct classifications. This proves that the node-based features have some effect on the Bitcoin market price and improves its classification with the use of a Random Forest.

Future work

The Bitcoin transaction network is presented in a weighted directed graph, where each node is a Bitcoin address, and each edge is a transaction. The transactions involving multiple senders and multiple receivers are neglected due to the untraceability of the many-to-many relations. Some researches have been devoted towards de-anonymizing the Bitcoin network, where they transfer the transaction network into a user network. The goal is grouping Bitcoin addresses belonging to the same individual or entity together, to reduce the untraceability of the many-to-many relations. The entity graph will consist of the same or more of transactions, with fewer nodes (now entities instead of individual Bitcoin addresses). This causes the network to become more dense and applicable to node-based features.

The introduction of the entity graph also influences the choice of window size used. A daily price prediction was chosen due to the nature of the node-based features, which only make sense in large enough connected graph components for a single window. Seen was that windows larger than two days provided a significant size of connected components. In the entity graph, the window size will probably be smaller to maintain big enough connected graph components for a single window. A result is that the classification model can obtain more training observations to predict the fluctuations in Bitcoin price.

The used Random Forest uses a fixed set for the training, validation, and testing set. A result is that we aren't sure that the model has the desired accuracy and variance on data that it has not seen before, besides the used testing sets. Cross-validation is a technique that is used to test the effectiveness of machine learning algorithms over different testing sets inside the overall dataset. With the use of cross-validation, the results of the classification model should become less biased, since it ensures that every observation from the original dataset has the chance of appearing in training or testing set. Cross-validation has a significant influence if we have a limited number of classification observations, such as the daily prediction in Bitcoin prices.

A major setback was the fact that we couldn't pair the Bitcoin who-trust-whom network with the transaction network. For that reason, the behavioral measures fairness and goodness are not used as a feature in the Random Forest. Further research can be done towards linking the used dataset together or finding another comparable who-trust-whom network to measure the impact of fairness and goodness on the Bitcoin price prediction.

With the use of transaction monitoring, money laundering can be detected inside a transaction network. Money laundering is the conversion of criminal income into assets that cannot be traced back to the underlying crime. With the use of machine learning the underlying structure of a transaction network can be found, which helps to identify potential money laundering activities or patterns. Since most financial institutions protect their transaction data, limited academical research has been devoted to real-life networks. As a future work of this research, the used methodologies can be applied to the confidential transaction networks hold by financial intuitions. In that way, it can be determined if these kinds of methods improve the research toward money laundering.

At last, only data prior to the Bitcoin Bubble was researched. In the Bitcoin Bubble the time-series, representing the Bitcoin price, showed significant large volatility, which causes predictions to be less reliable. As further research opportunities, there can be investigated how the classification model performs on Bitcoin data inside the Bitcoin Bubble.

References

[1] A. Amritkar, E. de Sturler, K. wirydowicz, D. Tafti,K. Ahuja. *Recycling Krylov subspaces for CFD applications and a new hybrid recycling solver*. Journal of Computational Physics, Volume 303, 2015, p. 222-237.

[2] E. Androulaki, G. Karame, M. Roeschlin, T. Scherer, and S. Capkun. *Evaluating User Privacy in Bitcoin.* In Proceedings of Financial Cryptography, 2013.

[3] P. Bannerjee. *UNODC estimates that criminals may have laundered US 1.6 trillion in 2009.* Public Information Officer: UNODC.

[4] A. Baumann, B. Fabian, M. Lischke. *Exploring the Bitcoin Network.* April 2014, doi 10.5220/0004937303690374

[5] L. Breiman. *Random Forests.* Machine Learning, 45(1): 532,2001. 18.

[6] M. Bressan, E. Peserico. *Choose the damping, choose the ranking?* Journal of Discrete Algorithms, Volume 8, Issue 2, June 2010, Pages 199-213.

[7] J. Brownlee. *What is the Difference Between Test and Validation Datasets?*. 2017.

[8] J. Chang, D.M. Blei. *Relational topic models for document networks.* In International Conference on Artificial Intelligence and Statistics, pages 8188, 2009.

[9] D. Coppersmith, S.J. Hong, J.R.M. Hosking. *Partitioning Nominal Attributes in Decision Trees.* Data Mining and Knowledge Discovery 3, 197217, 1999.

[10] T.M. Cover, J.A. Thomas. *Elements of Information Theory.* 1991, ISBN: 978-0-471-24195-9.

[11] N. Dershowitz, E.M. Reingold. *Calendrical Calculations.* Cambridge University Press. p. 289. ISBN: 978-0-521-70238-6.

[12] T. Dettmers. *Deep learning in a nutshell: Core concepts.* NVIDIA Devblogs, 2015.

[13] C. Eckart, G. Young. *The Approximation of One Matrix by Another of Lower Rank.* Psychometika, volume 1, p. 211218, 1936.

[14] G. Fagiolo. *Clustering in complex directed networks.* Phys. Rev. E 76, 026107, 2007.

[15] X. Fang, G. Xin, G. Havas. *On the worst-case complexity of integer Gaussian elimination.* Proceedings of the 1997 international symposium on Symbolic and algebraic computation. ACM. pp. 2831. 1997 ISBN: 0-89791-875-4.

[16] P. Fjallstrom. *Algorithms for Graph Partitioning: A Survey.* Computer and Information Science, 3(10), 1998.

[17] S. Fortunato. *Community Detection in Graphs.* Physics Reports, 486(35), p. 75174, 2010.

[18] D.A. Freedman. *Statistical Models: Theory and Practice.* Cambridge University Press. p. 26, 2009.

[19] L. Freeman. *A set of measures of centrality based on betweenness.* 1977. Sociometry. 40 (1): 3541. doi:10.2307/3033543.

[20] P.A. Gagniuc. *Markov Chains: From Theory to Implementation and Experimentation.* USA, NJ: John Wiley & Sons. pp. 1235. ISBN: 978-1-119-38755-8.

[21] A.S. Goldberger. *Classical Linear Regression.* Econometric Theory. New York: John Wiley & Sons. pp. 156212 [p. 158]. 1964. ISBN: 0-471-31101-4.

[22] A. Greaves and B. Au. *Using the bitcoin transaction graph to predict the price of bitcoin.* 2015.

[23] T. Guo and N. Antulov-Fantulin. *Predicting short-term bitcoin price fluctuations from buy and sell orders*, 2018.

[24] S. Haber, W.S. Stornetta. *How to time-stamp a digital document.* Journal of Cryptology, 3(2), 99111, 1991.

[25] T. Hastie, R. Tibshirani, J. Friedman. *The Elements of Statistical Learning.* Springer, 2008, ISBN: 0-387-95284-5.

[26] K. Hegazy, S. Mumford *Comparitive automated bitcoin trading strategies* 2016.

[27] B.W. Higgs, J. Weller, J.L. Solka. *Spectral embedding finds meaningful (relevant) structure in image and microarray data.* BMC Bioinformatics. 2006;7:74. doi: 10.1186/1471-2105-7-74.

[28] R. van der Hofstad. *Random Graphs and Complex Networks.* Department of Mathematics and Computer Science Eindhoven University of Technology P.O. Box 513 5600 MB Eindhoven, The Netherlands.l

[29] P. Hoff, A. Rafferty, M. Handcock. *Latent Space Approaches to Social Network Analysis.* Journal of the American Statistical Association, 97, 10901098, 2002.

[30] B. Hogan. *Visualizing and Interpreting Facebook Networks.* Analyzing Social Media Networks with NodeXL, 2011.

[31] M. Iansiti. K.R. Lakhani. *The Truth About BlockChain. Harvard Business Review.* 95(1), 118127. 2017.

[32] G. James, D. Witten, T. Hastie, R. Tibshirani. *An introduction to statistical learning.* Springer Science, New York 2013, ISBN: 978-1-4614-7138-7.

[33] G. James. *An Introduction to Statistical Learning: with Applications in R.* 2013, Springer. p. 176. ISBN: 978-1461471370.

[34] H. Jang and J. Lee. *An Empirical Study on Modeling and Prediction of Bitcoin Prices with Bayesian Neural Networks Based on BlockChain Information.* IEEE Access, vol. 6, pp. 54275437, 2017.

[35] M. Jiang, P. Cui, and C. Faloutsos. *Suspicious behavior detection: Current trends and future directions.* IEEE Intelligent Systems, 31(1):3139, 2016.

[36] N. Jindal and B. Liu. *Opinion Spam and Analysis.* Web Search and Data Mining, 2008, pp. 219230.

[37] B. Kamiski, M. Jakubczyk, P. Szufel. *A framework for sensitivity analysis of Decision Trees.* Central European Journal of Operations Research. 26 (1): 135159, 2017. doi:10.1007/s10100-017-0479-6.

[38] R.B. Kellogg, T.Y. Li, J. Yorke. *A constructive proof of the Brouwer fixed-point theorem and computational results.* SIAM Journal on Numerical Analysis, 1976.

[39] L. Khaidem, S. Saha, S.R. Dey. *Predicting the direction of stock market prices using Random Forest*, 2016.

[40] J. Kleinberg *Small-world phenomenon: an algorithmic perspective.* 2000.

[41] R. Kohavi and G. H. John. *Wrappers for feature subset selection.* Artificial intelligence, vol. 97, no. 1, pp. 273324, 1997.

[42] S. Kumar, F. Spezzano, V.S. Subrahmanian, C. Faloutsos. *Edge Weight Prediction in Weighted Signed Networks.* IEEE International Conference on Data Mining (ICDM), 2016.

[43] J. Leskovec, C. Faloutsos. *Sampling from large graphs.* In KDD '06: Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining, pages 631636, New York, NY, USA, 2006.

[44] Z. Liao, X. Lu, T. Yang, H. Wang. *Missing Data Imputation: A Fuzzy K-means Clustering Algorithm over Sliding Window.* 2009 Sixth International Conference on Fuzzy Systems and Knowledge Discovery.

[45] H Li, Z Chen, B Liu, X Wei. *Spotting Fake Reviews via Collective Positive-Unlabeled Learning.* Data Mining (ICDM), 2014. `ieeexplore.ieee.org`

[46] M. Luca, G. Zervas. *Fake It Till You Make It: Reputation, Competition, and Yelp Review Fraud.* Management Science, 2016. `pubsonline.informs.org`

[47] S.P. Lloyd. *Least square quantization in PCM.* Bell Telephone Laboratories Paper, 1982.

[48] I. Madan, S. Saluja, and A. Zhao. *Automated bitcoin trading via machine learning algorithms.* 2015.

[49] S. McNally, J. Roche, and S. Caton. *Predicting the Price of Bitcoin Using Machine Learning.* in Proceedings of the 2018 26th Euromicro International Conference on Parallel, Distributed and Network-based Processing (PDP), pp. 339343, Cambridge, March 2018.

[50] A. Mohr. *Quantum Computing in Complexity Theory and Theory of Computation.* p. 2. Retrieved 7 June 2014.

[51] S. Nakamoto. *Bitcoin: A peer-to-peer electronic cash system.* 2008.

[52] M.E.J. Newman. *A measure of betweenness centrality based on random walks.* Department of Physics and Center for the Study of Complex Systems, University of Michigan, Ann Arbor, MI 481091120.

[53] C. Peng, W. Xiao, P. Jian, Z. Wenwu. *A Survey on Network Embedding.* Department of Computer Science and Technology, Tsinghua University, China, 2017.

[54] B. Perozzi, R. Al-Rfou, S. Skiena. *Deepwalk: Online learning of social representations.* In Proceedings of the 20th ACM SIGKDD international conference on Knowledge discovery and data mining, p. 701710, 2014.

[55] T.N. Phyu. *Survey of Classification Techniques in Data Mining.* Proceedings of the International Multi Conference of Engineers and Computer Scientists 2009 Vol I IMECS March 18 - 20, 2009.

[56] L. Pretto. *A theoretical analysis of Google's PageRank.* Proceedings of the International Symposium on String Processing and Information Retrieval (SPIRE), 2002.

[57] J. Reunanen. *Overfitting in making comparisons between variable selection methods.* JMLR, 3: 13711382 (this issue), 2003.

[58] P. Reuter. *Chasing Dirty Money: The Fight Against Money Laundering.* Peterson Institute, 2005, ISBN: 0881325295.

[59] G Rudolph. *The fundamental matrix of the general random walk with absorbing boundaries*. 1999.

[60] S. Shalev-Shwartz, S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms.* ISBN: 978-1-107-05713-5

[61] M. Sipser. *Introduction to the Theory of Computation*. Course Technology Inc. ISBN 0-619-21764-2. 2016.

[62] S. Skiena. *Sorting and Searching". The Algorithm Design Manual*. Springer. p. 480, 2008. ISBN: 978-1-84800-069-8.

[63] A. Skrondal, S. Rabe-Hesketh . *Latent Variable Modelling: A Survey*. Board of the Foundation of the Scandinavian Journal of Statistics 2007. USA Vol 34: 712745, 2007.

[64] J. M. Steele. *The CauchySchwarz Master Class: an Introduction to the Art of Mathematical Inequalities*. The Mathematical Association of America. p. 1. ISBN 978-0521546775, 2014.

[65] D.L. Sussman, M. Tang, D.E. Fishkind, C.E. Priebe. *A Consistent Adjacency Spectral Embedding for Stochastic Blockmodel Graphs.* Journal of the American Statistical Association, V: 107, N: 499, pg: 1119-1128, year: 2012, Taylor and Francis.

[66] D.J. Watts. *The dynamics of networks between order and randomness.* Princeton Studies in Complexity. Princeton University Press, Princeton, NJ, (1999).

[67] D.K. Wind. *Concepts in predictive machine learning*. in Masters Thesis, 2014.

[68] S. Young, E. Scheinerman. *Random Dot Product Mod- els for Social Networks*. in Proceedings of the 5th International Conference on Algorithms and Models for the Web-Graph, pp. 138 149, 2007.

[69] E. Zivot, J. Wang *Rolling Analysis of Time Series. In: Modeling Financial Time Series with S-PLUS*. Springer, New York, NY,2006.

[70] `https://www.oreilly.com/library/view/practical-statistics-for/9781491952955/ch04.html`

[71] `https://medium.com/fintechexplained/part-3-regression-analysis-bcfe15a12866`

[72] `https://news.bitcoin.com/understanding-cryptocurrency-options-an-alternative-way-to-trade-crypto/`

[73] `https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html`

[74] `https://medium.com/@srnghn/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3`

[75] `https://www.buybitcoinworldwide.com/satoshi/to-bitcoin/`

[76] `https://github.com/Ashish7129/Graph_Sampling/blob/master/README.md`

[77] `http://snap.stanford.edu/data/soc-sign-bitcoin-otc.html`

[78] `https://cran.r-project.org/web/packages/jsonlite/index.html`

A Fairness and Goodness

In 2016, Kumar et al. introduced the concepts of "fairness" and "goodness" [42]. In their paper, they proposed two new measures of node behavior: fairness and goodness. The goodness of a node intuitively captures how much this node is liked/trusted by other nodes, while the fairness of a node captures how fair the node is in rating other nodes' likeability or trust level.

In detail, the fairness of a node is a measure of how fair or reliable the node is in assigning ratings (like/dislike, agree/disagree, trust/distrust) to other nodes. Intuitively, a 'fair' or 'reliable' rater should give a user the rating that it deserves, while an 'unfair' one would deviate from that value. Hence, the ratings assigned by unfair raters should be given low importance, while ratings given by fair raters should be considered significant. As an example, in real-world networks such as the Bitcoin networks, scammers create multiple accounts to increase their ratings and to reduce the ratings of good-natured users. To prevent unjust activities, these types of users should be given a low fairness score.

On the other hand, the goodness of a node specifies how much other nodes like/dislike, agree/disagree, or trust/distrust that node. Higher goodness implies the node is more trustworthy in the network. Hence, a 'good' or 'trustworthy' node would receive many high positive ratings from fair nodes, while a non-trustworthy user would receive high negative ratings from fair nodes.

A.1 Economic interpretation

In January 2016, Jiang, Cui, and Faloutsos published a study about current and future trends in suspicious behavior detection. They state that as the popularity of review platforms has grown, so have concerns towards the credibility of reviews. The quality of the reviews can be undermined by businesses leaving fake reviews for themselves or their competitors. In the research, they showed considerable evidence that this type of cheating is cultured in the industry [35][45]. The first comprehensive study on trustworthiness in online reviews investigated 5.8 million reviews and 2.1 million reviewers on Amazon. The study revealed by using duplicate spam reviews as positive training examples that at least 6% of all Amazon reviews can be seen as 'fake' [36].

Another example is a 2013 collaborative study of Boston University and Harvard Business School. They used two complementary approaches and datasets to commit review fraud on the popular review platform Yelp. A total of 16% suspicious reviews was identified. Also, there existed small significant variation across different platforms and even within a single platform, which showed the extent of suspicious activity. Due to the complex nature of review fraud, the estimation performed by this research is partially biased [46].

Users trading anonymous cryptocurrency, such as Bitcoin, rate others on their trustworthiness. Because buyers frequently look at ratings and reviews before trading Bitcoins, there is a substantial interest for fraudulent users to give fake ratings. As stated, fake reviewing has a significant impact on a social rating network and therefore, should be taken into account in a sort of measure. The fairness measure tries to distinguish unfair raters such that their rating is given lower importance, than ratings given by fair raters. Stock exchange prices often reflect the confidence that investors have in a stock, which implies that fairness and goodness can contribute in the price prediction of Bitcoin.

A.2 Fairness and Goodness in formula

Fairness and Goodness are calculated on weighted signed networks (WSN). WSN's are characterized by their property of capturing like/dislike, trust/distrust, and other social relationships between entities, mostly by their edges being either negatively or positively weighted [42]. Mathematically, a WSN is a weighted directed graph $G = (V, E, w)$ with V nodes, E edges and w as the weight function over the edges displaying the social relationship between the nodes.

Given a WSN, we don't know how fair and how good each node initially is. However, the fairness and goodness metric is dependent on each other, as described above. In their paper Kumar et al. state that both measures can be computed in a mutually recursive manner. And by showing that they satisfy two pre-defined axioms, the two measures are independent of each other [42]. These axioms are called the Fairness axiom and Goodness axiom, where the Goodness axiom establishes the independence of goodness on fairness, and the Fairness axiom established the independence of fairness on goodness (subsection A.3).

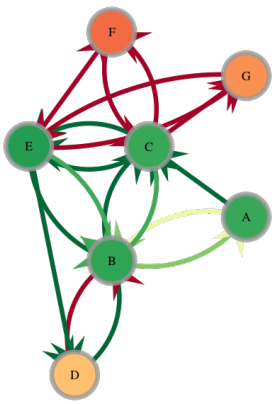
We now provide the two equations to compute the fairness and goodness scores in a mutually recursive manner as defined in the paper [42] on a WSN $G = (V, E, w)$.

$$f(u) = 1 - \frac{1}{|out(u)|} \sum_{v \in out(u)} \frac{|w(u,v) - g(v)|}{R}, \tag{7}$$

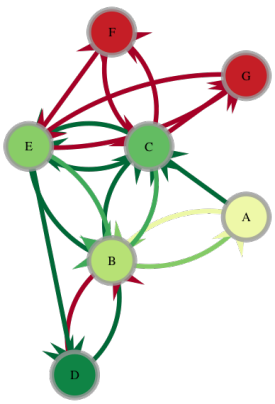
$$g(v) = \frac{1}{|in(v)|} \sum_{u \in in(v)} f(u)w(u,v). \tag{8}$$

$f(u)$ is the fairness score of a node $u \in V$, $g(v)$ the goodness score of a node $v \in V$ and $in(v), out(v)$ the in/out-degree of a node $v \in V$. Fairness scores always lie in the $[0, 1]$ interval and goodness scores lie in a $[-\frac{R}{2}, \frac{R}{2}]$ interval. The maximum possible difference goodness score is the range of differences R, with $R \in \mathbb{R}^+ \backslash \{0\}$. Further on, we will see that the choice of R depends on the weight function w, to preserve convergence of the measure (Theorem A.1). The proof that both scores are present in the described intervals is given in Proposition A.6.

Non-mathematical; the fairness formula (Equation 7) says the smaller the difference between the actual edge weight and the goodness of the recipient, the fairer the node. Afterward, the average for all the ratings given by node u is used to calculate the fairness of u. When calculating goodness (Equation 8), the incoming edge weights are weighted by the fairness of the nodes that are rating it, so that ratings by fair nodes are considered essential. Again, the average of these products overall predecessors yields the goodness of v.



(a) Fairness score.



(b) Goodness score.

n	$f(n)$	$g(n)$
A	0,86	0,09
B	0,87	0,32
C	0,86	0,60
D	0,36	0,88
E	0,89	0,49
F	0,20	-0,87
G	0,25	-0,87

(c) Fairness and goodness scores for each node.

Figure 41: An example network to explain fairness and goodness, with $R = 2$. The colors on the edges represent their weight on a scale from red to green, where red is the lowest score -1 and green the highest score 1. (a) Fairness measure. (b) Goodness measure. (c) Fairness and goodness scores for each node in the example network. To calculate the scores the FGA (Algorithm 6) is used, which will be explained in the upcoming sections.

In Figure 41 a small weighted signed network is illustrated. The weights of the network are between $[-1.1]$, which are represented in color scale from red (lowest) to green (highest). In Figure 41a, the fairness scores are illustrated. It is seen that nodes A,B,C, and E are fair nodes, as their ratings are close to the other ratings that the recipients get. This makes their ratings close to the goodness score of the node and this increases their fairness. Nodes D, F, and G have lower fairness scores, as their ratings deviate a lot from the goodness scores of their recipients. Figure 41b illustrates the goodness scores, it is seen that nodes A, B, C, D, and E are good. Node A is low positive as it only receives one edge which is low positive. On the other hand, B is low positive as well, as it receives edges varying edge weights, most of which are positive. nodes F and G have negative goodness ratings, as they only got negative ratings from very fair nodes.

This example very well reflects the behavior of the fairness and goodness measures. node B is interesting as it is good but not fair. nodes F and G are most likely to be fraudsters because they have low fairness and goodness scores. In Table of Figure 41c all fairness and goodness scores are listed.

A.3 Independence

Since Fairness and Goodness are calculated through a mutually recursive manner, it's not a given that both measures are independent of each other. In this section, two axioms are introduced to prove its independence, called the Fairness axiom and Goodness axiom. The Goodness axiom establishes the dependence of goodness on fairness, and the Fairness axiom established the dependence of fairness on goodness.

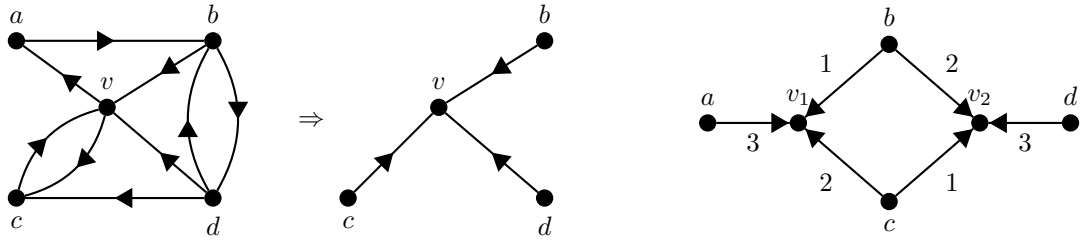
Before the axioms of Fairness and Goodness are discussed, we need to introduce ego-networks.

Definition A.1. Given a WSN $G = (V, E, w)$ and a node $u \in V$, we define the **ego-in-network** of u as

$$ego_{in}(u) = (V_u, E_u, w_u),$$

where:

- $V_u = \{u\} \cup in(u)$
- $E_u = \{(v, u) | v \in in(u)\}$
- $w_u(v, u) = w(v, u)$, $\forall (v, u) \in E_u$



(a) The ego-in-network of v.

(b) Two identical ego-in-networks of v_1 and v_2.

Figure 42: (a) The ego-in-network $ego_{in}(v)$ of node v. (b) Nodes v_1 and v_2 have identical ego-in-networks. Set $in(v_1) = \{a, b, c\}$ and $in(v_2) = \{b, c, d\}$ as the sets of predecessors of nodes v_1 and v_2. Clearly, $|in(v_1)| = |in(v_2)| = 3$. Define the one-to-one mapping $h : in(v_1) \to in(v_2)$ such that $h(a) = d$, $h(b) = c$ and $h(c) = b$. By definition A.2 v_1 and v_2 have identical ego-in-networks, since $w_{v_1}(u, v_1) = w_{v_2}(h(u), v_2)$, $\forall u \in in(v_1)$.

Definition A.2. Given a WSN $G = (V, E, w)$. Two nodes $v_1, v_2 \in V$ have **identical** ego-in-networks if $|in(v_1)| = |in(v_2)|$ and there exists a one-to-one mapping $h : in(v_1) \to in(v_2)$ such that $w_{v_1}(u, v_1) = w_{v_2}(h(u), v_2)$, $\forall u \in in(v_1)$.

Similarly, we can define the concept of a **ego-out-network** of $u \in V$, denoted by $ego_{out}(u)$, and two nodes having identical ego-out-networks. In Figure 42 an example of a ego-in-network and two identical ego-in-networks are visualized.

Axiom A.3. *(Fairness Axiom)*
Given a WSN $G = (V, E, w)$, let $v_1, v_2 \in V$ be two nodes having identical ego-out-networks and let h be the one-to-one mapping between the two ego-out-networks. A node is more fair than another if it gives ratings closer to the rating deserved by the recipient, i.e. $\forall k \in out(v_1)$,

$$|w_{v_1}(v_1, k) - g(k)| \leq |w_{v_2}(v_2, h(k)) - g(h(k))| \Rightarrow f(v_1) \geq f(v_2)$$

Axiom A.4. *(Goodness Axiom)*
Given a WSN $G = (V, E, w)$, let $v_1, v_2 \in V$ be two nodes having identical ego-in-networks and let h be the one-to-one mapping between the two ego-in-networks. nodes with higher fairness have higher impact on the nodes they rate and nodes with lower fairness have lower impact on the nodes they rate, i.e.

$$f(v) = f(h(v)) \forall v \in in^-(v_1) \text{ and } f(v) \geq f(h(v)) \forall v \in in^+(v_1) \Rightarrow g(v_1) \geq g(v_2).$$

Conversely,

$$f(v) = f(h(v)) \forall v \in in^+(v_1) \text{ and } f(v) \geq f(h(v)) \forall v \in in^-(v_1) \Rightarrow g(v_1) \leq g(v_2).$$

Here $in^+(v)$ and $in^-(v)$ are the sets of positive and negative incoming edges of a node $v \in V$.

Proposition A.5. *The defined fairness (Equation 7) and goodness (Equation 8) measures satisfy both the fairness- and goodness- axioms.*

Proof. (*Fairness Axiom*) Given a WSN $G = (V, E, w)$, assume $v_1, v_2 \in V$ are two nodes with identical ego-out-networks. By the definition, $|out(v_1)| = |out(v_2)|$ and $\exists\ h : out(v_2) \to out(v_1)$ which is a one-to-one mapping, such that $w_{v_2}(v_2, u) = w_{v_1}(v_1, h(u))$, $\forall u \in out(v_2)$.

Assume further that $\forall k \in out(v_1)$:

$$|w_{v_1}(v_1, k) - g(k)| \leq |w_{v_2}(v_2, h(k)) - g(h(k))|.$$

From the definition of fairness (Equation 7), we state:

$$\begin{aligned} f(v_1) - f(v_2) &= -\frac{1}{|out(v_1)|} \sum_{u \in out(v_1)} \frac{|w(v_1, u) - g(u)|}{R} + \frac{1}{|out(v_2)|} \sum_{u \in out(v_2)} \frac{|w(v_2, u) - g(u)|}{R} \\ &= \frac{1}{|out(v_1)|R} \left(\sum_{u \in out(v_2)} |w_{v_2}(v_2, u) - g(u)| - \sum_{u \in out(v_1)} |w_{v_1}(v_1, u) - g(u)| \right). \end{aligned}$$

From the definition of the one-to-one mapping h,

$$\sum_{u \in out(v_2)} |w_{v_2}(v_2, u) - g(u)| = \sum_{u \in out(v_1)} |w_{v_2}(v_2, h(u)) - g(h(u))|.$$

Hence,

$$\begin{aligned}(f(v_1)-f(v_2))\,|out(v_1)|R &= \sum_{u\in out(v_2)} |w_{v_2}(v_2,u)-g(u)| - \sum_{u\in out(v_1)} |w_{v_1}(v_1,u)-g(u)| \\ &= \sum_{u\in out(v_1)} \underbrace{|w_{v_2}(v_2,h(u))-g(h(u))| - |w_{v_1}(v_1,u)-g(u)|}_{\text{by assumption } \geq 0} \\ &\Rightarrow (f(v_1)-f(v_2))\,|out(v_1)|R \geq 0 \\ &\Rightarrow f(v_1)\geq f(v_2).\end{aligned}$$

□

Proof. (*Goodness Axiom*) Given a WSN $G=(V,E,w)$, assume $v_1,v_2\in V$ are two nodes with identical ego-in-networks. From the definition, $|in(v_1)|=|in(v_2)|$ and $\exists\ h: in(v_2)\to in(v_1)$, which is a one-to-one mapping, such that $w_{v_2}(u,v_2)=w_{v_1}(h(u),v_1)$, $\forall u\in in(v_2)$.

From the definition of goodness (Equation 8), we state:

$$\begin{aligned}g(v_1)-g(v_2) =& \frac{1}{|in(v_1)|}\sum_{u\in in(v_1)} f(u)w(u,v_1) - \frac{1}{|in(v_2)|}\sum_{u\in in(v_2)} f(u)w(u,v_2) \\ =& \frac{1}{|in(v_1)|}\left(\sum_{u\in in(v_1)} f(u)w_{v_1}(u,v_1) - \sum_{u\in in(v_2)} f(u)w_{v_2}(u,v_2)\right).\end{aligned}$$

From the definition of the one-to-one mapping h,

$$\sum_{u\in in(v_2)} f(u)w_{v_2}(u,v_2) = \sum_{u\in in(v_2)} f(u)w_{v_1}(h(u),v_1) = \sum_{u\in in(v_1)} f(h(u))w_{v_1}(u,v_1).$$

Hence,

$$(g(v_1)-g(v_2))\,|in(v_1)| = \sum_{u\in in(v_1)} [f(u)-f(h(u))]\,w_{v_1}(u,v_1)).$$

Remember that $in(v_1)$ represents the set of nodes that preceed node v_1. This set can be divided in two disjoint sets of positive/negative predecessors, such that $in(v_1)=in^+(v_1)\cup in^-(v_1)$ and $in^+(v_1)\cap in^-(v_1)=\emptyset$.

Now assume that $f(v)=f(h(v))\forall v\in in^-(v_1)$ and $f(v)\geq f(h(v))\forall v\in in^+(v_1)$. Remember that the sum of two disjoint subsets can be divided, such that:

$$\begin{aligned}(g(v_1)-g(v_2))\,|in(v_1)| &= \sum_{u\in in^+(v_1)} \underbrace{[f(u)-f(h(u))]}_{\geq 0}\underbrace{w_{v_1}(u,v_1))}_{\geq 0} + \sum_{u\in in^-(v_1)} \underbrace{[f(u)-f(h(u))]}_{=0} w_{v_1}(u,v_1)) \\ &\Rightarrow (g(v_1)-g(v_2))\,|in(v_1)|\geq 0 \\ &\Rightarrow g(v_1)\geq g(v_2).\end{aligned}$$

Subsequently, if we assume that $f(v)=f(h(v))\forall v\in in^+(v_1)$ and $f(v)\geq f(h(v))\forall v\in in^-(v_1)$, it will imply:

$$\begin{aligned}(g(v_1)-g(v_2))\,|in(v_1)| &= \sum_{u\in in^+(v_1)} \underbrace{[f(u)-f(h(u))]}_{=0} w_{v_1}(u,v_1)) + \sum_{u\in in^-(v_1)} \underbrace{[f(u)-f(h(u))]}_{\geq 0}\underbrace{w_{v_1}(u,v_1))}_{\leq 0} \\ &\Rightarrow (g(v_1)-g(v_2))\,|in(v_1)|\leq 0 \\ &\Rightarrow g(v_1)\leq g(v_2).\end{aligned}$$

□

A.4 Algorithm

In the upcoming section, the algorithm to compute fairness and goodness scores for each node in a WSN is presented. First, we present the algorithm and show that its iterations are compact interval bounded. Second, we prove its convergence, uniqueness and linear time complexity. Third, the argumentation of the starting values and stopping criterion is explained and numerically tested. At last, we look at the accuracy of the algorithm under the chosen starting values and stopping criterion.

Outline Algorithm. For calculation of the fairness and goodness score an iterative method is used. In computational mathematics, an iterative method is a mathematical procedure that uses an initial guess to generate a sequence of improving approximate solutions for a class of problems, in which the n-th approximation is derived from the previous ones [1].

Algorithm 6 Fairness and Goodness Algorithm

Require: A WSN $G = (V, E, w)$, a stopping criterion $\epsilon \in \mathbb{R}^+ \backslash \{0\}$ and starting values $f^0, g^0 \in \mathbb{R}^{|V|}$.
Ensure: Fairness and Goodness scores for all nodes in V

1: Define $R = 2 \max_{e \in E} |w(e)|$
2: Set $t = -1$
3: **do**
4: $\quad t = t + 1$
5: $\quad g^{t+1}(v) = \frac{1}{|in(v)|} \sum_{u \in in(v)} f^t(u) w(u, v), \forall v \in V$
6: $\quad f^{t+1}(u) = 1 - \frac{1}{R|out(u)|} \sum_{v \in out(v)} |w(u, v) - g^{t+1}(v)|, \forall u \in V$
7: **while** $\sum_{u \in V} |f^{t+1}(u) - f^t(u)| > \epsilon$ or $\sum_{u \in V} |g^{t+1}(u) - g^t(u)| > \epsilon$
8:
9: **return** $f^{t+1}(u)$ and $g^{t+1}(u), \forall u \in V$

Algorithm 6 describes the Fairness and Goodness Algorithm, in short FGA. The Algorithm requires a WSN, a stopping criterion and a starting value for both the fairness and goodness scores. In subsection A.4.3 the choice of the stopping criterion and starting values are argued. The Python code of Algorithm 6 is found in Appendix B.7.

The algorithm initially starts with definition of R, which represents the maximum possible difference goodness score. More precise, R is twice the maximum absolute weight of the required WSN, and is defined as $R = 2 \max_{e \in E} |w(e)|$ (line 1). In line 7 and 8, the goodness and fairness scores are updated according to equations (7) and (8). Both fairness and goodness scores are mutually recursive and are updated until the difference between two iterations is below the stopping criterion ϵ (line 9). The algorithm returns the fairness and goodness scores from the last iteration, which represent the final scores (line 11).

Compact interval bounded. Some iterative methods are compact interval bounded for all iterations. Being bounded on a compact interval can ensure a fixed-point under continuous functions [38]. Our iterative functions are non-continuous due to network properties, however being bounded on a certain interval will support the convergence of the algorithm (subsection A.4.1).

Proposition A.6. *Given a WSN $G = (V, E, w)$ as the requirement of Algorithm 6, such that $R = 2 \max_{e \in E} |w(e)|$. $\forall u \in V$, the fairness and goodness iterations of the algorithm are compact interval bounded, such that:*

$$f^t(u) \in [0, 1] \wedge g^t(u) \in [-\tfrac{R}{2}, \tfrac{R}{2}] \Rightarrow f^{t+1}(u) \in [0, 1] \wedge g^{t+1}(u) \in [-\tfrac{R}{2}, \tfrac{R}{2}]$$

Proof. Assume the left side of the relation, i.e. $\forall u \in V$ and a single $t \in \mathbb{N}$, $f^t(u) \in [0, 1]$ and $g^t(u) \in [-\frac{R}{2}, \frac{R}{2}]$. Since $R = 2 \max_{e \in E} |w(e)|$ we can derive that $w(e) \in [-\frac{R}{2}, \frac{R}{2}], \forall e \in E$. From both

the definitions of the fairness and goodness iterations it can be derived that;

$$g^{t+1}(u) = \frac{1}{|in(v)|} \sum_{u \in in(v)} \underbrace{f^t(u)}_{\in[0,1]} \underbrace{w(u,v)}_{\in[-R/2,R/2]} = \frac{1}{|in(v)|} \sum_{u \in in(v)} \underbrace{f^t(u)w(u,v)}_{\in[-R/2,R/2]} \overset{\diamond}{\Rightarrow} g^{t+1}(u) \in [-\tfrac{R}{2}, \tfrac{R}{2}].$$

In the above equation, there is taken into account that; the sum of n values in a fixed interval divided by n results the value being in that fixed interval. This is used in the step denoted by the $\diamond$ symbol. For the fairness iteration a similar calculation can be made;

$$f^{t+1}(u) = 1 - \frac{1}{R|out(u)|} \sum_{v \in out(v)} | \underbrace{w(u,v)}_{\in[-R/2,R/2]} - \underbrace{g^{t+1}(v)}_{\in[-R/2,R/2]} | = 1 - \frac{1}{R|out(u)|} \sum_{v \in out(v)} \underbrace{|w(u,v) - g^{t+1}(v)|}_{\in[0,R]}$$

$$\overset{1}{=} 1 - \frac{1}{R} \underbrace{\frac{1}{|out(u)|} \sum_{v \in out(v)} |w(u,v) - g^{t+1}(v)|}_{\in[0,R]} = 1 - \underbrace{\frac{1}{R|out(u)|} \sum_{v \in out(v)} |w(u,v) - g^{t+1}(v)|}_{\in[0,1]}$$

$$\Rightarrow f^{t+1}(u) \in [0,1]$$

□

A.4.1 Convergence and Uniqueness

An iterative algorithm is said to converge when, as the iterations proceed, the output gets closer and closer to a specific value. More precisely, the absolute error between two iterations becomes smaller than the stopping criterion ϵ.

In Proposition A.6 it is shown that the iterations of both scores are bounded on a certain interval. Choosing a starting value within these bounded intervals, assures all iterations to be bounded. In addition, our Fairness and Goodness Algorithm defines the variable $R = 2\max_{e \in E} |w(e)|$, this assumption is necessary in providing convergence for the algorithm. In the upcoming theorems we will show that the algorithm will converge and that the solution is unique upon convergence.

Theorem A.1. ***(Convergence Theorem)*** *Let $f^t(u)$ be the fairness and $g^t(u)$ the goodness, of node u after iteration t of the FGA (Algorithm 6). Let $f^\infty(u)$ be the final fairness score and $g^\infty(u)$ the final goodness score. Then, the error is bounded by $|f^\infty(u) - f^t(u)| \leq \frac{1}{2^t}$, $\forall u \in V$. As consequence, if t increases, $f^t(u)$ converges to $f^\infty(u)$. Similarly, $|g^\infty(u) - g^t(u)| \leq \frac{R}{2^t}$, $\forall u \in V$.*

Proof. (Fairness Convergence) This theorem is proven by mathematical induction for $t = 1, 2, \ldots$ We use the definitions of the fairness and goodness iterations defined as in Algorithm 6. We need to show that $\forall t = 1, 2, \ldots$ and $\forall u \in V$:

$$|f^\infty(u) - f^t(u)| \leq \frac{1}{2^t}.$$

Starting iteration $\mathbf{t = 1}$.

$$
\begin{aligned}
&|f^{\infty}(u) - f^{1}(u)| \\
&= \left| \left(\frac{1}{R|out(u)|} \sum_{v \in out(v)} |w(u,v) - g^{1}(v)| \right) - \left(\frac{1}{R|out(u)|} \sum_{v \in out(v)} |w(u,v) - g^{\infty}(v)| \right) \right| \\
&= \left| \frac{1}{R|out(u)|} \sum_{v \in out(v)} \left(\left| w(u,v) - \sum_{u \in in(v)} \frac{f^{0}(u) w(u,v)}{|in(v)|} \right| - \left| w(u,v) - \sum_{u \in in(v)} \frac{f^{\infty}(u) w(u,v)}{|in(v)|} \right| \right) \right| \\
&\overset{\diamond}{\leq} \frac{1}{R|out(u)|} \sum_{v \in out(v)} \left| \left| w(u,v) - \sum_{u \in in(v)} \frac{f^{0}(u) w(u,v)}{|in(v)|} \right| - \left| w(u,v) - \sum_{u \in in(v)} \frac{f^{\infty}(u) w(u,v)}{|in(v)|} \right| \right| \\
&\overset{\bullet}{\leq} \frac{1}{R|out(u)|} \sum_{v \in out(v)} \left| w(u,v) - \sum_{u \in in(v)} \frac{f^{0}(u) w(u,v)}{|in(v)|} - w(u,v) + \sum_{u \in in(v)} \frac{f^{\infty}(u) w(u,v)}{|in(v)|} \right| \\
&= \frac{1}{R|out(u)|} \sum_{v \in out(v)} \left| \sum_{u \in in(v)} \frac{w(u,v)(f^{\infty}(u) - f^{0}(u))}{|in(v)|} \right| \\
&\overset{\diamond}{\leq} \frac{1}{R|out(u)|} \sum_{v \in out(v)} \sum_{u \in in(v)} \frac{|w(u,v)(f^{\infty}(u) - f^{0}(u))|}{|in(v)|}
\end{aligned}
$$

In the above inequalities two different types of triangle inequalities are used: The most traditional one, where $|x + y| \leq |x| + |y|$, used in the steps characterized by the $\diamond$ symbol. And the reverse triangle inequality, where $||x| - |y|| \leq |x - y|$, $\forall x, y, z \in \mathbb{R}$, used in the steps characterized by the $\bullet$ symbol. Since a third triangle inequality states $|x \cdot y| \leq |x||y|$, $\forall x, y \in \mathbb{R}$, we have;

$$|w(u,v)(f^{\infty}(u) - f^{0}(u))| \leq |w(u,v)||f^{\infty}(u) - f^{0}(u)|.$$

In the FGA we assumed $R = 2 \max_{e \in E} |w(e)|$, which gives us $|w(u,v)| \leq \frac{R}{2}$. Also, all our iterations are bounded such that $f^{t}(u) \in [0,1]$ ($\forall t \in \mathbb{N}$), which gives us $|(f^{\infty}(u) - f^{0}(u))| \leq 1$. Hence, $|w(u,v)||f^{\infty}(u) - f^{0}(u)| \leq \frac{R}{2}$. Thus,

$$
\begin{aligned}
|f^{\infty}(u) - f^{1}(u)| &\leq \frac{1}{R|out(u)|} \sum_{v \in out(v)} \sum_{u \in in(v)} \frac{|w(u,v)(f^{\infty}(u) - f^{0}(u))|}{|in(v)|} \\
&\leq \frac{1}{R|out(u)|} \sum_{v \in out(v)} \sum_{u \in in(v)} \frac{R}{2|in(v)|} = \frac{1}{R|out(u)|} \sum_{v \in out(v)} \frac{R}{2} = \frac{R}{2R} = \frac{1}{2}.
\end{aligned}
$$

$\forall \mathbf{t = t + 1}$. Let's assume that $|f^{\infty}(u) - f^{t}(u)| \leq \frac{1}{2^{t}}$. Then,

$$
\begin{aligned}
|f^{\infty}(u) - f^{t+1}(u)| &\leq \frac{1}{R|out(u)|} \sum_{v \in out(v)} \sum_{u \in in(v)} \frac{|w(u,v)||f^{\infty}(u) - f^{t}(u)|}{|in(v)|} \\
&\leq \frac{1}{R|out(u)|} \sum_{v \in out(v)} \sum_{u \in in(v)} \frac{|w(u,v)|}{|in(v)|2^{t}} \leq \frac{1}{R|out(u)|} \sum_{v \in out(v)} \sum_{u \in in(v)} \frac{R}{|in(v)|2^{t+1}} \\
&= \frac{1}{R} \cdot \frac{R}{2^{t+1}} = \frac{1}{2^{t+1}},
\end{aligned}
$$

which proves the induction. Assume $t \to \infty$, then $\forall u \in V$:

$$\frac{1}{2^{t}} \to 0 \Rightarrow |f^{\infty}(u) - f^{t}(u)| \to 0 \Rightarrow f^{t}(u) \to f^{\infty}(u).$$

□

Proof. (Goodness Convergence) This theorem is also proven by mathematical induction for $t = 1, 2, \ldots$. We use the definitions of the fairness and goodness iterations defined as in Algorithm 6. We need to show that $\forall t = 1, 2, \cdots \in \mathbb{N}$ and $\forall u \in V$:

$$|g^{\infty}(u) - g^{t}(u)| \leq \frac{R}{2^t}.$$

Starting iteration $\mathbf{t = 1}$.

$$\begin{aligned}
&|g^{\infty}(u) - g^{1}(u)| \\
&= \left| \frac{1}{|in(v)|} \sum_{u \in in(v)} f^{\infty}(u) w(u,v) - \frac{1}{|in(v)|} \sum_{u \in in(v)} f^{0}(u) w(u,v) \right| \\
&= \left| \frac{1}{|in(v)|} \sum_{u \in in(v)} \left(\frac{w(u,v)}{R|out(u)|} \sum_{v \in out(v)} |w(u,v) - g^{0}(v)| - \frac{w(u,v)}{R|out(u)|} \sum_{v \in out(v)} |w(u,v) - g^{\infty}(v)| \right) \right| \\
&= \left| \frac{1}{|in(v)|} \sum_{u \in in(v)} \left(\frac{w(u,v)}{R|out(u)|} \sum_{v \in out(v)} |w(u,v) - g^{0}(v)| - |w(u,v) - g^{\infty}(v)| \right) \right| \\
&\overset{\diamond}{\leq} \frac{1}{|in(v)|} \sum_{u \in in(v)} \left(\frac{|w(u,v)|}{R|out(u)|} \sum_{v \in out(v)} \big| |w(u,v) - g^{0}(v)| - |w(u,v) - g^{\infty}(v)| \big| \right) \\
&\overset{\bullet}{\leq} \frac{1}{|in(v)|} \sum_{u \in in(v)} \left(\frac{1}{R|out(u)|} \sum_{v \in out(v)} |w(u,v)| \left| g^{\infty}(v) - g^{0}(v) \right| \right).
\end{aligned}$$

Similar inequalities, as in the previous proof, are used and characterized by similar symbols $\diamond, \bullet$. In the FGA we assumed $R = 2\max_{e \in E} |w(e)|$, which gives us $|w(u,v)| \leq \frac{R}{2}$. Also, all our iterations are bounded such that $g^{t}(u) \in [-\frac{R}{2}, \frac{R}{2}]$ $(\forall t \in \mathbb{N})$, which gives us $|(g^{\infty}(u) - g^{0}(u))| \leq R$. Hence, $|w(u,v)||g^{\infty}(u) - g^{0}(u)| \leq \frac{R^2}{2}$. Thus,

$$|g^{\infty}(u) - g^{1}(u)| \leq \frac{1}{|in(v)|} \sum_{u \in in(v)} \left(\frac{1}{R|out(u)|} \sum_{v \in out(v)} \frac{R^2}{2} \right) = \frac{1}{|in(v)|} \sum_{u \in in(v)} \frac{R}{2} = \frac{R}{2}.$$

$\forall \mathbf{t = t + 1}$. Let's assume that $|g^{\infty}(u) - g^{t}(u)| \leq \frac{R}{2^t}$. Then,

$$\begin{aligned}
|g^{\infty}(u) - g^{t+1}(u)| &\leq \frac{1}{|in(v)|} \sum_{u \in in(v)} \left(\frac{1}{R|out(u)|} \sum_{v \in out(v)} |w(u,v)| \left| g^{\infty}(v) - g^{t}(v) \right| \right) \\
&\leq \frac{1}{|in(v)|} \sum_{u \in in(v)} \left(\frac{1}{R|out(u)|} \sum_{v \in out(v)} \frac{|w(u,v)|R}{2^t} \right) \\
&\leq \frac{1}{|in(v)|} \sum_{u \in in(v)} \left(\frac{1}{R|out(u)|} \sum_{v \in out(v)} \frac{R^2}{2^{t+1}} \right) = \frac{1}{|in(v)|} \sum_{u \in in(v)} \frac{R}{2^{t+1}} = \frac{R}{2^{t+1}},
\end{aligned}$$

which proves the induction. Assume $t \to \infty$, then $\forall u \in V$;

$$\frac{R}{2^t} \to 0 \Rightarrow |g^{\infty}(u) - g^{t}(u)| \to 0 \Rightarrow g^{t}(u) \to g^{\infty}(u).$$

□

Theorem A.2. ***(Uniqueness Theorem)*** *FGA (Algorithm 6) produces a unique solution of fairness and goodness upon convergence.*

Proof. If fairness scores are unique, and since goodness scores will be fixed for fixed fairness scores, it is sufficient to only prove that fairness scores are unique in order to prove both. Let's assume the fairness score is not unique for a node $u \in V$, then $\exists f_1^\infty(u), f_2^\infty(u) \in [0,1]$ such that

$$D = |f_1^\infty(u) - f_2^\infty(u)|, \text{ with } D \geq 0 \in \mathbb{R}.$$

Then it follows that:

$$\begin{aligned}
D &= |f_1^\infty(u) - f_2^\infty(u)| \\
&= \left| \left(\frac{1}{R|out(u)|} \sum_{v \in out(v)} |w(u,v) - g_2^\infty(v)| \right) - \left(\frac{1}{R|out(u)|} \sum_{v \in out(v)} |w(u,v) - g_1^\infty(v)| \right) \right| \\
&= \left| \frac{1}{R|out(u)|} \sum_{v \in out(v)} \left(\left| w(u,v) - \sum_{u \in in(v)} \frac{f_2^\infty(u) w(u,v)}{|in(v)|} \right| - \left| w(u,v) - \sum_{u \in in(v)} \frac{f_1^\infty(u) w(u,v)}{|in(v)|} \right| \right) \right| \\
&\overset{\leq}{\diamond} \frac{1}{R|out(u)|} \sum_{v \in out(v)} \left| \left| w(u,v) - \sum_{u \in in(v)} \frac{f_2^\infty(u) w(u,v)}{|in(v)|} \right| - \left| w(u,v) - \sum_{u \in in(v)} \frac{f_1^\infty(u) w(u,v)}{|in(v)|} \right| \right| \\
&\overset{\leq}{\bullet} \frac{1}{R|out(u)|} \sum_{v \in out(v)} \left| w(u,v) - \sum_{u \in in(v)} \frac{f_2^\infty(u) w(u,v)}{|in(v)|} - w(u,v) + \sum_{u \in in(v)} \frac{f_1^\infty(u) w(u,v)}{|in(v)|} \right| \\
&= \frac{1}{R|out(u)|} \sum_{v \in out(v)} \left| \sum_{u \in in(v)} \frac{w(u,v)(f_1^\infty(u) - f_2^\infty(u))}{|in(v)|} \right| \\
&\overset{\leq}{\diamond} \frac{1}{R|out(u)|} \sum_{v \in out(v)} \sum_{u \in in(v)} \frac{|w(u,v)(f_1^\infty(u) - f_2^\infty(u))|}{|in(v)|} \\
&\overset{\leq}{*} \frac{1}{R|out(u)|} \sum_{v \in out(v)} \sum_{u \in in(v)} \frac{|w(u,v)||f_1^\infty(u) - f_2^\infty(u)|}{|in(v)|}.
\end{aligned}$$

Similar inequalities, as in the previous proof, are used and characterized by similar symbols $\diamond, \bullet$. In addition, a multiplication inequality is introduced, in which $|x \cdot y| \leq |x||y|$, $\forall x, y, z \in \mathbb{R}$ (characterized by $*$). In the FGA we assumed $R = 2 \max_{e \in E} |w(e)|$, which gives us $|w(u,v)| \leq \frac{R}{2}$. Also, at beginning of the proof, we assumed that $|f_1^\infty(u) - f_2^\infty(u)| = D$. Hence, $|w(u,v)||f_1^\infty(u) - f_2^\infty(u)| \leq \frac{RD}{2}$. Thus,

$$\begin{aligned}
D &\leq \frac{1}{R|out(u)|} \sum_{v \in out(v)} \sum_{u \in in(v)} \frac{|w(u,v)||f_1^\infty(u) - f_2^\infty(u)|}{|in(v)|} \\
&\leq \frac{1}{R|out(u)|} \sum_{v \in out(v)} \sum_{u \in in(v)} \frac{RD}{2|in(v)|} = \frac{1}{R|out(u)|} \sum_{v \in out(v)} \frac{RD}{2} = \frac{D}{2}.
\end{aligned}$$

Since $D \leq D/2$, with $D \geq 0$, this implies $D = 0$ we get $|f_1^\infty(u) - f_2^\infty(u)| = 0$, which makes the fairness score unique. □

A.4.2 Time complexity

Time complexity quantifies the amount of time taken by an algorithm to run as a function of the length of the input [61]. Time complexity is important as it can make a big difference as to whether the algorithm is practical for large WSNs. For example assume our algorithm has a time complexity depending on a quadratic factor of the number of nodes. If the number of nodes grows, the computation time grows at least quadratic, making it impractical to compute if the number of nodes becomes too large.

For determining the time complexity we use the 'big O notation' $\mathcal{O}$, which is the upper limit of the computation time growth [50]. The $\mathcal{O}$-notation often depends on the input size of the algorithm, for example $\mathcal{O}(n)$ means the algorithm performs in linear time with respect to input value n (worst case). To compute $\mathcal{O}$-notation the lower order terms are ignored, since the lower order terms are relatively insignificant for large input. As seen in Algorithm 6, during each iteration all in/out-going edges of all nodes are revised, once for the goodness score (line 5) and once for the fairness score (line 6). Assume that the algorithm requires a WSN $G = (V, E, w)$ and that i is the number of the current iteration, it follows that;

$$\mathcal{O}(f(t), g(i)) = \mathcal{O}(2i|E|) = \mathcal{O}(i|E|).$$

In Theorem A.1 we proved the convergence of the algorithm, so $\exists n \in \mathbb{N}$ where the algorithm terminates. The upper limit of computation time is linearly depended on the number of iterations n and the number of edges $|E|$. Using the rate of convergence we can show that n is always significant smaller than $|E|$ and conclude that $\mathcal{O}(f(n), g(n)) = \mathcal{O}(|E|)$, if $|E|$ becomes large.

Definition A.7. A sequence $(x_t)_{t\in\mathbb{N}}$ is said to **converge linearly** to $L \in \mathbb{R}$, if there exists a number $\mu \in (0, 1)$ such that

$$\lim_{t\to\infty} \frac{|x_{t+1} - L|}{|x_t - L|} = \mu.$$

The number μ is called the **rate of convergence**.

The drawback of the above definition is that it does not catch sequences which converge reasonably fast, and whose rate of convergence is variable. Our iterative process is an example of that. All nodes have individual rate of convergence, depending on the number of in-/out- going edges, Figure 43b illustrates that. Therefore, we introduce the following definition:

Definition A.8. A sequence $(x_t)_{t\in\mathbb{N}}$ is said to **converge at least linear** to $L \in \mathbb{R}$, if there exists a sequence $(\epsilon_t)_{t\in\mathbb{N}}$ such that;

$$|x_t - L| \leq \epsilon_t, \forall t \in \mathbb{N}.$$

In addition, the sequence $(\epsilon_t)_{t\in\mathbb{N}}$ should converge linear to zero.

Theorem A.3. *The iterative fairness- and goodness scores, calculated by FGA (Algorithm 6), converge at least linear with convergence rate* $\frac{1}{2}$.

Proof. In Theorem A.1 we showed that the algorithm converges over time for both fairness and goodness scores. Define $f^\infty(u), g^\infty(u) \in \mathbb{R}$ such that $f^t(u)$ converges to $f^\infty(u)$ and $g^t(u)$ converges to $g^\infty(u)$, for $u \in V$. In that same theorem we showed that $\forall t \in \mathbb{N}$ and $\forall u \in V$:

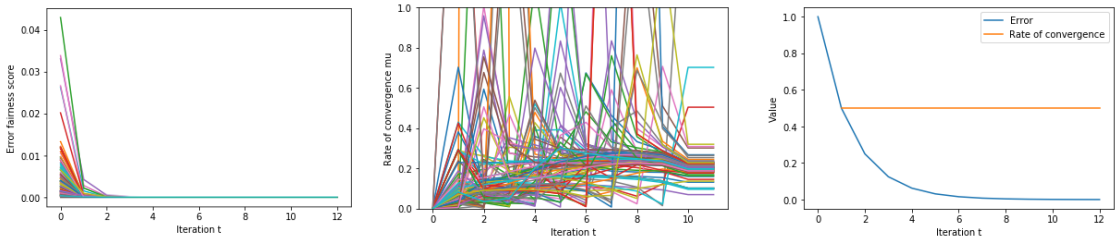
$$|f^t(u) - L_f| \leq \frac{1}{2^t} \text{ and } |g^t(u) - L_g| \leq \frac{R}{2^t}.$$

Define the sequences $(\epsilon_t^f)_{t\in\mathbb{N}} = \frac{1}{2^t}$ and $(\epsilon_t^g)_{t\in\mathbb{N}} = \frac{R}{2^t}$. Both sequences converge linear to 0 with convergence rate $\frac{1}{2}$, which makes our algorithm convergence at least linear with convergence rate $\frac{1}{2}$.

Only the calculation for proving linear convergence for sequence ϵ_t^f is showed, similar calculations can be made for ϵ_t^g.

$$\mu_f = \lim_{t\to\infty} \frac{|\epsilon_{t+1}^f - L|}{|\epsilon_t^f - L|} = \lim_{t\to\infty} \frac{|\epsilon_{t+1}^f|}{|\epsilon_t^f|} = \lim_{t\to\infty} \frac{|\frac{1}{2^{t+1}}|}{|\frac{1}{2^t}|} = \lim_{t\to\infty} \frac{2^t}{2^{t+1}} = \frac{1}{2}.$$

□



(a) Absolute error fairness score of 100 sample nodes in the WSN.

(b) Rate of convergence μ of 100 sample nodes in the WSN.

(c) Absolute error and rate of convergence of sequence $(\epsilon_t^f)_{t\in\mathbb{N}}$.

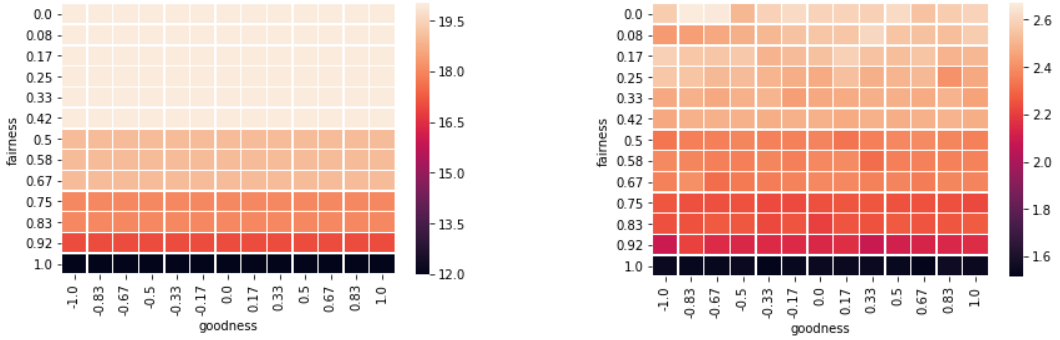
Figure 43: (a)(b) The absolute error and rate of convergence μ of the fairness score calculated on a sample of 100 nodes in the WSN. (c) The absolute error and rate of convergence of sequence $(\epsilon_t^f)_{t\in\mathbb{N}}$, with $\epsilon_t = \frac{1}{2^t}$ $\forall t \in \mathbb{N}$. The analysis is made using a WSN based on the Bitcoin OTC dataset with weights scaled between [-1,1], starting values $f^0(u) = 1$ and $g^0(u) = 0$, $\forall u \in V$ and stopping criterion $\epsilon = 10^{-6}$.

A.4.3 Choosing starting values and stopping criterion

Besides a WSN, the Fairness and Goodness Algorithm requires a stopping criterion $\epsilon \in \mathbb{R}\backslash\{0\}$ and starting values $f^0, g^0 \in \mathbb{R}^{|V|}$. Since the fairness and goodness iterations are compact interval bounded (Proposition A.6), a starting value in the bounded intervals $f^0(u) \in [0,1]$ and $g^0(u) \in [-\frac{R}{2}, \frac{R}{2}]$, $\forall u \in V$, ensures convergence (Theorem A.1). The main drawback of choosing the stopping criterion and starting values poorly is slow convergence. In this subsection the convergence speed is checked for different scenarios for both the starting values and stopping criterion.

Starting values. Fix the stopping criterion to be $\epsilon = 10^{-6}$. To test the time complexity and iterative power of the algorithm, an $n \times n$ raster was constructed in which each entry of the raster represents a fixed starting value for f and g. The f-axis varies from 0 to 1, whereas the g-axis varies from $\frac{R}{2}$ to $\frac{R}{2}$. The WSN used is based on the Bitcoin OTC dataset, where its weights are re-scaled between $[-1, 1]$ such that $R = 2$ (section 2.1).

The heatmap in Figure 44a shows that the only the choice of the starting value for fairness, influences the number of iterations. The optimal starting value for fairness is; $f^0 = \mathbb{1}^{|V|}$. From the heatmap in Figure 44b the same conclusion can be drawn, that also the choice of initial goodness score is insignificant.



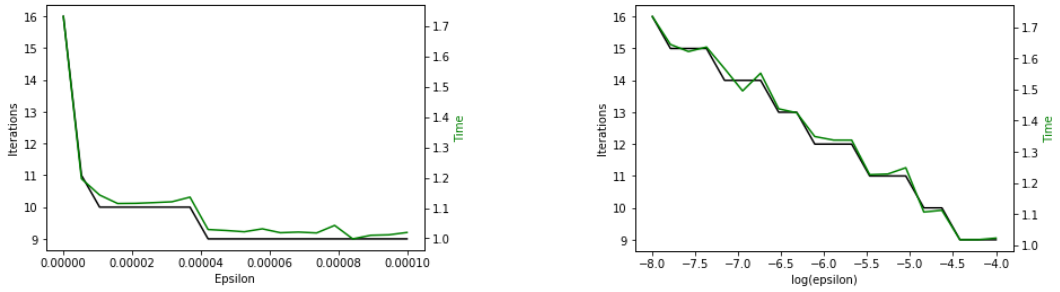
(a) Number of iterations.

(b) Computation time.

Figure 44: Two heatmaps visualizing the iterative power (a) and time complexity (b) for initial fairness and goodness values fixed on a raster. The analysis is made with the use of the Bitcoin OTC dataset and stopping criterion $\epsilon = 10^{-6}$.

stopping criterion. Fix the following initial fairness and goodness values; $f^0(u) = 1$ and $g^0(u) = 0$, $\forall u \in V$. The upcoming analysis looks at the time complexity and iterative power of alternating values for the stopping criterion ϵ. The same WSN, based on the Bitcoin OTC dataset, is used while analyzing the starting values.

In Figure 45a the iterative power and time complexity are sketched over alternating stopping criteria. Both the number of iterations as the time seems to have some logarithmic relation relation with the stopping criterion. In Figure 45b both the iterative power and time complexity are visualized with use of an logarithmic axis, as suspected the relationship is logarithmic.



(a) Linear scale.

(b) Logarithmic scale.

Figure 45: Iterative power and time complexity for alternating stopping criteria ϵ. The analysis is made with the use of the Bitcoin OTC dataset and initial values $f^0(u) = 1$ and $g^0(u) = 0$, $\forall u \in V$. (a) has a linear scale and (b) a logarithmic scale to show the logarithmic relations.

The choice of the stopping criterion should depend on the preferable error in the solution of the algorithm and the overall size of the input. As described in subsection A.4.2 about time complexity, the time complexity is mostly depending on the number of edges in the provided WSN. If $|E|$ is too large, the choice of ϵ can help improving the computation time. However, if we decrease the value of ϵ exponentially, the time complexity only increases linearly, which makes ϵ insignificant in the overall computation time.

A.4.4 Accuracy

Like most networks, WSN are incomplete and time-dependent. There may be like/dislike, trust/distrust or agree/disagree relations between people which we don't know or are yet to form. The overall fairness and goodness scores should not be drastically effected by a single bad rating. In other words, the scores of one's behavior need to be stable over time.

There are multiple approaches to evaluate the accuracy performance of the fairness and goodness scores, based on holding back data from the dataset. One approach is to remove $N\%$ of the edges from the network and try to predict the fairness and goodness scores with use of the remaining edges. We varied the value of N from 10 to 90% in steps of 10%, to observe the effect of a fraction of edges missing from the network. We consider the WSN G_{otc} based on the Bitcoin OTC dataset, where its weights are re-scaled between $[-1, 1]$. For each N we constructed 100 WSNs by sampling out $N\%$ of the edges from G_{otc}. From these 100 randomly generated sampled WSNs we calculate the mean of the errors, this reduces the influence of bad sampling.

The errors between expected fairness/goodness scores and the observed scores are measured by two standard metrics; the Root Mean Square Error (RMSE) and the Pearson Correlation Coefficient (PCC). The RMSE lies in range of 0 to 1 for the fairness scores and 0 to 2 for the goodness scores. According to the Cauchy-Schwarz [64] inequality the PCC has a value between $+1$ and -1, where $+1$ is total positive linear correlation, 0 no linear correlation, and -1 total negative linear correlation. Both these measures characterize different aspects of the prediction, RMSE quantifies how close the prediction is to the true value on average, whereas PCC measures the relative trend between the two. They are calculated as:

$$RMSE_{(X,Y)} = \frac{1}{n}\sum_{i=1}^{n}(X_i - Y_i)^2 \text{ and } PCC_{(X,Y)} = \frac{cov(X,Y)}{\sigma_X \sigma_Y},$$

for a pair of random variables (X, Y), n the number of observations and σ_X, σ_Y the standard deviations of the respective random variables. In Figure 46 the results of the experiment are shown. It is shown that the prediction of the fairness and goodness scores are linearly stable with an increasing N, making the proposed measures robust to network sparsity.

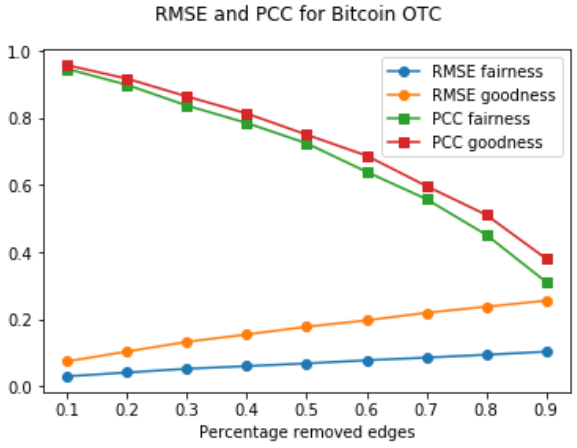


Figure 46: Performance of the fairness and goodness scores, measured by the Root Mean Square Error (RMSE) and the Pearson Correlation Coefficent (PCC). Predicting scores when $N\%$ percentages of the edges are removed from a WSN based on the Bitcoin OTC network. Algorithm 6 is used to calculate the fairness and goodness scores, using starting values $f^0(u) = 1$ and $g^0(u) = 0$, $\forall u \in V$ and stopping criterion $\epsilon = 10^{-6}$.

A.5 Missing values

Some nodes in a WSN will have no incoming- or outgoing edges, this will cause 'missing values' for the fairness and goodness scores. Let $G = (V, E, w)$ be a WSN, take in consideration the mutually recursive iterations, on which the fairness and goodness scores are based:

$$f(u) = 1 - \frac{1}{|out(u)|} \sum_{v \in out(u)} \frac{|w(u,v) - g(v)|}{R} \text{ and } g(v) = \frac{1}{|in(v)|} \sum_{u \in in(v)} f(u)w(u,v),$$

where $f(u)$ is the fairness score, $g(v)$ the goodness score, and $in(v), out(v)$ the in/out degree, $\forall v \in V$. Assume a node $u \in V$ has no outgoing edges, i.e. $|out(u)| = 0$. While calculating the fairness score the fraction $\frac{1}{|out(u)|}$ becomes undefined, creating a 'missing value' for the fairness score of a node u. Similarly can be said for a missing value of the goodness score, in which the corresponding node has no incoming edges.

The paper of Kumar et al. states that the missing values should be equal to its starting values, for fairness scores that would be 0 and for goodness one [42]. This causes addresses without contributed ratings to be the most 'fair' or 'reliable', and addresses without any received ratings to be the most 'liked' or 'trusted'. However, how could an individual without any ratings be more 'fair' than someone who has received ratings? From experience, especially in fraud and money laundering, it is best not to assume the best in people. Therefore a prediction based on the initial values of the fairness and goodness score will be unjust.

The missing values create uncertainties, and there are multiple approaches to predict the missing values for the fairness and goodness scores. Two methods will be discussed; spectral embedding and inverse scoring. Spectral embedding uses dimensionality reduction to create a network embedding for a WSN, on which a predictive model can approach the missing values. Inverse scoring uses the fairness and goodness scores of the inverse WSN to predict the missing values.

First, the missing values for the Bitcoin OTC network are discussed, which will be used as a reference on which different predictive models are applied. Second, spectral embedding and two predictive models (regression and clustering) on the found network embedding are introduced. Third, the inverse scoring of a WSN is argued. At last, the accuracy of all predictive methods is calculated and compared through the N%-taking out method.

Missing values of the Bitcoin OTC dataset. To compare the accuracy of the prediction models, the missing values of the Bitcoin OTC dataset are used. The Bitcoin OTC dataset represents a who-trust-whom network of people who trade Bitcoin using a platform called Bitcoin-otc (section 2.1). We consider the WSN $G_{otc} = (V, E, w)$ based on the Bitcoin OTC dataset, where its weights are re-scaled between $[-1, 1]$. The WSN G_{otc} has 5.986 nodes, and 36.108 edges. After calculating the fairness and goodness measure for G_{otc}, 1.135 missing values are detected, from which 1.113 missing fairness scores and 22 missing goodness scores. In Table 9 an overview is given on the impact of the missing values.

	Fairness	**Goodness**
Type of edge missing	Out-going	In-going
Total missing values	1.113	22
Percentage missing values	18,58%	0,37%

Table 9: Missing values of G_{otc}.

A.5.1 Prediction based on a spectral embedding

In many predictive models, for networks, the time complexity relies on the number of nodes and edges in the network, becoming impractical for large networks [44]. The technique of spectral embedding uses dimensionality reduction to create a network embedding of the WSN, on which predictive models can be trained to approach the missing values. The low dimension of the network embedding enables the model to be more scalable to large networks, which makes it an interesting technique considering the size of the Bitcoin network [27].

Random Dot Product Graph model (RDGP). A type of stochastic models, called the latent space models, was first proposed by Hoff, Rafferty, and Handcock in 2002 [29]. Under this model, each node is associated with a latent vector. A latent vector is a vector of random variables, whose realized values are hidden [63]. The latent space models assume that nodes are positioned in an n-dimensional latent space, and they tend to create edges between nodes that are closer in their latent positions. Due to this simple geometry-based assumption, the latent space models have an advantage in providing useful visualization and interpretation of a network [8].

A example of a latent space model is the random dot product graph (RDPG) model [68]. Under the RDPG model, the probability that an edge between two nodes is present is given by the dot product of their respective latent vectors. Let $\mathbf{X}, \mathbf{Y} \in \mathbb{R}^{n \times d}$ be such that;

$$\mathbf{X} = [X_1, X_2, \ldots, X_n]^T \text{ and } \mathbf{Y} = [Y_1, Y_2, \ldots, Y_n]^T,$$

with $X_i, Y_i \in \mathbb{R}^d$, $\forall i = 1, \ldots, n$, n being the number of nodes in the network, and d the target dimension. The matrices $\mathbf{X}$ and $\mathbf{Y}$ are random and satisfy;

$$\mathbb{P}[\langle X_i, Y_j \rangle \in [0, 1]] = 1, \, \forall i, j = 1, \ldots, n.$$

Conditioned on $\mathbf{X}$ and $\mathbf{Y}$, a representative matrix (normally adjacency or incidence) $\mathbf{A} \in \mathbb{R}^{n \times n}$ is independent and the entry $\mathbf{A}_{ij}$ is a Bernoulli random variable with parameter $\langle X_i, Y_j \rangle$ with $i \neq j$, such that:

$$\mathbb{P}[\mathbf{A}|\mathbf{X}, \mathbf{Y}] = \prod_{i \neq j} \mathbb{P}[\mathbf{A}_{ij}|X_i, Y_j] = \prod_{i \neq j} \langle X_i, Y_j \rangle^{\mathbf{A}_{ij}} (1 - \langle X_i, Y_j \rangle)^{1 - \mathbf{A}_{ij}},$$

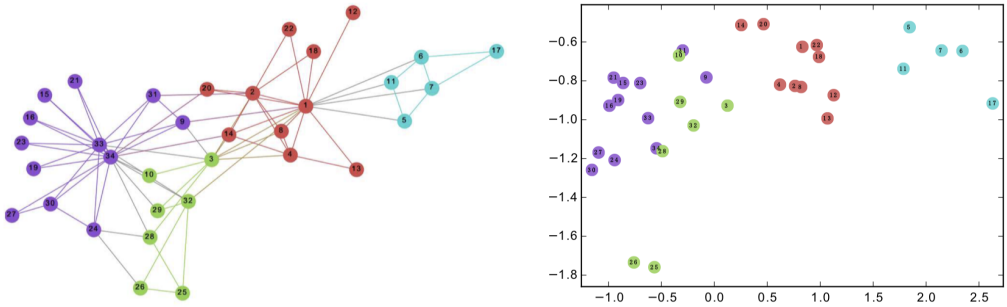
where the product is taken over all ordered pairs of nodes [68]. The variable $\mathbb{P}[\mathbf{A}|\mathbf{X}, \mathbf{Y}]$ represents the probability that an edge between two nodes exists.

Network embedding. To tackle the high-dimensional problem in networks, substantial effort has been committed to developing network embedding, i.e., learning low-dimensional vector representations for network nodes. Inside the network embedding, the relationships among the nodes is captured by the distances between nodes, and structural characteristics of a node are encoded into its embedding vector [53]. Figure 47a shows an example network, for which a 2-dimensional network embedding is determined (Figure 47b). Inside the embedding similar nodes are marked by the same color, the example shows that relations between nodes can be well modeled in a two-dimensional network embedding [54].

Sussman et al. presented an embedding motivated by the RDPG model that uses a decomposition of a representative matrix of the network [65]. The decomposition gives an embedding of the nodes as vectors in a low-dimensional space. The RDPG motivates the following embedding:

$$(\mathbf{X}, \mathbf{Y}) = \underset{(\dot{\mathbf{X}}, \dot{\mathbf{Y}}) \in \mathbb{R}^{n \times d} \times \mathbb{R}^{n \times d}}{\arg\min} ||\mathbf{A} - \dot{\mathbf{X}} \dot{\mathbf{Y}}^T||_F \tag{9}$$

where d is a fixed target dimension of the embedding, and $|| \cdot ||_F$ the Frobenius norm.



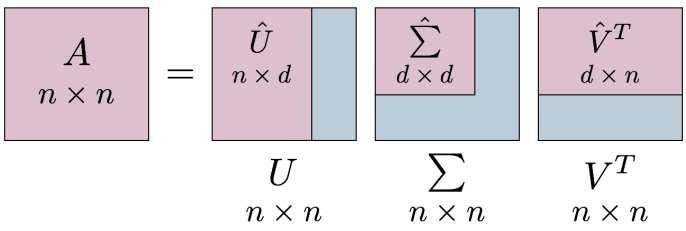
(a) Example network G.

(b) Network embedding of network G.

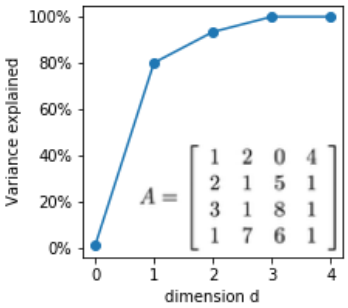
Figure 47: An example of network embedding, where similar nodes are marked by the same color. The coloring shows that the relations between nodes can be well modeled in a two-dimensional network embedding. Images are extracted from DeepWalk (Perozzi, Al- Rfou, and Skiena 2014) [54].

Definition A.9. Let $A \in \mathbb{R}^{n \times n}$. Define $A = U\Sigma V^T$ as the **singular value decomposition** (SVD) of A, where $U, V \in \mathbb{R}^{n \times n}$ are orthogonal and $\Sigma \in \mathbb{R}^{n \times n}$ is diagonal, with diagonal values $\sigma_1(A) \geq \sigma_2(A) \geq \cdots \geq \sigma_n(A) \geq 0$ the singular values of A.

Let $\tilde{U} \in \mathbb{R}^{n \times d}$ and $\tilde{V} \in \mathbb{R}^{n \times d}$ be the first d columns of U and V respectively, and let $\tilde{\Sigma} \in \mathbb{R}^{d \times d}$ be the sub-matrix of Σ given by the first d rows and columns. In 1936, Eckart and Young showed that Equation 9 is solved by the expressions $\mathbf{X} = \tilde{U}\tilde{\Sigma}^{1/2}$ and $\mathbf{Y} = \tilde{V}\tilde{\Sigma}^{1/2}$ [13]. We refer to $(\mathbf{X}, \mathbf{Y})$ as the "spectral embedding" of matrix A, and a vector in the spectral embedding as a "spectral embedding direction". A more clear illustration is given in Figure 48a.



(a) Visualization Singular Value Decomposition (SVD)



(b) Explained variance example matrix A.

Figure 48: (a) A visualization of Singular Value Decomposition and the determination of matrices $\hat{U}, \hat{\Sigma}$ and $\hat{V}$. (b) The percentage of explained variance in the spectral embedding for matrix A.

Explained variance. For each dimension d the SVD produces a set of $2 \cdot d$ vectors originated from matrices $\hat{U}$ and $\hat{V}$. Each set of vectors comes with a measure of its importance, i.e., the percentage of explained variance;

$$\hat{\sigma}_d^2(A) = \sum_{i=1}^{d} (\sigma_i(A))^2 \,, \tag{10}$$

with singular values $\sigma_1(A) \geq \cdots \geq \sigma_n(A) \geq 0$ of representative matrix A. Figure 48b illustrates the percentage of explained variance of a random matrix A. If our spectral embedding needs to explain

at least 50% of our matrix, we should only choose a spectral embedding of dimension $d = 1$. If this percentage should be 100%, dimension $d = 3$ is sufficient.

Algorithm outline. Algorithm 7 illustrates the main steps of the embedding procedure. In summary, the steps involve computing the singular value decomposition of a representative matrix, reducing the dimension, and coordinate-scaling the singular vectors by the square root of their singular values. The target dimension d is given as an input of the algorithm.

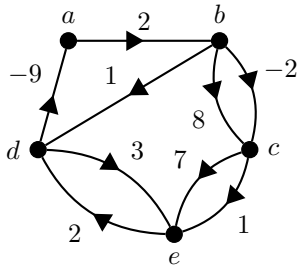
Algorithm 7 The spectral embedding procedure for directed graphs.

Require: Representative matrix $A \in \mathbb{R}^{n \times n}$ and target dimension $d \in \{1, 2, \dots n\}$.
Ensure: Z, a spectral embedding of dimension $2d$

1: Compute the singular value decomposition, $A = U\Sigma V^T$, with $U, \Sigma, V \in \mathbb{R}^{n \times n}$. Let Σ have a decreasing main diagonal.
2: Let $\tilde{U} \in \mathbb{R}^{n \times d}$ and $\tilde{V} \in \mathbb{R}^{n \times d}$ be the first d columns of U and V respectively, and let $\tilde{\Sigma} \in \mathbb{R}^{d \times d}$ be the sub-matrix of Σ given by the first d rows and columns.
3: Define $Z = [\tilde{U}\tilde{\Sigma}^{1/2} | \tilde{V}\tilde{\Sigma}^{1/2}] \in \mathbb{R}^{n \times 2d}$ to be the concatenation of the coordinate-scaled singular vector matrices.
4: **return** Z, the spectral embedding

Choosing a correct representative matrix. Spectral embeddings are performed on a representative matrix of the network. Since the fairness and goodness measures are mostly depending on the in- and out-degree of its nodes, we can argue how to choose this representative matrix. We consider types of representative matrices, the incidence matrix, and the adjacency matrix.

The incidence matrix represents the number of edges between two nodes. Whereas the adjacency matrix takes into account the weight of the edges. In Figure 49 an example WSN, its incidence and adjacency matrix are illustrated.



(a) Original WSN G

$$\begin{bmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 2 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

(b) Incidence matrix

$$\begin{bmatrix} 0 & 2 & 0 & 0 & 0 \\ 0 & 0 & -3 & 1 & 0 \\ 0 & 0 & 0 & 0 & 4 \\ -9 & 0 & 0 & 3 & 0 \\ 0 & 0 & 0 & 2 & 0 \end{bmatrix}$$

(c) Adjacency matrix

Figure 49: Example WSN and its incidence and adjacency matrix.

The argumentation of both matrices lies in the fact that both include a different aspect of the network. The incidence matrix takes into account the number of relations, whereas the adjacency matrix represents how strong these relations are. Spectral embeddings are mostly based on the adjacency matrix of the network. However, with the use of the BlockChain technology, we can cluster Bitcoin addresses to entities, which makes the introduction of the incidence matrix more arguable. The incidence matrix can capture how many relations are between clusters of Bitcoin addresses (entities), where it would be hard for the adjacency matrix to obtain the weights of a multi-graph.

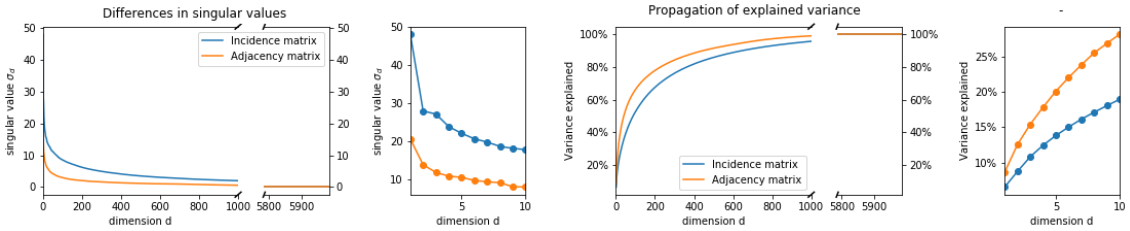
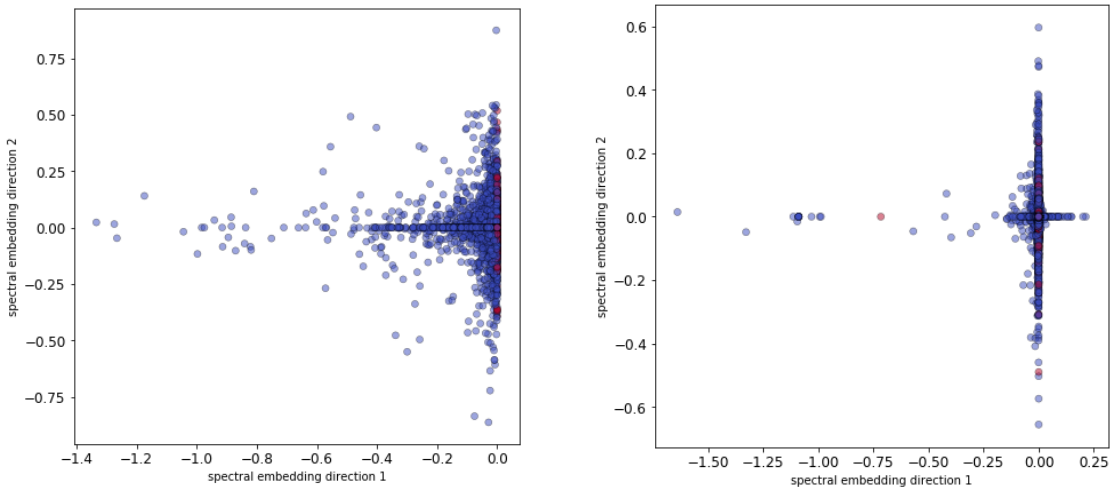


Figure 50: Difference in singular values and explained variance of the incidence and adjacency matrix.

In Figure 50, the proportion of explained variance is compared for both matrices, where the explained variance for the adjacency matrix is the largest. This means that for the same choice of dimension d the network embedding of the adjacency matrix is more likely to represent the original network. However, this doesn't mean that the data causing this variance corresponds to the data on which the fairness and goodness scores are based. Both the adjacency and incidence matrix embeddings are somehow representative, but we can't conclude from its variance, which is more suitable for predicting the fairness and goodness scores.



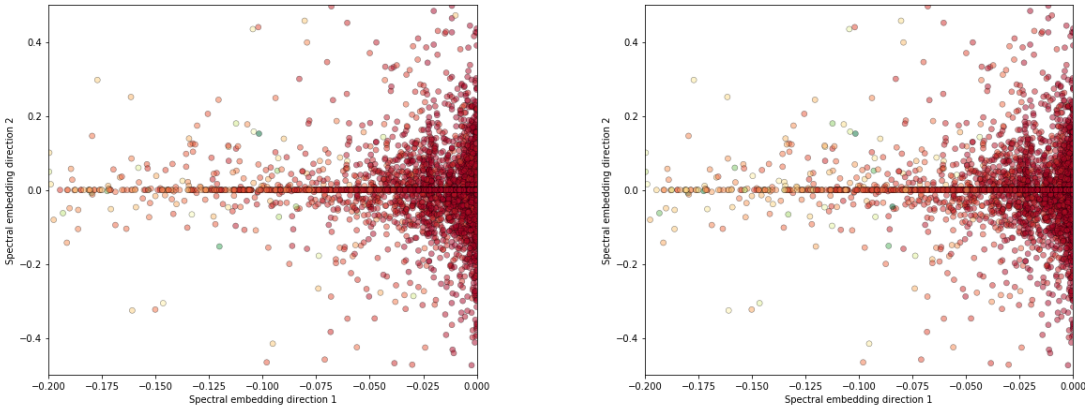
(a) Network embedding of incidence matrix.

(b) Network embedding of adjacency matrix.

Figure 51: Network embeddings of G_{otc} with $d = 1$. The red dots represents the missing values of the fairness and goodness scores.

In Figure 51 two example network embeddings, for the incidence and adjacency matrix, are visualized. The red dots in the figure indicate the missing values of the fairness and goodness scores. On these spectral embeddings, the prediction models, approximating the missing fairness and goodness scores, are based. Interesting is to see that the missing values, of both fairness and goodness, mostly lie on straight lines. This creates the suspicion that the first two spectral embedding directions are correlated with the in- and out-degree of G_{otc}.

Choice of target dimension d. To simplify the prediction, with the use of spectral embedding, a fixed target dimension d is chosen. In Figure 52, a degree node coloring is given over the network embedding of the incidence matrix. As suspected, the first two spectral embedding vectors are correlated with the in- and out-degree of the network G_{otc}. Since the fairness and goodness measures depend on the degrees, a network embedding with dimension $d = 1$ is chosen to represent the WSN.



(a) Node coloring in-degree.

(b) Node coloring out-degree.

Figure 52: The degree node coloring of the network embedding of the incidence matrix, with (a) the in-degree and (b) the out-degree. Red colors indicate low degree, whereas green colors indicate high degree. The Figures are zoomed-in on a subset of the network embedding to make the visualization more clear.

Predictive models for missing fairness/goodness score. Predictive models are used to approximate the missing values on the network embeddings. Two types of predictive models are introduced; linear regression and clustering. Linear regression tries to find a linear trend in the data, which explains the relationship between the two embedding directions and missing values [70]. Clustering organizes nodes into communities or clusters of nodes, where the nodes in the same cluster are highly similar. The missing values of a node are likely to be similar to the fairness and goodness scores of a node in the same cluster. In Figure 53, the network embeddings are visualized with respect to the fairness score.

(A) Linear regression.

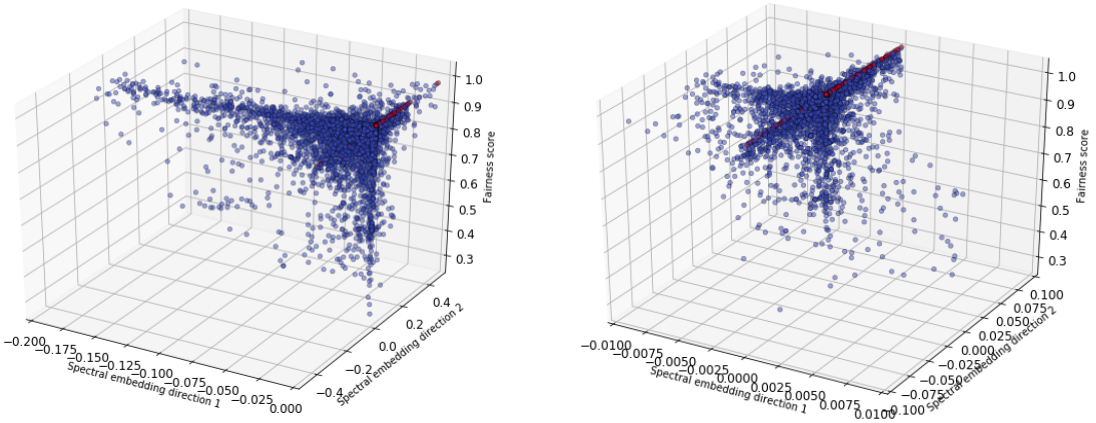
Linear regression is a statistical process for estimating the relationships among variables [18]. In this case, the dependent variable is either the fairness or goodness score, and the two spectral embedding directions the predictive variables. Linear regression is one of the most famous methodologies to forecast a variable's behavior [71]. Mostly because it is a simple, yet powerful technique. The importance of regression analysis is that it helps determine which factors matter most, which it can ignore, and how those factors interact with each other [70].

Outline method. For the set of predictive variables $\{\mathbf{x}_1, \mathbf{x}_2\} \in \mathbb{R}^n$ and a dependent variable $\mathbf{y} \in \mathbb{R}^n$, the linear relationship between the variables can be written as:

$$\mathbf{y} = \beta_0 + \beta_1 \mathbf{x}_1 + \beta_2 \mathbf{x}_2 + \epsilon,$$

with parameters $\beta_0, \beta_1, \beta_2 \in \mathbb{R}$ and an unobserved random error ϵ with expectation zero. The above equation can be written as a system of linear equations:

$$\begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} \\ \vdots & \vdots & \vdots \\ 1 & x_{n1} & x_{n2} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \end{bmatrix} + \begin{bmatrix} \epsilon_1 \\ \vdots \\ \epsilon_n \end{bmatrix}$$



(a) Network embedding of incidence matrix.

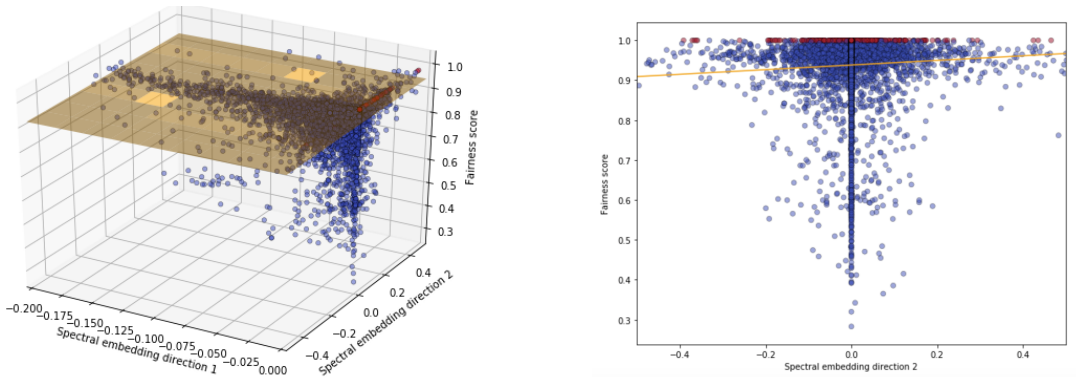
(b) Network embedding of adjacency matrix.

Figure 53: The network embeddings with respect to the fairness score of G_{otc}. The Figures are zoomed-in on a subset of the network embedding to visualize more clear the distribution of the fairness scores. The red dots represents the missing values of the fairness scores, which needs to be predicted.

The system of linear equations can be written in matrix notation, such that $\mathbf{X} = [\mathbb{1}^n, \mathbf{x}_1, \mathbf{x}_2] \in \mathbb{R}^{n \times 3}$ is the predictor matrix, $\boldsymbol{\beta} \in \mathbb{R}^3$ the vector of parameters, and $\boldsymbol{\epsilon} \in \mathbb{R}^n$ the random error vector. With the use of the least squares estimation the vector of estimated polynomial regression parameters $\hat{\boldsymbol{\beta}}$ can be derived [21];

$$\hat{\boldsymbol{\beta}} = \left(\mathbf{X}^T\mathbf{X}\right)^{-1}\mathbf{X}^T\mathbf{y}$$

The linear regression is implemented in Python using the `LinearRegression` function of the `sklearn` class.



(a) 3-dimensional visualization.

(b) 2-dimensional visualization.

Figure 54: Visualization of the linear regression of predicting the missing fairness scores of G_{otc}. The network embedding is based on the incidence matrix, with dimension $d = 11$. The Figures are zoomed-in on a subset of the network embedding to visualize more clear the distribution of the fairness scores. The red dots represents the missing values of the fairness scores, which needs to be predicted.

(B) Prediction based on clustering.
An interesting structure feature that real networks present are the clustering or community property, under which the graph is organized into modules commonly called communities or clusters. The essence is that nodes of the same cluster are highly similar while on the contrary, nodes across clusters present low similarity. The main idea is that nodes in the same cluster should reflect similar behavior, which makes the fairness and goodness scores of these nodes more similar, than those of the nodes outside the cluster.

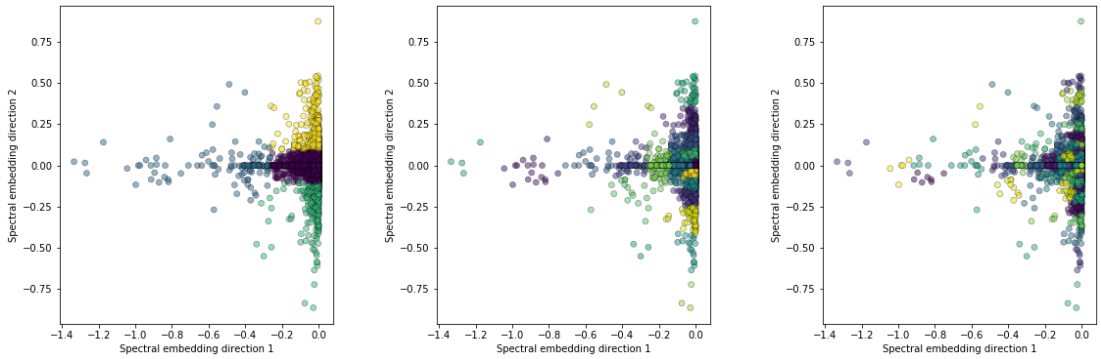
Outline method. The K-means clustering criteria aims to clusters nodes into K clusters in which each observation belongs to the cluster with the nearest mean [47]. Given is the network embedding $Z \in \mathbb{R}^{N\times 2}$, with observations $\{\mathbf{v}_1, \ldots, \mathbf{v}_N\} \in \mathbb{R}^2$ each representing the node-coordinates of the $\{v_1, \ldots, v_N\}$ nodes in the 2-dimensional network embedding. The clustering organized each node into K ($\leq N$) sets $S = \{S_1, S_2, \ldots, S_K\}$ while minimizing the within-cluster sum of squares errors:

$$\arg\min_{S} \sum_{i=1}^{K} \sum_{v_j \in S_i} ||\mathbf{v}_j - \boldsymbol{\mu}_i||^2 = \arg\min_{S} \sum_{i=1}^{K} |S_i| \mathrm{Var}(S_i),$$

with $\boldsymbol{\mu}_i$ the mean of nodes in S_i. The missing values of the fairness and goodness scores are then predicted by taking the average of the fairness and goodness scores of all nodes in a certain cluster. Let $v \in V$ be a node with a missing fairness score, such that;

$$f(v) = \frac{1}{|\hat{S}| - 1} \sum_{v_j \in \hat{S} \setminus \{v\}} f(v_j), \text{ with } \hat{S} = \{S_i \in S : v \in S_i\}.$$

Then $f(u)$ is the missing fairness score for node u. The K-means clustering algorithm is implemented in Python using the `KMeans` function of the `sklearn` class. In Figure 55 a visualization with alternating K is made for the found clusters, using the K-means algorithm over the 2-dimensional network embedding for the incidence matrix.



(a) K=4. (b) K=20. (c) K=50.

Figure 55: The found K clusters using the K-means clustering algorithm. The clustering is made on a 2-dimensional network embedding of the incidence matrix.

But how is the optimal value of K chosen? The accuracy is measured by the $N\%$ method, where $N\%$ missing values are created in the G_{otc} network, excluding the already found missing values. To reduce bad sampling, 500 networks are simulated, and the error is calculated by taking the average of all sampled network errors. (For more information see subsection A.4.4).

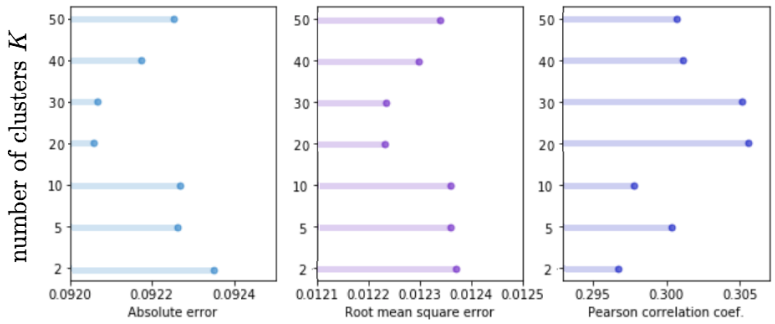


Figure 56: Calculated errors of the K-mean clustering algorithm, using the a $N\% = 20\%$ leaving out method and alternating values of K.

In Figure 56 some standard errors are listed for different values of K. The errors between expected missing values and the observed missing values are measured by three standard metrics; the absolute error, the Root Mean Square Error (RMSE) and the Pearson Correlation Coefficient (PCC). All these measures characterize different aspects of the prediction, the absolute error and RMSE quantifies how close the predictions are to the true values on average, whereas PCC measures the relative trend between the two. It can be concluded that the optimal value of clusters will be around $K = 20$.

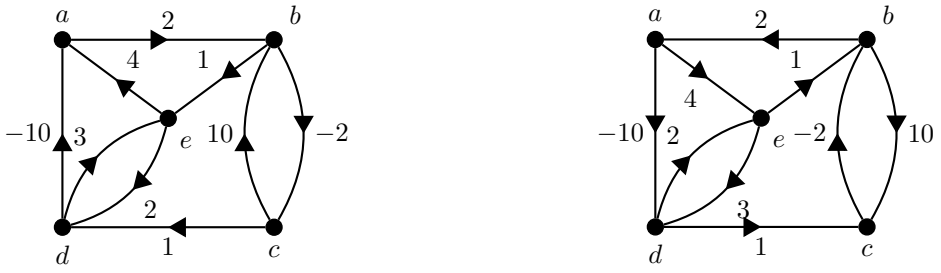
A.5.2 Predictive models based on inverse scoring Fairness and Goodness

A disadvantage of dimensionality reduction is the disappearance of information, and it is unclear if the kept information can explain the fairness and goodness scores. This new method, which we call 'inverse scoring', conserves all data in the dataset. It is based on the principle that the missing fairness and goodness scores can be predicted by the fairness and goodness score of it's 'inverse' WSN.

Definition A.10. Given a WSN $G = (V, E, w)$, the **inverse WSN of** G is defined as $G_I = (V, E_I, w_I)$, where:

$$E_I = \{e_I(u, v) : e(v, u) \in E \text{ with } u, v \in V\} \text{ and } w_I(u, v) = w(v, u),\ \forall u, v \in V$$

To make the above definition more clear Figure 57a represent a example WSN G and Figure 57b its corresponding inverse WSN G_I. All edges are 'flipped', which means that an incoming edge $e = (u, v)$ in the original WSN G becomes an outgoing edge $e_I = (v, u)$ in the inverse WSN G_I, with equal weight. Also, on the inverse WSN, the fairness and goodness scores can be calculated.



(a) Original WSN G (b) Inverse WSN G^I

Figure 57: Small example on how a WSN is transformed into its inverse.

Definition A.11. Given a WSN $G = (V, E, w)$ and its inverse WSN $G_I = (V, E_I, w_I)$. We define $f_I(u)$ as the **inverse fairness score** and $g_I(u)$ as the **inverse goodness score**,$\forall u \in V$, such that:

$$f_I(u) = 1 - \frac{1}{|out(u)|} \sum_{v \in out(u)} \frac{|w_I(u, v) - g_I(v)|}{R}$$

$$g_I(v) = \frac{1}{|in(v)|} \sum_{u \in in(v)} f_I(u) w_I(u, v)$$

Correlation between the original and inverse WSN. The predictive model of inverse scoring is based on a correlation between the fairness and goodness scores of the original WSN and its inverse WSN. Interesting is to see if there indeed exists any relationship between the two scores. By rewriting the fairness and goodness scores in matrix notation, an interesting property can be extracted.

Definition A.12. Let $A \in \mathbb{R}^{n \times n}$ be the adjacency matrix of a WSN with n nodes. The vectors $\mathbf{f}, \mathbf{g} \in \mathbb{R}^n$ are the **the vector representations of the fairness and goodness scores**, such that:

$$\mathbf{f} = \mathbb{1}^n - \left|A^T - \mathbf{g}(\mathbb{1}^n)^T\right| \mathbf{d}_{out} \text{ and } \mathbf{g} = \left(A^T \mathbf{f}\right) \mathbf{d}_{in},$$

with $\mathbf{d}_{in}, \mathbf{d}_{out} \in \mathbb{R}^n$ the vector representations of the in- and out- degree of the nodes in the WSN.

Similar, the inverse fairness $\mathbf{f}_I \in \mathbb{R}^n$ and goodness $\mathbf{g}_I \in \mathbb{R}^n$ scores can be defined. By definition of an inverse WSN, the adjacency matrix of the inverse WSN is given by the transpose of the adjacency matrix ($A^T \in \mathbb{R}^{n \times n}$) of the original WSN. It's clear to see that if $A = A^T$ both fairness and inverse fairness scores are equal (same for goodness). To give extra strength to the argument, a representation is made in Figure 58. The Figure visualizes the original fairness and goodness scores in comparison to its inverse scoring. The used WSN is a randomly sampled network with 6.000 nodes and 37.000 edges. The WSN is sampled symmetric such that $A = A^T$ and uses uniform sampled weights for the edges. As expected, the fairness score and its inverse are perfectly correlated, and similar can be said about the goodness scores.

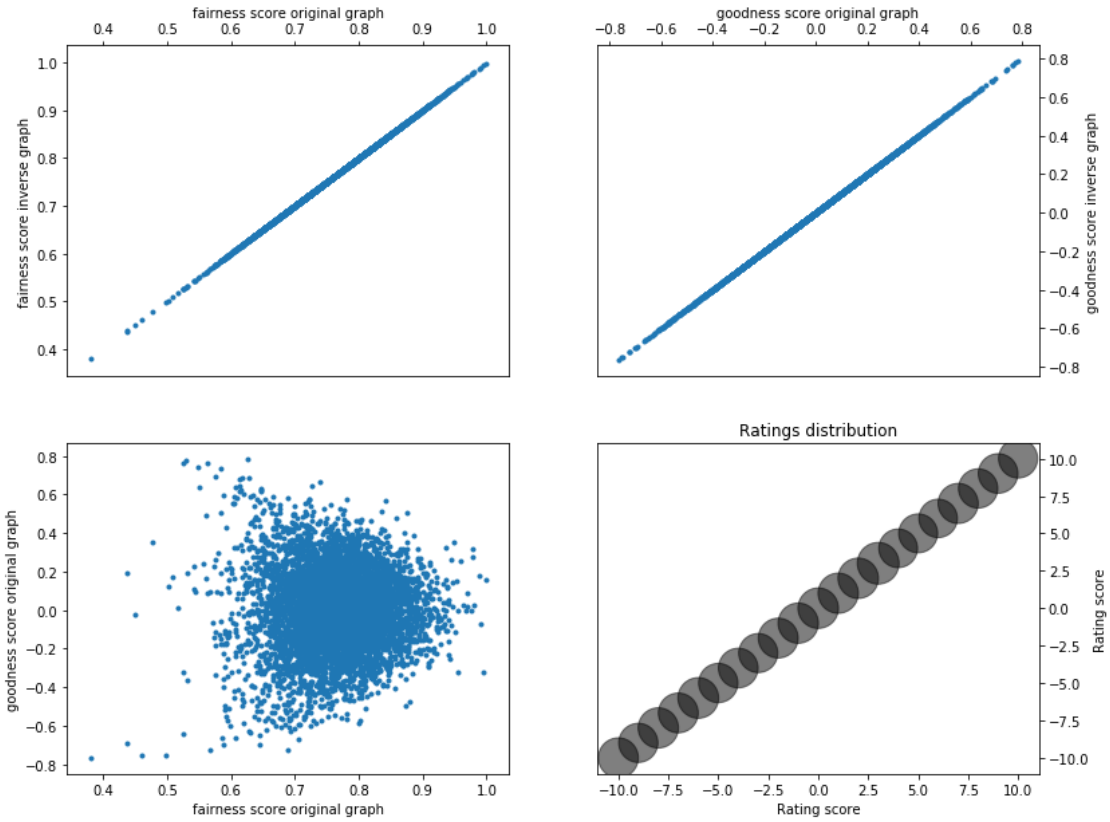


Figure 58: Random sampled symmetric WSN of 6.000 nodes and 37.000 edges, with its weights uniform distributed.

However, real-life networks are not symmetric. This is caused by two factors: Bitcoin users don't give mutual ratings, and Bitcoin users forget to give ratings. This causes distortions in the correlation between fairness/goodness and their inverse scoring. In the upcoming subsection, we are going to investigate how these two factors influence the correlation between both scores.

(A) Unsymmetrical ratings.
Symmetrical ratings are defined as ratings that are equal for both the receiving and sending the user of the Bitcoin transaction. So if Bitcoin address A sends user B Bitcoins and receives for that transaction a rating of 5, user A gives a mutual rating of 5 to user B. In Figure 59 is illustrated how unsymmetrical ratings influence the correlation between the fairness/goodness scores and their inverses. With an increase of absolute difference between the given ratings, the fairness/goodness scores and their inverse become less correlated.

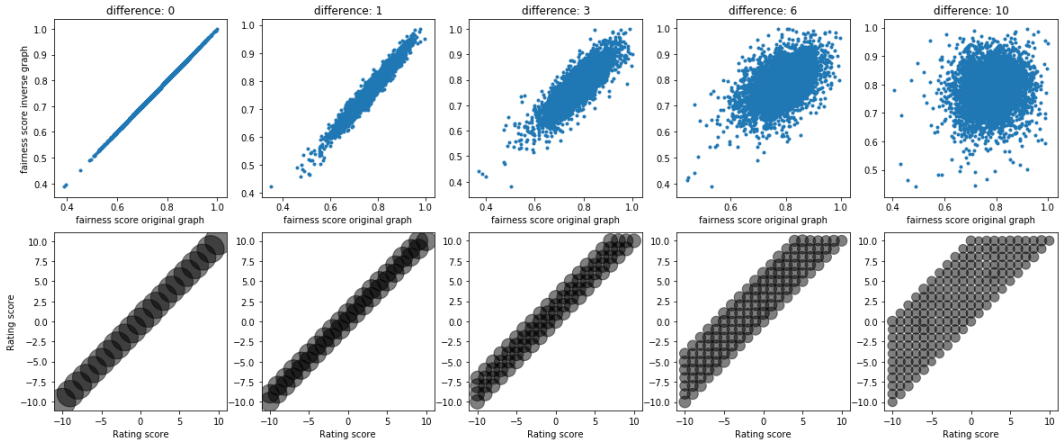


Figure 59: Random sampled symmetric WSN of 6.000 nodes and 37.000 edges, where the weights are sampled uniform with respect to a possible absolute difference. These differences walk from 0 to 10, to illustrate what the impact of un-similarity has on the correlation between the score and their inverses.

(B) Missing ratings.
Missing ratings appear when a user rated another user but didn't receive a rating back. In Figure 60 is illustrated how missing ratings influence the correlation between the fairness/goodness scores and their inverses. With the increase of the percentage of missing ratings, the fairness/goodness scores and their inverse become less correlated.

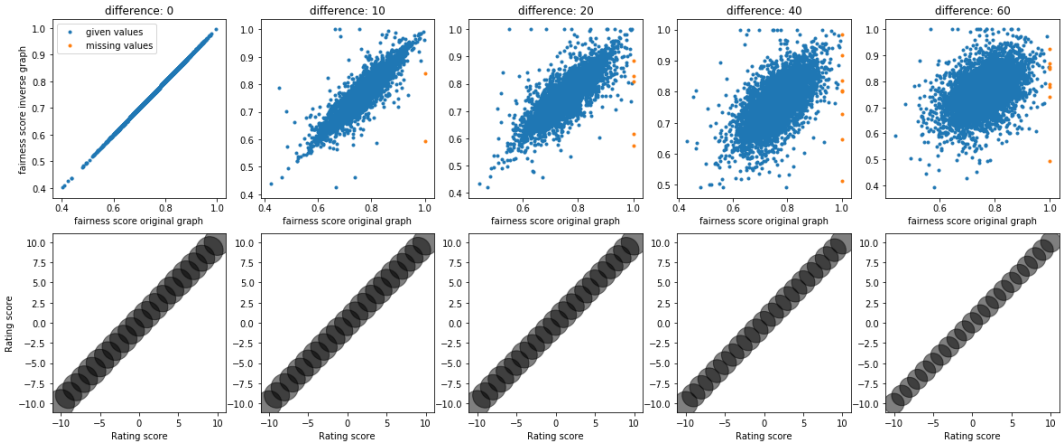


Figure 60: Random sampled symmetric WSN of 6.000 nodes and 37.000 edges, with its weights uniform distributed. A certain percentage of the transactions have missing ratings. These percentages walk from 0 to 60, to illustrate what the impact of un-similarity has on the correlation between the score and their inverses.

Choosing a mapping. The original OTC Bitcoin network has both unsymmetrical features as missing ratings, both cause distortions in the relationship between the fairness/goodness scores and their inverse scoring. How the missing values of the fairness and goodness scores are chosen for their inverse scoring can differ. We consider two scenarios; a simple identity mapping between both scores, and a linear trend line mapping the fairness/goodness scores to their inverses.

Definition A.13. Let $f(u), g(u)$ be the fairness and goodness scores of a WSN and $f_I(u), g_I(u)$ the inverse fairness and goodness scores of the inverse WSN, $\forall v \in V$ of both WSNs.

i) The function $h_1 : \mathbb{R} \to \mathbb{R}$ is the **identity mapping** between the fairness and goodness scores and their inverses, such that:

$$h_1 : f(u) \mapsto f_I(u), \forall v \in V$$

ii) The function $h_l : \mathbb{R} \to \mathbb{R}$ is the **linear mapping** between the fairness and goodness scores and their inverses, such that:

$$h_1 : f(u) \mapsto \alpha f_I(u) + \beta, \forall v \in V \text{ for some parameters } \alpha, \beta \in \mathbb{R}$$

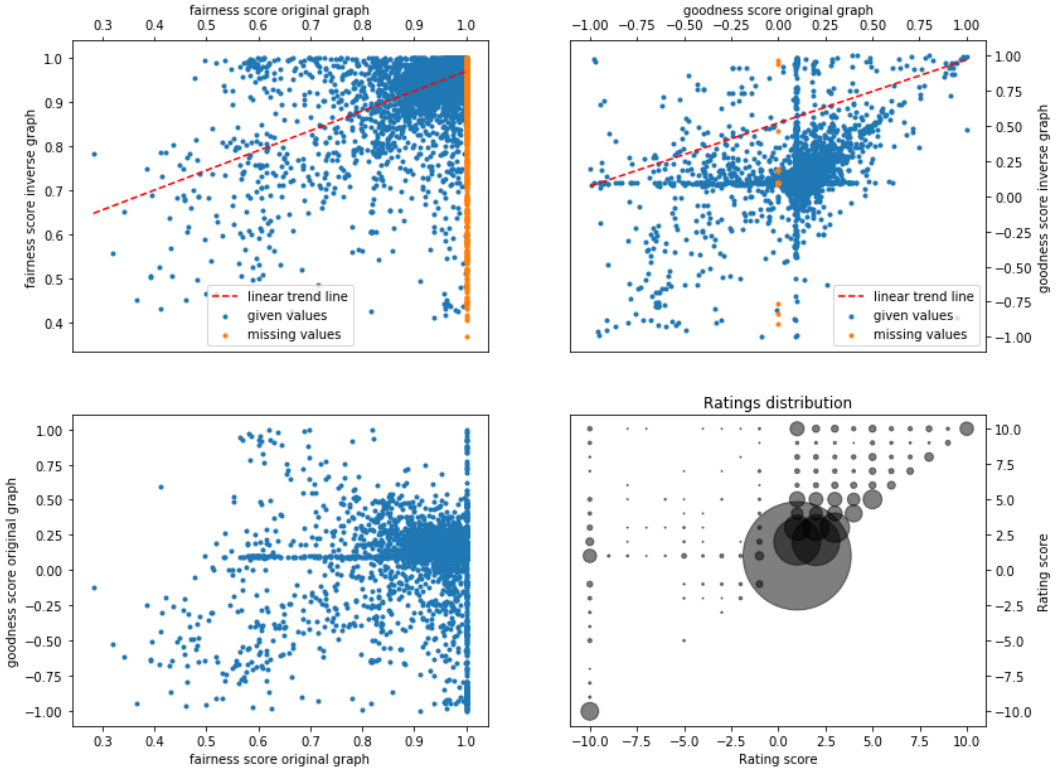


Figure 61: Fairness and goodness scores of WSN G_{otc}, the correlation between the original scores and their inverses, and the rating distribution.

For both types of mappings, something can be said. The identity mapping is simple to compute, and simple to interpretive. However, the distortions create between the fairness/goodness scores, and their inverses are not equivalent. Some WSN will have more distortions created by un-symmetric ratings, and some will have more distortions created by missing values. In this case a linear trend line (the linear mapping) between both scores can capture those differences and be more accurate, however, this includes an extra step in calculating the missing values.

In Figure, 61 the correlation between the fairness/goodness scores and their inverses, and the rating distribution of the WSN G_{otc} is visualized. The WSN G_{otc} is based on the Bitcoin OTC dataset. There exist significant distortions between the original and inverse scores, since G_{otc} has many unsymmetrical ratings and missing ratings. The red line in the Figure indicates the linear trend-line between both scores.

A.5.3 Accuracy of the predictive models

One of the simplest methods of determining the accuracy between all predictive models is the $N\%$ method. This approach is based on generating $N\%$ missing values for the G_{otc} WSN, and trying to predict these missing values with the use of the predictive models. The WSN G_{otc} is based on the Bitcoin OTC dataset.

Bitcoin networks are dynamic over time, which can cause high sparsity of transactions at certain time intervals. Therefore a substantial increase of missing values can occur, interesting is to see how the predictive power of the models is under different amounts of missing values. With varying values of N between 2% and 70%, large percentages of missing values are investigated. To reduce the influence of bad sampling for each N, 100 randomly generated WSNs are used.

The analysis is made separately for both the fairness and goodness score since a node should have either a missing fairness or missing goodness score. If the node has both missing values, the node is trivially disconnected from the network (zero in- and out-degree).

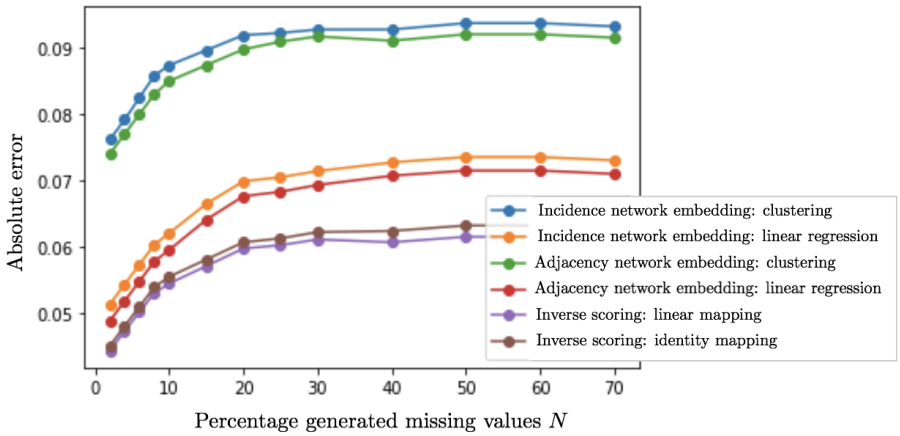


Figure 62: Absolute error of all predictive models approximating the missing fairness scores, with $N\%$ generated missing fairness scores.

In Figure 62, the performance of the predictive models are measured by the absolute error between the predicted missing fairness score and the actual fairness score. Since 20% of the fairness scores are missing in the original OTC network, we mainly compare the accuracy of the predictive models for that value of N. The predictive models, based on the inverse scoring of the WSN gives the best results. In which the linear mapping performs better than the identity mapping.

Note that this analysis is made for a specific WSN G_{otc} the results could differ for different WSNs.

A.6 Implementation on the Bitcoin OTC dataset

The Bitcoin OTC dataset represents the who-trusts-whom network of people who trade Bitcoin using a platform called Bitcoin-otc. The Bitcoin OTC dataset provides four entries per line; source, target, rating, and timestamp. *Source* notated the node id of the source, *target* the node id of the target, *rating* the source's rating for the target, ranging from -10 to +10 and *timestamp* the time of the rating, measured as seconds since Epoch.

We will construct a WSN $G_{otc} = (V, E, w)$ by taking all sources and targets as nodes V and the performed ratings as the edges E. The direction of the edges is from source to target, and their ratings are used as weights w. For more detailed information on the Bitcoin OTC dataset, we refer to subsection 2.1.

In total G_{otc} has 5.986 nodes and 36.108 transactions. For each node in G_{otc} the fairness and goodness scores are calculated with the use of the FGA (Algorithm 6). As extra requirements we used stopping criterion $\epsilon = 10^{-6}$ and starting values $f^0(u) = 1$ and $g^0(u) = 0$, $\forall u \in V$. It took the algorithm 14 iterations to calculate all fairness and goodness scores. In Figure 63, the resulting fairness and goodness scores are visualized, before determining the missing values.

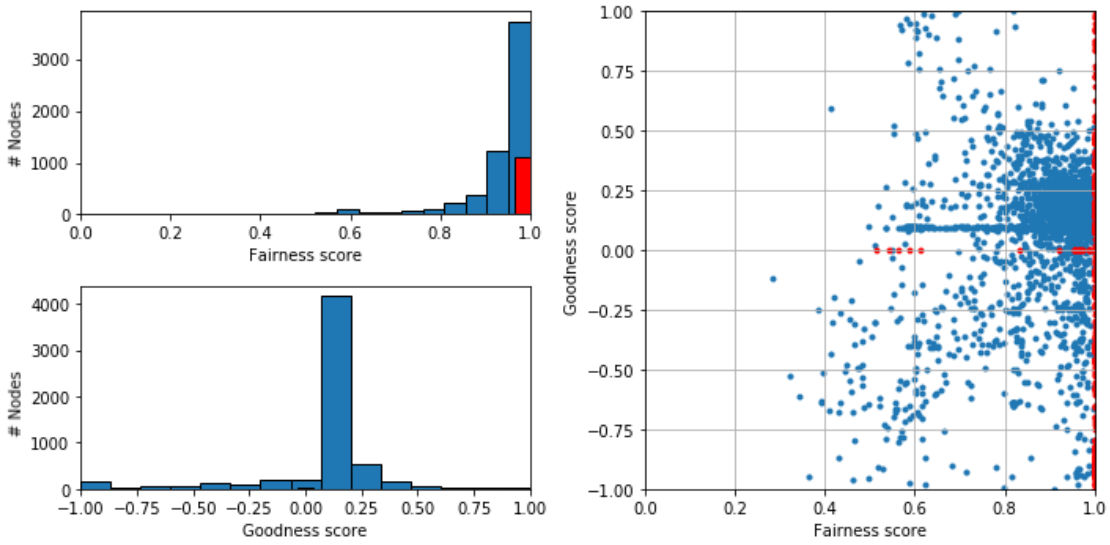


Figure 63: The calculated fairness and goodness scores for the Bitcoin OTC dataset, without predicting the missing values, which are colored red. As expected they lie on straight lines at the fairness score being one and the goodness score being zero.

Predicting the missing values. The Bitcoin OTC dataset has 1.135 missing values, from which 1.113 missing fairness scores and 22 missing goodness scores. In Table 10, an overview is given on the impact of the missing values.

	Fairness	**Goodness**
Type of edge missing	Out-going	In-going
Total missing values	1.113	22
Percentage missing values	18,58%	0,37%

Table 10: Missing values of G_{otc}.

Using linear mapping on the inverse scoring of the fairness and goodness scores will give the best

estimate of the missing values (subsection A.5). The function for the linear trend line, from the inverse fairness score to the fairness score for the missing value, is defined as p_f. Similarly, the function p_g is the trend line between goodness scores. Through analysis the following variables for both trendlines are extracted;

$$p_f(x) = 0.45 * x + 0.52 \text{ and } p_g(x) = 0.42 * x + 0.12.$$

With the use of both functions, the missing values are predicted. In Figure 64, the final fairness and goodness scores are visualized.

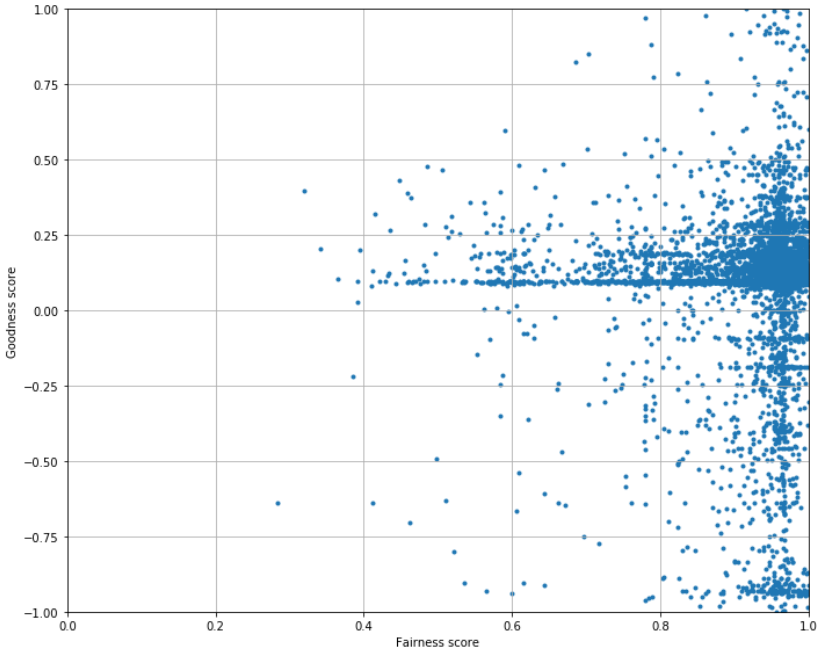


Figure 64: Fairness and goodness scores for the Bitcoin OTC dataset.

A.7 Summary & Conclusion

A WSN is a weighted signed network that is characterized by their property of capturing like/dislike, trust/distrust, and other social relationships between entities. For a WSN we introduced two measures of node behavior: where the goodness of a node intuitively captures how much this node is liked/trusted by other nodes, while the fairness of a node captures how fair the node is in rating other nodes' likeability or trust level.

The fairness and goodness scores are computed in a mutually recursive manner, were both scores are independent of each other and compact interval bounded. If the initial values of both scores lie in this interval, the algorithm calculating the fairness and goodness scores converges to a unique solution. Furthermore, the algorithm has linear time complexity and robust to network sparsity.

Some nodes in the WSNs will have no incoming- or outgoing edges, and this will cause missing values for the fairness and goodness scores. The definition of both scores depends on a fraction with denominator the in- or out-degree of a node, which becomes undefined if these degrees are zero. The missing values create uncertainties, and two approaches to predict the missing values are discussed; spectral embedding and inverse scoring. The predictive models based on the inverse scoring turned out to be the most accurate for the WSN, based on the Bitcoin OTC dataset.

B Code

B.1 Web scraper to construct the Bitcoin OTC dataset

```
### 5 March 2019
### A. van Schetsen

library(jsonlite)
library(data.table)

parent_link <- "https://www.bitcoin-otc.com/viewgpg.php?outformat=json"
parent <- as.data.table(fromJSON(parent_link))
parent[,c("keyid","fingerprint","registered_at","last_authed_at","is_authed"):= NULL]
parent <- parent[!nick=="#bitcoin-otc",]
parent[,nick:=tolower(nick)]
parent <- parent[order(nick),]

missing_parent = data.table()
otc = data.table()

# Load all JSON nickname files and save the transactions in data table 'otc'
for(i in parent$id){
  nickname = parent[id==i,nick]
  nick_link <- paste("https://www.bitcoin-otc.com/viewratingdetail.php?nick="
                  ,nickname,"&outformat=json",sep="")
  nick_data <- as.data.table(fromJSON(nick_link))
  if(!(length(nick_data)==0)){
    colnames(nick_data)[4] = "timestamp"
    nick_data[,rater_nick:=tolower(rater_nick)]

    # Some nicknames are not included in the parent datatable, save & add them
    missing <- data.table(nick=nick_data$rater_nick
                [!nick_data$rater_nick%in%parent$nick])
    missing[,id:=max(as.integer(parent$id))+seq(.N)]
    parent = rbind(parent,missing,fill=TRUE)
    parent <- parent[order(nick),]
    missing_parent = rbind(missing_parent,missing)

    nick_data <- nick_data[order(rater_nick),]
    nick_data[,source:=parent[nick%in%nick_data$rater_nick,id]]
    nick_data[,target:=i]
    otc = rbind(otc,nick_data[,.(source,target,rating,timestamp)])
  }
}

# Write Bitcoin OTC dataset
write.csv(otc,file="DT_OTC.csv")
```

B.2 Constructing the Bitcoin MIT dataset

```
#### 16 April 2019
#### A. van Schetsen

library(data.table)
library(ggplot2)

################################################
# Construct transaction network
################################################

# Block hashes
```

```
bh <- fread('bh.dat', header = FALSE)
colnames(bh) = c("blockID", "hash", "timestamp", "n_txs")
ts = c(1446336000,1502323200) # Dec 2015 until July 2017
bh <- bh[timestamp>=ts[1]&timestamp<ts[2],]

# Transaction overview
tx_overview <- fread('tx.dat', header = FALSE)
colnames(tx_overview) = c("txID", "blockID", "n_inputs", "n_outputs")
tx_overview <- tx_overview[blockID%in%bh$blockID,]
tx_overview <- tx_overview[n_inputs==1,]
tx_overview <- tx_overview[n_outputs==1,]
tx_overview <- merge(tx_overview,bh[,.(blockID,timestamp)],by="blockID",all=FALSE)
txID = tx_overview$txID

# Find transactions; trin & trout
trin <- find_trin(txID)
trout <- find_trout(txID)

# Merge all data into DT
trx <- merge(trin,trout, by="txID", allow.cartesian=TRUE, all=TRUE)
colnames(trx) = c("txID","in_addrID","in_sum","out_addrID","out_sum")
trx[,amount:=in_sum-out_sum]
trx = trx[amount>0,]
trx = trx[in_addrID>0,]
trx = trx[out_addrID>0,]

DT = merge(trx[,.(txID,in_addrID,out_addrID,amount)],tx_overview[,.(txID,timestamp)],
        by="txID",all=FALSE)

# Write Bitcoin MIT dataset
write.csv(DT,file="DT_MIT.csv")

################################################
# Functions
################################################

find_trin <- function(trID){
  files = list.files()[-c(1:2)]
  trin = data.table()
  for (file in files){
    trin_temp <- fread(file, header=FALSE)
    colnames(trin_temp) = c("txID", "input_seq", "prev_txID",
                "prev_output_seq", "addrID", "sum")
    trin_temp[,c("input_seq","prev_txID","prev_output_seq"):= NULL]
    trin_temp = trin_temp[txID%in%trID,]
    trin = rbind(trin,trin_temp)
    rm(trin_temp)
  }
  return(trin)
}

find_trout <- function(trID){
  files = list.files()[-c(1:2)]
  trout = data.table()
  for (file in files){
    trout_temp <- fread(file, header=FALSE)
    colnames(trout_temp) = c("txID", "output_seq_seq", "addrID", "sum")
    trout_temp[,output_seq_seq:=NULL]
    trout_temp = trout_temp[txID%in%trID,]
    trout = rbind(trout,trout_temp)
    rm(trout_temp)
  }
  return(trout)
}
```

B.3 1-ARW-betweenness centrality measure

```
#### 21 April 2019
#### A. van Schetsen
#### 1-ARW-betweenness centrality measure

import numpy as np

def ARW(A,alpha):
    n = A[0].size

    # No incoming edges
    pI = A.sum(axis=0)
    cI = np.where(pI==0.0)[1]
    LcI = 0

    # Neighbour distribution
    pO = A.sum(axis=1)
    cO = np.where(pO==0.0)[0]
    pO[cO] = 1
    A[cO,:] = np.ones((len(cO),n))/n
    N = A/pO[:,0]

    # Propagation matrix
    S = np.ones((n,n))/n
    P = (1-alpha)*N + (alpha)*S

    # Expected length absorbing walk
    Lc = np.zeros(n)

    for c in range(n):
        # Check value non-incoming nodes
        if ((c in cI) & (LcI == 0)) | (c not in cI):
            # Calculating ARW
            Pc = np.delete(P, c, 0)
            Pc = np.delete(Pc, c, 1)
            I = np.diag(np.ones(n-1))
            F = np.linalg.inv(I-Pc)
            L = np.matmul(F,np.ones(n-1))

            Lc[c] = np.mean(L)
            if c in cI:
                LcI = Lc[c]
        else:
            Lc[c] = LcI

    Lc_sum = np.sum(Lc)
    Lcw = Lc/Lc_sum
    return Lcw
```

B.4 Snowball sampling

```
#### 16 May 2019
#### A. van Schetsen
#### Snowball sampling

import random
import networkx as nx

class Queue():
    def __init__(self):
        self.queue = list()
```

```
    def enqueue(self,data):
        if data not in self.queue:
            self.queue.insert(0,data)
            return True
        return False

    def dequeue(self):
        if len(self.queue)>0:
            return self.queue.pop()
        else:
            #plt.show()
            exit()

    def size(self):
        return len(self.queue)

    def printQueue(self):
        return self.queue

def snowball(Gd,size,k_start,k,min_comp):
    G = Gd.to_undirected()
    q = Queue()
    m = k_start
    max_size = 0
    dictt = set()

    while(m):
        id = random.sample(list(G.nodes()),1)[0]
        max_size = max_size + len(nx.descendants(G,id))
        q.enqueue(id)
        m = m - 1

    if max_size < size:
        size = max_size

    while(len(dictt) < size):
        if(q.size() > 0):
            id = q.dequeue()
            if(id not in dictt):
                dictt.add(id)
                neighbors = set(G.neighbors(id))
                neighbors.difference_update(dictt)
                list_neighbors = list(neighbors)

                if(len(list_neighbors) > k):
                    for x in random.sample(list_neighbors,k):
                        q.enqueue(x)
                elif(len(list_neighbors) <= k and len(list_neighbors) > 0):
                    for x in list_neighbors:
                        q.enqueue(x)
            else:
                continue
        else:
            initial_nodes = random.sample(list(dictt),k_start)
            for id in initial_nodes:
                dictt.remove(id)
                q.enqueue(id)

    print('snowball␣in␣component:␣'+str(max_size))
    return Gd.subgraph(dictt)
```

B.5 Extracting window features

```
#### Window feature engineering
#### 1 May 2019

import networkx as nx
import pandas as pd
import numpy as np
import datetime
from collections import Counter

# ============================================================================
# Bitcoin MIT
# ============================================================================

Data = pd.read_csv("DTpp.csv").drop(columns='Unnamed: 0')
price = pd.read_csv("pp_prices_day.csv").drop(columns='Unnamed: 0')

# ============================================================================
# Sample data
# ============================================================================

# Labels
labels = list(['start_window','start_timestamp','stop_window',
    'stop_timestamp','price','users','transactions'])
labels = labels + list(['mean_transaction_value','median_transaction_value'])
labels = labels + list(['avg_inout_degree','median_degree','alpha_power_law'])
labels = labels + list(['new_addresses','transaction_new_addresses'])
labels = labels + list(['prediction_date','prediction_timestamp'])

pi = 1     # interval of price prediction
wi = 2    # interval of window
features = pd.DataFrame(columns=labels)
i = 0
k = 100

wstart =  datetime.datetime(2015,12,1)
wstart_actual = wstart.timestamp()
wstop = wstart + datetime.timedelta(days=wi)
wstop_actual = wstop.timestamp()

# Max Address ID up to that point
df = Data[(Data['timestamp']<wstart_actual)]
users = pd.unique(df[['in_addrID','out_addrID']].stack())
maxID = max(users)

while wstop_actual <= datetime.datetime(2017,8,2).timestamp():
    print(wstop)
    f_temp = list([wstart,wstart.timestamp(),wstop,wstop_actual])
    df = Data[(Data['timestamp']>=wstart_actual)&(Data['timestamp']<wstop_actual)]
    df = df.groupby(['in_addrID','out_addrID'])['amount'].agg('sum').reset_index()

    # price
    f_temp = f_temp + list(price[price['timestamp']==wstop_actual]['price'])

    # Contruct the directed graph
    Gd = nx.from_pandas_edgelist(df,source='in_addrID',target='out_addrID',
            edge_attr='amount',create_using=nx.DiGraph())
    n, m = Gd.number_of_nodes(), Gd.number_of_edges()
    f_temp = f_temp + list([n,m])

    # Transaction value stats
    trans_value = df['amount']
    f_temp = f_temp + list([int(trans_value.mean()),int(trans_value.median())])
```

```
    # Degree stats
    degree = list(dict(Gd.degree).values())
    f_temp.append(float(sum(dict(Gd.in_degree).values()))/n)
    f_temp.append(int(np.median(degree)))

    deg_count = Counter(degree)
    deg_labels, deg_values = zip(*sorted(deg_count.items()))
    deg_z = np.polyfit(np.log(deg_labels),np.log(deg_values),1)
    deg_p = np.poly1d(deg_z)
    f_temp.append(-deg_z[0])

    # New users
    users = pd.unique(df[['in_addrID','out_addrID']].stack())
    users_new = users[users>maxID]
    if len(users_new) > 0:
        maxID = np.max([maxID,max(users_new)])
    f_temp.append(len(users_new)/n)
    df_new = df[(np.isin(df['in_addrID'],users_new))|
            (np.isin(df['out_addrID'],users_new))]
    f_temp.append(len(df_new)/m)

    # Prediction date
    p = wstop + datetime.timedelta(days=pi)
    f_temp = f_temp + list([p,p.timestamp()])

    # Save in overall data frame
    features.loc[i] = f_temp
    i += 1

    # Next window
    wstart =  wstart + datetime.timedelta(days=pi)
    wstop = wstop + datetime.timedelta(days=pi)
    wstop_actual, wstart_actual = wstop.timestamp(), wstart.timestamp()

features.to_csv('pp_windowfeatures_day.csv')
```

B.6 Extracting node-based features

```
#### Node-based features
#### 17 May 2019

import networkx as nx
import pandas as pd
import datetime
import numpy as np

import snowball   # code of section 'Snowball sampling'
import absorbing  # code of section '1-ARW-betweeness centraily measure'

# ============================================================================
# Bitcoin MIT / Bitcoin Coinbase
# ============================================================================

Data = pd.read_csv("DTpp.csv").drop(columns='Unnamed: 0')
Data_price = pd.read_csv("pp_prices_day.csv").drop(columns='Unnamed: 0')

# ============================================================================
# Parameters
# ============================================================================

pi = 1      # interval of price prediction
wi = 2      # window size
k = 100     # Min size graph component for node features

start = datetime.datetime(2016,12,30)
stop = datetime.datetime(2017,1,2)

alpha = 0.15         # Restart probability ARW
ss = 1000            # Max snowball graph size
ks = 1000            # Wide search snowball
ns = 50              # Number of sampled snowball graphs

validation = True    # Construct the validation dataset

# ============================================================================
# Data storage
# ============================================================================

labels = list(['nodeID','time','iteration'])
labels = labels + list(['in_degree','out_degree','ARW',
                        'PageRank','closeness_centrality',
                        'total_flow','netto_flow','predict'])
RF = pd.DataFrame(columns=labels)

# ============================================================================
# Algorithm
# ============================================================================

wstart =  start
wstart_actual = wstart.timestamp()
wstop = wstart + datetime.timedelta(days=wi)
wstop_actual = wstop.timestamp()
j = 0

while wstop_actual <= stop.timestamp():
    predict_time = wstop + datetime.timedelta(days=pi)
    predict=int(Data_price[Data_price['timestamp']==predict_time.timestamp()]['sign'])
    print(predict_time)

    df = Data[(Data['timestamp']>=wstart_actual)&(Data['timestamp']<wstop_actual)]
```

```
    df = df.groupby(['in_addrID','out_addrID'])['amount'].agg('sum').reset_index()

    # Filter out components smaller than k
    G = nx.from_pandas_edgelist(df,source='in_addrID',target='out_addrID')
    components = list(nx.connected_components(G))
    nodes_large = set()
    for comp in components:
        if(len(comp)>=k):
            nodes_large = nodes_large | comp
    df = df[np.isin(df['in_addrID'],list(nodes_large))]

    # Contruct the directed graph
    Gd = nx.from_pandas_edgelist(df,source='in_addrID',target='out_addrID',
        edge_attr='amount',create_using=nx.DiGraph())
    n, m = Gd.number_of_nodes(), Gd.number_of_edges()

    # Snowball sampling
    for s in range(ns):
        print(s)

        # Gsnowball = snowball.snowball(Gd,ss,1,ks)
        Gsnowball = snowball_fillup.snowball(Gd,ss,ks,k)
        A = nx.to_numpy_matrix(Gsnowball, nodelist=None, weight='amount')

        Dsnowball = pd.DataFrame(list(Gsnowball.nodes()),columns=['nodeID'])
        Dsnowball['time'] = predict_time
        Dsnowball['iteration'] = j
        j += 1

        # Measures
        print('Measures␣1')
        Dsnowball['in_degree'] = [val for (node, val) in
                Gsnowball.in_degree()]
        Dsnowball['out_degree'] = [val for (node, val) in
                Gsnowball.out_degree()]
        Dsnowball['ARW'] = absorbing.ARW(A,alpha)
        Dsnowball['PageRank'] = nx.pagerank(Gsnowball,1-alpha).values()
        Dsnowball['closeness_centrality'] = nx.closeness_centrality(Gsnowball).values()

        print('Measures␣2')
        in_amount = np.array([val for (node, val) in Gsnowball.in_degree(weight='amount')])
        out_amount = np.array([val for (node, val) in Gsnowball.out_degree(weight='amount')])
        Dsnowball['total_flow'] = np.maximum(in_amount,out_amount)
        Dsnowball['netto_flow'] = in_amount-out_amount

        Dsnowball['predict'] = predict
        RF = RF.append(Dsnowball)

        if (s+1) %50 == 0:
            RF.to_csv('RF_temp.csv')


    # Next window
    wstart =  wstart + datetime.timedelta(days=pi)
    wstop = wstop + datetime.timedelta(days=pi)
    wstop_actual, wstart_actual = wstop.timestamp(), wstart.timestamp()
```

B.7 Fairness and goodness measure

```
def initiliaze_scores(G):
    fairness = {}
    goodness = {}

    nodes = G.nodes()
    for node in nodes:
        fairness[node] = 1
        goodness[node] = 0
    return fairness, goodness

def compute_fairness_goodness(G):
    fairness, goodness = initiliaze_scores(G)
    nodes = G.nodes()
    iter = 0
    while iter < 100:
        df = 0
        dg = 0

        print("------------------")
        print("Iteration␣number", iter)

        print('Updating␣goodness')
        for node in nodes:
            inedges = G.in_edges(node, data='weight')
            g = 0
            for edge in inedges:
                g += fairness[edge[0]]*edge[2]

            try:
                dg += abs(g/len(inedges) - goodness[node])
                goodness[node] = g/len(inedges)
            except:
                pass

        print('Updating␣fairness')
        for node in nodes:
            outedges = G.out_edges(node, data='weight')
            f = 0
            for edge in outedges:
                f += 1.0 - abs(edge[2] - goodness[edge[1]])/2.0
            try:
                df += abs(f/len(outedges) - fairness[node])
                fairness[node] = f/len(outedges)
            except:
                pass

        if df < math.pow(10, -6) and dg < math.pow(10, -6):
            break
        iter+=1

        # Find missing values
        missing = {}
        missing['fairness'] = list()
        missing['goodness'] = list()
        for node in nodes:
            if G.out_degree(node) == 0:
                missing['fairness'].insert(0,node)
            if G.in_degree(node) == 0:
                missing['goodness'].insert(0,node)

    return fairness, goodness, missing
```