Universiteit Leiden

Master Computer Science

Graph pattern mining for blockchain networks

Name:	Atish Kulkarni
Student ID:	S2483122
Date:	27/07/2021
Specialisation:	Data science
1st supervisor:	Iris Yocarini
2nd supervisor:	Matthijs van Leeuwen

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Contents

Abstract

Addressing the growth of the field of Blockchain and cryptocurrencies, this study aims to analyse datasets of two cryptocurrencies, Bitcoin and Ethereum, for suspicious transaction activities. We represent the Blockchain data as a graph network and execute graph pattern mining methods to search for suspicious transaction patterns. We developed two graph pattern mining algorithms to discover suspicious transaction patterns following a circular or diamond-shaped structure. This research also conducts a quantitative evaluation of patterns observed and investigates their relation with Blockchain metrics. To assess the observed patterns, we propose a novel locality sensitive hashing-based method named relevance evaluation. The relevance evaluation is a statistical evaluation method for graph patterns. The relevance evaluation conducts various statistical tests and helps to classify graph patterns as usual or unusual observations. Results of this study unveil the relation between suspicious transaction activities such as money laundering and cryptocurrencies.

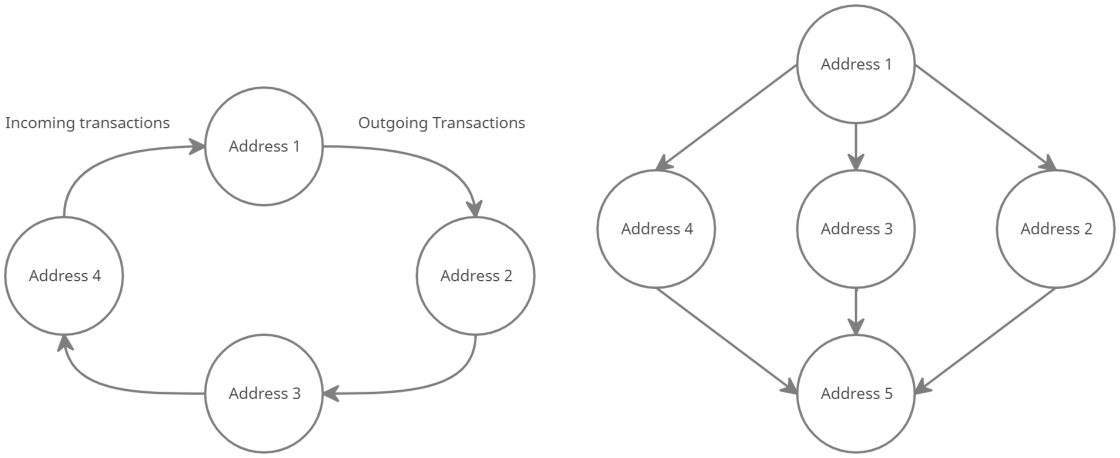
1 Introduction

In the 21st century, the impact created by the new emerging technologies has changed the structure of everyday services. The nature of financial services is on the verge of change due to the introduction of the new Blockchain technology. Digital currencies created on Blockchain technology, also known as cryptocurrencies, are becoming part of everyday finance. Similar to fiat currencies like the US dollar and Euro, cryptocurrencies can store value and are transferable across users. Such transfers create a graph network of transactions between users of cryptocurrencies. This research aims to find graph network patterns that are candidates for suspicious activities like money laundering and test their relevance. Candidate patterns are discovered by applying graph pattern mining techniques on graph networks created by transactions of Blockchain-based cryptocurrencies.

A graph network is a data structure that explains the relationship between all of its data points. The graphs discussed in this study are created using transaction data; hence, they are weighted directed graphs. A directed edge connects a pair of nodes in a weighted directed graph network. Each such edge has a timestamp and a weight associated with it. Explanation of a graph can be simplified with an example of a transaction where person A transfers x amount of money to person B on date y. If a graph represents this information, person A and B are the nodes, the transaction from A to B is a directed edge, the amount transferred x is the weight, and date y of transfer is the timestamp. This example shows a single directed weighted edge existing between two nodes, representing a single transaction. Various such transactions between unique users create a graph network. The study conducted in [1] shows an example to create a graph network for transaction data of cryptocurrencies. Node, edge, weight, and timestamp are the components of the graph network that are relevant in this study to design graph pattern mining algorithms.

Graph pattern mining is defined as the process of finding relevant sub-graph structures in a graph. Existing methods such as machine learning classifiers like SVM, neural network classifiers like multi-layer perceptrons, probabilistic neural networks and graph-based data mining methods like frequent pattern mining are compared in [2]. Comparison in [2] shows that graph pattern mining is the most preferred method for discovering suspicious patterns. In graph pattern mining conditions for the structure of the pattern, weight and timestamps are defined. These conditions limit the number of patterns discovered and also preserve the quality of the patterns discovered. Graph pattern mining can be roughly divided into two types, frequent pattern mining and fixed structural pattern mining. In the case of frequent pattern mining, the aimed structure of the pattern is not predefined. Frequent pattern mining algorithms discover the patterns based on the frequency of the patterns in the dataset. However, in fixed structural pattern mining, algorithms discover patterns based on the predefined structure. Previous research [3] [4] has shown the involvement of suspicious activities like money laundering is associated with patterns having structural similarity with circular and diamond shape. In circular patterns, an amount is transferred among different cryptocurrency users with the condition that the source of this transaction gets a similar amount back. In diamond patterns, the source of the transaction distributes the amount in n-parts to n-addresses. Later, all of these n-addresses contribute these amounts to a single address. Figure 1, (a) illustrates an example of a circular pattern and (b) shows a diamond pattern, respectively. To find circular and diamond patterns, we designed two fixed structural pattern mining algorithms. Algorithms

circle track in Chapter 5.1 and diamond track in Chapter 5.2 are designed for finding circular and diamond patterns in cryptocurrency transaction data.



(a) (b)

Figure 1: An example of a circular pattern is shown in (a) and (b) shows a diamond pattern.

Graph structures of circular patterns, diamond patterns and the basic terminology used in this study can be understood based on example patterns in the Figure 1. In the Figure 1 (a) nodes are Address 1, Address 2, Address 3, Address 4. Each directed arrow between a pair of nodes is a directed edge. A node that initiates the first transaction in a pattern as a sender is termed the source node. Figure 1 (a) demonstrates a circular pattern where Address 1 is the source node. The node that is the receiver for the last transaction in a pattern is termed the sink node. For a circular pattern, the source node and sink node are always the same. Nodes separated by a single edge are first degree neighbours, and nodes separated by two edges are second degree neighbours. For example, for node Address 1 in Figure 1 (a), nodes Address 2 and Address 4 are first degree neighbours, and Address 3 is the second degree neighbour. Based on the direction of an edge, for node Address 1 in Figure 1 (a), node Address 2 is the outgoing neighbour, and node Address 4 is the incoming neighbour. Another term used in this study is a support node, which is defined as the second degree neighbour of a source node. For Source node Address 1 in Figure 1 (a), Address 3 is the support node. Following the same description, for Figure 1 (b), Address 1 is the source node, Address 5 is the sink node. For Source node Address 1, Address 5 is the support node. The circular and diamond pattern can be divided into three layers where the source node is in layer 1, the support node is in layer 3 and all other nodes are in layer 2. In Figure 1 (a) Address 1 is in layer 1, Address 2, Address 4 are in layer 2 and Address 3 is in layer 3. Figure 1 (b) can be divided into three layers, where Address 1 is in layer 1, Address 2, Address 3, Address 4 are in layer 2, and Address 5 is in layer 3. This research aims to investigate circular and diamond pattern formations in two real-world datasets, the Bitcoin and Ethereum Blockchain networks, that might be linked with suspicious behaviours.

Our algorithms circle track and diamond track are inspired by the pattern mining approaches

discussed in [5]. Comparison of approaches in [5] analyses algorithms that find frequent patterns without any user-specific structural input. Approaches discussed in [5] use different methods for traversing datasets and find frequent occurring patterns above the user-defined frequency threshold. In contrast to this, algorithms designed in this study have a structural target for the patterns. In this study, we did not integrate the Apriori-based approach [6] that was the earliest in the field of graph pattern mining. Apriori approach is not suitable, as weight assumptions in Apriori based algorithms are incompatible with our dataset. Each edge in cryptocurrency transaction data most probably has a unique weight associated with it. However, Algorithms based on the Apriori approach such as Apriori, AprioriTid, and AprioriHybrid assume equal weight on every edge observed in the dataset. Along with this, Apriori-based algorithms use a breadth-first search approach to traverse the graph. Such traversing requires multiple scans of the dataset. Multiple scans result in higher time and memory complexity. The problem of having high time complexity was also observed in the early versions of the diamond track algorithm. We split the data into n smaller chunks in the diamond track algorithm to solve time and memory complexity issues. The concept of splitting the data in multiple smaller chunks is inspired by partition-based algorithms discussed in [5]. Partition-based algorithms require only a few scans compared to Apriori-based algorithms to find out frequent patterns. Results in [5] show that the partition-based approach addresses time and memory complexities and reduces them substantially. We also took a hybrid approach of combining breadth-first search (BFS) and depth-first search (DFS) and intersection counting. The hybrid approach yields multiple circular patterns for a pair of the source node and support node in a single search of the dataset. The study [7] shows the benefits of a hybrid approach that combines BFS, DFS, and intersection counting. Results of [7] show that intersection counting is a simple operation, and as no complex data structure is required and time and memory complexity is low. Compared to patterns discussed in studies [8], [4], [1] in our study we do not define a fixed geometrical structure for the patterns to be discovered. Algorithms are designed to find patterns similar to circular and diamond shapes based on structure, weight, and timestamp conditions. All the patterns that follow conditions are considered candidate patterns. Such flexible approach allows us to explore patterns discussed in [8], [4], [1] and as well as some more complex patterns as shown in Figure 28.

Once potential suspicious patterns are discovered, there is a need to classify them as malicious transaction activities or routine transactions. Even if we assume all discovered patterns are suspicious, there is a necessity to evaluate the risk associated with patterns, as each transaction pattern has a different monetary value. Graph pattern mining is often combined with other supporting methods such as significance testing and neural network classifiers to evaluate the discovered patterns. Following the same concept, we aim to combine the graph pattern mining method with the relevance evaluation. The term *usual patterns* refer to patterns created in a Blockchain network due to common transactions. Opposite to this, the term *unusual pattern* refers to patterns that might result from suspicious transaction activities. Evaluation of discovered patterns under the relevance evaluation helps to classify them as usual or unusual observations based on the risk associated with them. In [8] under significance testing, Chrysanthi Kosyfaki et al. have classified patterns as usual or unusual based on randomization test of weights involved in the pattern but keeping the structure of the patterns same. As the structure of patterns was kept the same, significance testing in [8] only evaluates the probability of observing weights in a particular order. However, in any suspicious activity, the pattern's structure is the primary problem rather than its weights. Hence, we present the rel-

evance evaluation; a novel Locality sensitive hashing (LSH) based approach to evaluate the risk associated with patterns and to classify them as usual or unusual observations. Relevance evaluation evaluates patterns based on the structure as well as weights. The results of the relevance evaluation help to identify patterns with high monetary and structural risk.

The selection of Bitcoin and Ethereum data for searching suspicious patterns was made based on their total market capitalisation. These two cryptocurrencies hold the largest market capital share; hence, analysis of their transaction data can reveal important insights of their networks. Although cryptocurrencies have seen an exponential rise, data of cryptocurrencies has not been studied to full potential. Studies like [9] [8] have explored some patterns in transaction dataset of Bitcoin. However, patterns studied on fiat currencies discussed in [4] [3] such as large circular graphs, and k-partite graphs are not searched in data sets of Bitcoin and Ethereum. Hence, in this study, we analyse Bitcoin and Ethereum transaction data for circular and diamond patterns. Studying Bitcoin and Ethereum can help explore whether money laundering patterns are evident in Blockchain transactions. This research also aims to study the patterns observed for their frequency, size, co-relation with other Blockchain metrics, and relevance. Results of this study can help to address the concerns of governing authorities. This study should also help to fact-check a common belief about cryptocurrencies having many suspicious activities. The results of this study help to answer the following questions.

1. Are money laundering patterns evident in Blockchains of Bitcoin and Ethereum?
2. How many patterns are observed within a time constraint of 7 days?
3. What is the size of patterns observed?
4. What kind of monetary fluctuations are observed in suspicious patterns?
5. How to evaluate transaction patterns based on amount and structure?

In the rest of the paper, we first introduce the background of Blockchain, Bitcoin, Ethereum, and LSH in Chapter 2. Literature review in the Chapter 2.3 gives brief information on studies referred and inspiration for the algorithms. Information about data of Bitcoin and Ethereum and other statistics associated with data are discussed in the Chapter 3. Problem statements in the Chapter 4 shows the formal representation of the patterns aimed in this study. Steps involved in algorithms and their working is explained in the Chapter 5. The chapter 5.3 explains the novel approach of LSH for relevance testing. Results for algorithms and relevance test are shown in Chapter 6. Finally, in Chapter 7, we discuss the results and conclude the paper in Chapter 8. The Table 15 in the Chapter 9.1 explains graph network related terms used in this study and as well as some Blockchain concepts.

2 Background and related work

This chapter presents information about the structure of Blockchain, its properties, and factors to be considered while representing it as a graph. This chapter also briefly explains the working of LSH and the framework used to implement LSH in this study. In this chapter, we also discuss existing studies related to graph pattern mining, money laundering tracking and Blockchain. We briefly explain the approach and methods of existing studies and their motivation for our study.

2.1 Blockchain

2.1.1 Blockchain basics

Blockchain is one of the new emerging technologies in the field of database storage. The primary difference between traditional data storage and Blockchain is evident when Blockchain structure is considered. Traditional data storage has four types of operations create, read, update, and delete (together known as CRUD operations). In contrast to this, Blockchain has only two types of operation, read and write. Data in the Blockchain cannot be modified or deleted after being created. Moreover, depending on the time and memory threshold, data is grouped and put together in a single block. Each such block has a header known as the hash of that block. In every block, a hash of the previous block is stored. This storage of the hash of the previous block helps to keep track of all previous transactions. Hence, as the name implies and shown in the Figure 2, the chain of blocks is held together by hashes of different blocks. In the Figure 2, Each square represents a block and bi-directional arrows represent the link created between the blocks. Each block of Blockchain has a memory limit. Every block stores all transactions within memory limits and time threshold limits. Exceeding memory or time thresholds results in the creation of new blocks. Such multiple blocks are combined to create a weighted directed graph network.

2.1.2 Properties of Blockchain transactions

- Transaction structure: In this study, we focus on the transaction data of Blockchain. Similar to traditional transactions, Blockchain transactions always have senders, receivers, amounts, and timestamps. Cryptocurrencies are stored in a wallet represented by an alphanumeric string holding a certain amount of a cryptocurrency. This alphanumeric string is also known as a wallet address. Each unique wallet address acts as a node when the Blockchain is represented as a graph. Unlike most network graphs, each unique node in a Blockchain does not represent a unique user or identity. Two or more nodes may represent the same user having multiple wallet addresses. Therefore, unlike fiat currency accounts, ownership of any cryptocurrency wallet cannot be associated with a particular identity easily. The study conducted in [10] has tried to group wallet addresses of cryptocurrencies and identify their owners. We do not consider such grouping as grouping nodes together may result in a reduced number of patterns.

- Transaction speed: The transactions in a Blockchain are faster than some traditional bank transactions. The average transaction time for Bitcoin is about 2 hours, and for Ethereum, it is less than 10 minutes. The transaction time depends on network difficulty, the number of miners, and the total hash rate on the Blockchain.

- Monetary value: The trading of cryptocurrencies also influences their prices and makes them highly volatile when compared to fiat currencies. The volatile nature of cryptocurrencies influences the monetary value represented by a pattern.

Anonymity, varying transaction time, and volatile value of cryptocurrency are the unique properties to be considered while representing the Blockchain data as a graph. These factors can also influence the pattern formation in the Blockchain network as they are co-related (as shown in Figures: 26, 24, 25, 27). Due to unique properties, transaction data of Blockchain is

processed on a weekly basis in this study. If any money laundering occurs using these cryptocurrencies, then the time frame of all the transactions involved in a money-laundering pattern should be ideally narrow. If the time frame is wide for the transactions involved in a pattern, it will not go along with the volatility associated with an asset. In 2017, the price of Bitcoin was changing more than 10% a week. If a pattern takes a long time to occur, then the value sent and the value received would be substantially different. The study conducted in [8] shows a similar concept where transactions of Bitcoin were searched for patterns for a sliding time window.

2.1.3 Importance of investigating cryptocurrencies

Study of cryptocurrency transactions is essential as the field of Blockchain and cryptocurrencies has seen exponential growth in the past few years. Compared to fiat currencies, transactions of cryptocurrencies are faster, secure, and have no limit on the amount. Such advantages over fiat currencies have helped to catch the attention of different industries and have spiked the adoption of Blockchain and digital currencies [11] [12]. Digital fiat currencies like Tether and USDC are created on top of Ethereum, providing users stable value digital currencies. These stable value digital currencies replicate the government-issued currencies like the dollar on the Blockchain. At the time of this study, the total market capitalization of all digital currencies together exceeds 1 trillion dollars. The growing adoption and market capitalization of digital currencies have created a need to study them in the same way as fiat currency.
Although the adoption of Blockchain and digital currencies has many advantages, it comes with its challenges too. The anonymity associated with the users of cryptocurrencies has always been a topic of debate. Studies like [13] have discussed these issues in detail and have tried to identify users based on their transaction behaviour. While many developed European and western countries have allowed the possession of Bitcoin, some developing countries such as India and China have been trying to ban or heavily regulate Bitcoin. Concerns of large financial entities and countries about cryptocurrencies are associated with the anonymity of cryptocurrencies, their use in the grey or black market, and their possible use in tax evasion and money laundering.

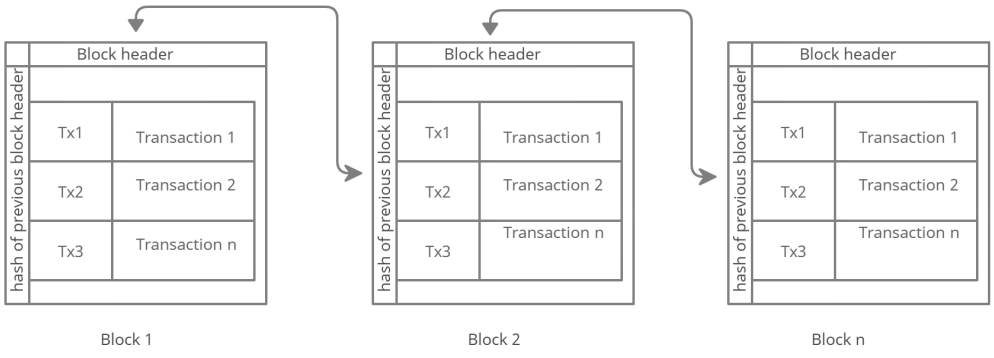


Figure 2: Example of Blockchain structure. Bi-directional arrows represent link created by block header connections.

2.2 Locality Sensitive hashing

Locality sensitive hashing is one of the most prominent methods of comparing individual elements in the data set [14]. Comparing elements, LSH calculates similarity based on either the cosine distance or the Euclidean distance. LSH uses a family of hash functions [15], which groups the elements from the dataset based on the similarity between them. Hash functions of LSH are tunable and can be adjusted according to the purpose of the similarity calculation. Comparing an element with all other elements of a dataset in a usual iterative way has a time complexity of $O(n^2)$. Using LSH, this time complexity of comparing elements can be reduced to $O(n)$. LSH does not compute exact cosine or Euclidean distance, but performs approximate computation. The use of LSH has the best results when each element in the dataset has many features, and the total number of elements is also high. Similarity obtained using LSH has been previously used to find and filter duplicates or to get the closest match to a small data sample. Popular music recognition apps like Shazam are based on LSH. Although LSH is widely used in image search and text mining, it is not very well explored on graph network datasets. Our evaluation method in Chapter 5.3 uses LSH for similarity calculation.

2.2.1 LSH workflow

LSH workflow is described in [16] in detail and is broken down into three main sub-steps as follows

1. *Shingling:* In this step, each item in the data set is represented as a set of elements. In randomly generated graphs, each graph will be converted into a vector using the total number of possible overall edges and the amounts observed in the pattern. This vectorization method ensures that the length of each randomly generated graph vector will be the same, making the calculation of Euclidean distance easier.

2. *Min-Hashing:* Min hashing helps to convert shingle sets into hashes or signatures. The fundamental idea behind min-hashing is that every column is hashed in such a way that similarity is preserved. Thus, if the similarity is high for two columns, then there is a high probability that both columns should fall into the same sub-set. Moreover, if the similarity of two columns is low, then there is a high probability that both columns should not fall into the same sub-set.

3. *hashing:* Using the signatures, the vector similarity of the discovered pattern with all the random patterns is calculated. The query vector is compared with the stored vectors in the data set for this computation. If the pair has the same hash or falls in the same sub-set more than n number of times, such pair is saved as a candidate for similarity match. Euclidean distances are calculated between vectors of n such matches, and results are ranked according to the Euclidean distance.

2.3 Related work

In this subsection, we briefly discuss existing pattern mining studies conducted on Bitcoin Blockchain. We discuss similarities and differences between our study and previous studies. Studies discussed in this subsection help to understand our choices regarding pattern selection, representation of Blockchain, algorithm approach and evaluation approach. Selection of

diamond and circular patterns is based on patterns discussed in studies [17], [3], [8]. Representation of Blockchain as weighted directed network graph is inspired from [1]. The approach taken while designing algorithms circle track and the diamond track is based on a comparison of approaches in [5]. The evaluation approach in the relevance evaluation is a combination of randomization test in [8] and weighted similarity calculation in [18]. The selection of LSH to compare a pattern with random patterns is based on [14]. Details of each of these studies and their relation to our study are explained below.

The study conducted in [1] shows how data from blocks of Bitcoin Blockchain is processed as graphs. The study [1] also discusses the different types of transactions. Similar to this study, C. Cachin et al. [1] have also conducted their study on the Blockchain of Bitcoin and Ethereum. The terminology used in this study and their study is similar, and hence [1] can be referred to explore the components of Blockchain in detail. Some core concepts such as Bitcoin mining, Unspent Bitcoin transactions outputs are not discussed in this study due to the primary focus on pattern mining. These concepts and their relation with graph construction are explained in [1]. Their study showed how the structure and components of Blockchain contribute and affect graph creation. In our study, we have taken inspiration from their methods while constructing graphs of Blockchain transactions.

In [17] along with transaction pattern mining, Bitcoin network statistics about the network of Bitcoin and overall wallet balance are discussed. Their research found that more than 70% of Bitcoin wallet addresses are associated with the biggest exchange of that time, known as Mt. Gox. As the Bitcoin Blockchain has grown exponentially over the decade, data Statistics discussed in [17] differ from current Bitcoin data. In their study, an investigation about long-chain transaction patterns was conducted. A long-chain transaction pattern is where a large amount of Bitcoin is forwarded numerous times with tiny decay in amount every time it is forwarded. Such types of transactions were also a point of interest for malicious activity tracking. Although, over the period, long-chain transactions can grow exponentially with many branches. Such chains extend beyond the one-week time frame and hence are not considered in this study. Tracking such chains is informative in the case of small networks. Study [17] was conducted in 2012; hence, limited data was available for Bitcoin. Observations such as addresses are majorly associated with a single exchange are not valid due to the exponential growth of the Blockchain field. Hence, it is not possible to analyse transactions while keeping an exchange as a central point. Concepts such as Fork-Merge patterns and self-loops are discussed in [17]. Fork-Merge patterns are patterns similar to diamond patterns and have splitting of amount into multiple addresses and self loops are similar to circularpatterns, but sender and receiver is the same. Based on the similarity of patterns in [17] and our study, we studied pattern structures and made the selection of circular and diamond patterns.

In [3] authors Chen Zhao and Yong Gu, have processed Bitcoin transaction networks as graphs. In this study, the basic concepts of transactions and their nature is studied. The study discusses two possible patterns involved in money laundering activities. The first one being a cyclic pattern as shown in 1 (a) and the second one being not a very structure-specific pattern. As demonstrated earlier in Figure 1(a), in circular patterns the source node is also the sink node; thus the amount is forwarded back to the first node of the pattern. The forwarded amount has little or no decay while being transferred to the next node. Study [3] discusses an idea where a suspicious transaction may have an amount splitting in smaller chunks and then sent

to multiple nodes. This is very similar to the structure illustrated in Figure 1 (b). As observed in the results of [3] in suspicious patterns, a large amount is often split into a large number of smaller chunks. Based on this observation, we decided not to limit circular and diamond patterns structurally.

Study [19] has similar research which tries to identify money laundering patterns and addresses involved in them. For this identification, machine learning models are used along with a labelled data set. In this study, a specific 2-motif pattern referenced as "short thick bands" (STB) is considered a potential candidate for the money laundering activity. The structure of STB can be defined as a hybrid of circular and diamond patterns. STB pattern merges circular pattern and diamond pattern to create a complex sub-graph termed as directed hypergraph in [19]. Although the research theme is similar to this study, the primary focus of authors Stephen Ranshous et al. is classifying suspicious addresses and not on suspicious transactions. Studying the structure of the STB pattern directed our study to consider circular and diamond patterns. Studying STB patterns also helped in formulating conditions for circle track and diamond track algorithms.

Apart from Bitcoin, traditional bank data sets have been similarly studied for pattern mining. Research done in [4] tries to extract k-partite graphs. A k-partite structure is a graph structure where the source distributes the amount into k candidates. The number k, in this case, can be any number more than a certain threshold number of nodes required. Taking a similar approach in our study, we designed an algorithm that can extract patterns having a diamond shape as shown in Figure 1 with a minimum of 2 nodes in the second layer. In [20] authors Andrea Fronzetti Colladon and Elisa Remondi have analysed an Italian factoring company's data for money laundering activities. Their study uses network analysis metrics such as centrality, in-degree, out-degree to construct graphs of transactions. Further, clustering graph, they classify transactions as suspicious and non-suspicious. In our study, the concept of layers in a pattern and the idea of not putting any limit on the number of nodes in layer 2 are inspired from [4].

Authors Wegberg et al. [9], have studied Bitcoin's involvement in money laundering activities. In their study, authors have monitored the types of services used. Some mixing services are available in the Bitcoin network, which helps merge multiple small transactions into one transaction. Mixing makes the flow computation hard as the exact transferred amount is not precise. If there is the involvement of any mixer service, then it signals a possible malicious activity alert. Along with this, the internet remains an essential need for any cryptocurrency transaction. Authors have tried to monitor different aspects such as type of browser, popularity, or reviews of the mixing services, and availability of personal details. Hence, authors have tried to use details available outside of Blockchain data to find malicious activity in their study. Although the research theme is different, the fundamental idea of money laundering and how it is executed was studied using [9].

The study conducted in [8] is similar to this study, as it also considers data from Bitcoin Blockchain to find pattern formations. Authors Kosyfaki et al. have created an algorithm to find different types of small sub-graphs in a graph network. For this search of the patterns, the authors consider a flow threshold for every edge present. Following the same idea, in this study, we have used a fluctuating threshold. As the prices of cryptocurrencies are volatile, we choose

a threshold based on the weekly price of the cryptocurrency. Using this fluctuating threshold, all the transactions above 3000 dollars are searched for the patterns. Patterns mined in [8] do not have a fixed geometrical structure. In contrast to this, we focus on patterns with either circular or geometric structures in this study. Significance testing conducted in [8] randomizes the flow observed while keeping the structure of the pattern the same. Keeping the structure the same helps test the significance of the flow observed, however, not the structure observed in the pattern. Taking inspiration from this method, we designed a method that conducts tests of relevance. Relevance evaluation uses methods like LSH along with the similarity test, skewness test and kurtosis test. Unlike significance testing, the relevance evaluation helps to classify patterns as usual or unusual observations based not only on flow but also on the pattern's structure. Study in [8] uses sliding time windows for tracking patterns. The maximum time window considered in [8] is of 900 seconds or 15 minutes. Such a small-time window may serve best for patterns described in [8]. However, considering patterns like Figure 1 and average transaction time of 1 hour for Bitcoin in 2016-2017, we consider larger time windows of 7 days. Studying [8] helped in selecting a time window for a data split and inspired the creation of the relevance evaluation.

To study various pattern mining approaches, we referred to the study conducted in [5]. Total five types of approaches are discussed in [5] i.e., Apriori based Algorithms, Partition-based Algorithms, hybrid DFS, Pattern Growth Algorithms and SQL-based algorithms. Out of these four approaches, Partition-based Algorithms, SQL-based algorithms, and hybrid DFS are relevant for our study as our algorithms are based on similar concepts. Based on hybrid DFS, we created algorithms that find neighbours of a pair of source nodes and support nodes in parallel. Similar to Partition-based Algorithms, in the diamond track algorithm of this study, we process data in a graph by dividing it into n parts. Dividing data helps to confirm the discovery of a pattern in the early stages of the algorithm. Taking a subset of the data and cleaning step discussed in both algorithms of this study is based on SQL-based algorithms in [5]. Although [5] provides an in-depth comparison of different approaches, most of the algorithms based on these approaches are designed for frequent pattern mining. Similarly, python packages like Stanford Network Analysis Platform (SNAP) and networkx are highly scalable and efficient but do not take custom input for pattern mining. Hence, in this study, we created our algorithms to find patterns shown similar to Figure 1 independent of these packages. The primary aim of these algorithms is to find patterns in the datasets of Bitcoin and Ethereum. Hence, we do not focus on metrics such as efficiency and scalability discussed in [5], [8]. The complexity of patterns discovered in [5], [8] is also less as they have a limit on maximum nodes.

Although graph pattern mining is the primary focus of this paper, relevance testing of discovered patterns is also an essential aspect of this study. To test relevance, we focus on the structural information of patterns and the weights involved. To apply this, we permute random edges to create random graphs. To compare and find out the similarity between discovered patterns and randomly created graphs, we propose using Locality Sensitive Hashing (LSH). LSH is a well-proven method for retrieving similar items for a query item. LSH has been previously proven effective in different domains having similarity search problems in high dimensional data sets. This advantage of reduced time and accuracy helps in the case of this study as well. Weighted vectorization of random graphs can result in high-dimensional data sets. In [21] LSH has been tested for retrieving similar images in an extensive database. Authors Ramiro Camino et al. of [21] conduct a similarity search for images in a distributed way. In [14] The authors

Ting Liu et al. have compared different neighbourhood approximation methods for graphs. Methods like LSH, Quantization, and Tree search were compared [14]. From the results, the authors claim LSH is superior compared to the other two methods. We selected LSH to compute the similarity between a pattern and random patterns quickly and accurately based on this comparison.

Sub-graph similarity and node similarity is a well-explored topic in the field of graph pattern mining. In [18] authors Tariq et al. have conducted a study that calculates node similarities for weighted graph networks. For calculating similarity, the method proposed in [18] converts graphs into a weighted adjacency matrix. Following a similar concept in this study, we convert discovered patterns into weighted vectors before performing relevance tests. Such a weighted vector structure allows comparing flow observed in the pattern and the structure of the pattern.

3 Data

In this chapter, we elaborate on the data of Bitcoin and Ethereum. Time frame, source of data, and weight thresholds are discussed in this chapter. As previously described, the transaction data of both cryptocurrencies is divided based on a time window of seven days. This chapter also mentions graph network statistics for each window of seven days and the entire dataset of both cryptocurrencies. As the Blockchain has some unique transaction aspects, we also present challenges and assumptions related to data.

3.1 Bitcoin

We used Bitcoin Blockchain data from January 2016 – February 2018. The selection of this time frame was made based on the price fluctuation for Bitcoin. The time frame from January 2016 – February 2018 provides us with the consolidating price of 2016, the exponential rise of 2017, and the downfall of early 2018. Different market sentiments affect the Blockchain transaction metrics such as number of users, number of transactions, average amount as shown in Figures 25, 26, 27. Blockchain metrics also show correlation with number of patterns, as shown in Figure 27.
The Bitcoin Blockchain is open-source data and can be downloaded through *Bitcoin core*. However, *Bitcoin core* downloads data from the first block of Bitcoin, and it takes a large amount of time and memory to complete the download. Hence, in this study, we downloaded data from *MIT data library*. Data from *MIT data library* is split into four parts transaction input, transaction output, transaction history, and block history. Transaction input describes transaction ID, senders in a transaction, and the amount sent by each one. Transaction output describes transaction ID, receivers, and the amount received. Transaction history mentions timestamps and block numbers for each of the transaction IDs. Block history has all rest of the information, such as block number, its position in Blockchain, and other details. We preprocessed these four files and combined information in them to generate graphs. Based on timestamps in the transaction history, all four files are split at a time interval of 7 days. After splitting the files into small chunks of data, each of them was searched for patterns. We also downloaded Bitcoin price data to set the varying edge weight thresholds. Based on the price of Bitcoin for a week, we calculated the edge weight threshold for each split that helps to monitor all transactions above 3000 dollars.

Figure 3 shows the distribution of the number of transactions in Bitcoin blocks. Each split of the Bitcoin data contains about 1100 blocks. Table 1 below shows graph network statistics for each split of the data as well as the entire dataset. The number of transactions in a Bitcoin block is highly co-related with price (shown in Figure 25). Hence, data described for a split in Table 1 is shown using a range of values.

Name	**Number of nodes**	**Number of Edges**	**Average flow per edge**
Bitcoin	237 Million	196 Million	4.46 Bitcoin
Bitcoin data for each split	1 Million to 3.3 Million	650k to 1.5 Million	2.23 Bitcoin to 4.46 Bitcoin

Table 1: Statistics mentioned for Bitcoin data are for the entire Bitcoin dataset, and statistics mentioned for each split are for each division in the dataset.

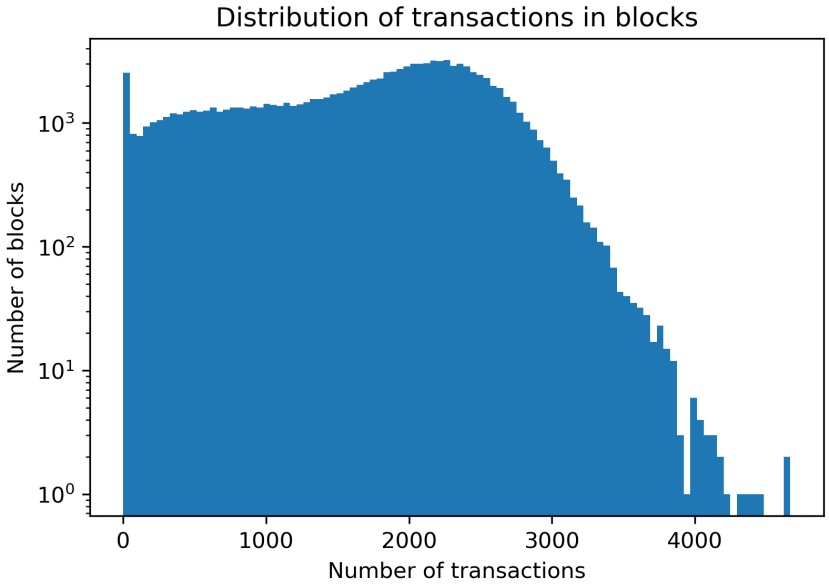


Figure 3: Bitcoin: Figure shows distribution of transactions for Bitcoin. Average transactions per block are approximately 1700. Only limited number of blocks show transactions more than 3000 in one single block. However, large number of blocks have less than 5 transactions.

3.2 Ethereum

Ethereum Blockchain data from January 2016 – February 2017 was used in this study. Similar to Bitcoin, the entire Ethereum Blockchain download takes approximately 4 TB of space and a large amount of time. However, API's of some websites allow downloading small portions of Ethereum transaction data. In this study, data was collected from *mainnet infura* via API. The use of API has the advantage of having data preprocessed to some extent. However, the disadvantage of having a limited time frame. Data exported from API source contains a lot of information for each transaction in Blockchain, out of which block ID, transaction ID, sender, receiver, amount, and timestamp is the relevant information for construction of graph network.

Wallet addresses, i.e., senders and receivers, are mapped to unique integers to simplify the process of pattern mining. Similar to Bitcoin data, Ethereum data is split at a time interval of 7 days and using Ethereum price data, an edge weight threshold is assigned to each week. Figure 4 shows distribution of transactions for Ethereum blocks. On average, a block contains 150 transactions. Each split of Ethereum data contains about 25000 blocks. Table 2 Shows data statistics for the entire Ethereum dataset as well as each split. Ethereum is under development compared to Bitcoin, and its Blockchain is expanding based on projects under Ethereum. The growth of Blockchain influences the transactions, and hence statistics shown for the split of Ethereum data have a wider range than Bitcoin data split.

Name	**Number of nodes**	**Number of Edges**	**Average flow per edge**
Ethereum	17.8 Million	143.6 Million	500 Ethereum to 1000 Ethereum
Approximate Ethereum data for each split	250 k to 2 Million	3 Million to 7.1 Million	500 Ethereum to 1000 Ethereum

Table 2: Statistics for entire Ethereum dataset and each of its split.

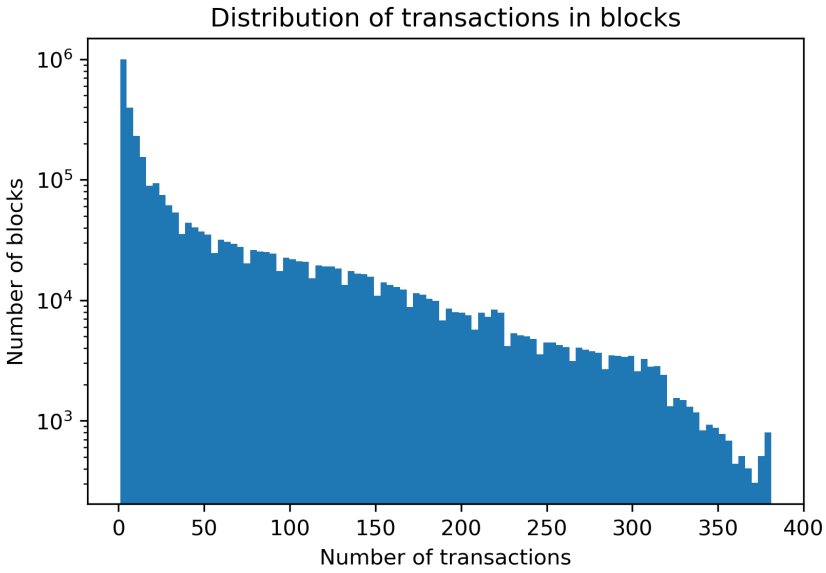


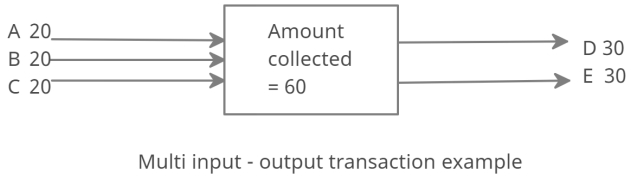
Figure 4: Ethereum: Figure shows distribution of transactions for Ethereum. Average number of transactions in an Ethereum block is approximately 150. Similar to Bitcoin distribution in Figure 3 many blocks have less than 5 transactions.

3.3 Challenges and assumptions

The graph structure in Figure 1 shows that each edge representing a single transaction in a directed weighted graph has one sender, one receiver, one amount, and one timestamp. However, Bitcoin transactions have one unique possibility of multi-input and multi-output functions. Multi-input and multi-output features create the possibility of one single transaction having more than one input and more than one output. Having the ability to send Bitcoins to multiple people in one transaction serves as an advantage in saving time and memory in a

block. However, this also reduces the transparency of the amounts involved. As the amount is collected from n-input addresses and then distributed to n-output addresses, there is no clarity about the individual amount send and received. Mixer services use this ability of multi-input and multi-output transactions of Bitcoin Blockchain to merge transactions and create confusion regarding the amount [9]. This unclarity about amounts involved in transactions creates a challenge of handling such transactions and pre-processing them in one input and output format.

Assumption 1: This study creates pairs of multi-input multi-output transactions based on the number of inputs and outputs. The total weight observed is divided equally between all the pairs created, assuming that they contribute equally. In every block, there are typically less than 5% transactions having a multi-input, multi-output structure. In this study, we pre-process them to create equally weighted single input-output type of transactions.
Figure 5 shows an example of Multi-input and multi-output transactions and the unclarity about amounts associated with it. Senders A, B, C send amount 20 each in a single transaction. A total amount of 60 is first collected and then distributed to 2 users, D and E. As shown in Figure 5, we create pairs of senders and receivers. Hence, if multi-input multi-output m senders and n receivers, it creates mXn additional transactions.

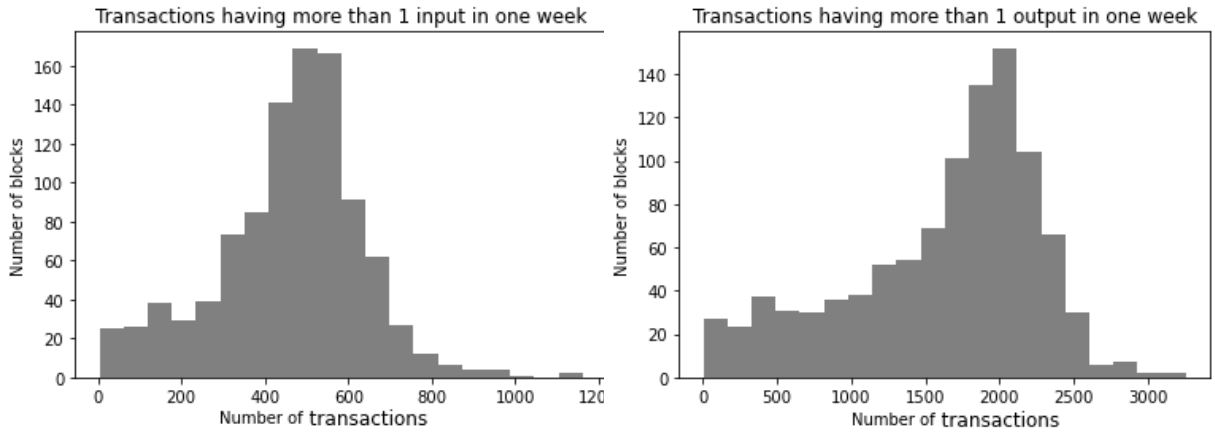


Source	Target	Flow
A	D	10
A	E	10
B	D	10
B	E	10
C	D	10
C	E	10

Figure 5: Example of a multi-input and multi-output transaction. A, B, C are the senders contributing amount of 20 each which is distributed as 30 for each receiver, i.e., D, E. Hence this transaction is simplified into 6 one-to-one transactions. But it assumes that all inputs contributed equally towards all receivers.

Due to multi-input, multi-output transactions, another challenge introduced is with coin treasury transactions. The total number of Bitcoins ever to be issued is 21 million, stored in the coin treasury of the Bitcoin Blockchain. Bitcoin and Ethereum miners get the reward for maintaining the Blockchain [22]. The transaction which pays this kind of reward is known as a coin-base or coin treasury transaction. Such transaction is not part of the input chunk of the data. Such coin-base transactions create missing entries when input and output chunks of the Blockchain data are compared. Coin-base transactions further make it challenging to merge the input and outputs of the transaction as an adjacency list.

*Assumption 2:*Coin treasury transactions are not considered in this study with the assumption that they are not suspicious. Coin treasury transactions are initiated through an algorithm of cryptocurrencies. Their involvement in any suspicious activity is highly unlikely. Hence, we removed such transactions in this study.



(a) (b)

Figure 6: Figure (a) shows the distribution of the transactions having more than 1 input in weekly data. And Figure (b) shows the distribution of more than 1 outputs. These two figures show that transactions with more than 1 input and output cannot be omitted from the data set.

4 Problem Statement

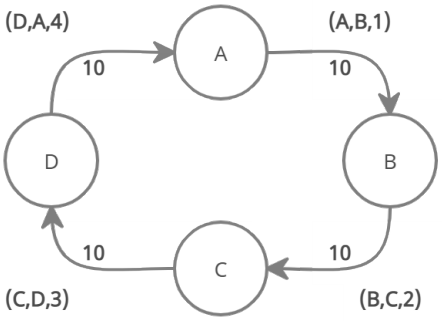
In this chapter, we define the problem of finding patterns in a formal way. Notations shown in Table 3 are used while defining this problem. Notations used in Table 3 are similar to [8]. While defining problems, we use terms such as source node, support node and layers previously defined in Chapter 1.

We denote a graph for each split in the data using notation $G(V, E, T, F)$. Each split of the data is always within duration constraint δ. In $G(V, E, T, F)$ V is the set of nodes, E is the set of edges, T is the set of timestamps and F is the set of weights or flow. In such a graph two nodes $u, v \in V$ can be connected by edge e and $e \in E$. Edges between u, v, are denoted by using $e(u, v)$. $f(e)$ represents one or more weights observed on edge e. Using $f_t(e)$, we denote flow observed on an edge e at timestamp t. v_{in} represents incoming edges of a node v and v_{out} represents outgoing edges of a node v. $f(v_{in})$ and $f(v_{out})$ represent total incoming flow and outgoing flow for a node v, respectively. With $f_t(v_{in})$ and $f_t(v_{out})$ we denote incoming and outgoing flow for a node v at a timestamp t. N is the number of unique nodes and M is the number of unique edges in a graph. $\sigma(a, b, c)$ calculates standard deviation between data points a, b, c. The degree of a node is the count of the number of nodes connected to it. For a node, in-degree measures the number of nodes connected by incoming edges. Out-degree measures nodes connected by outgoing edges. For each such split, the aim is to find circular patterns $G_c(V_c, E_c, T_c, F_c)$ or diamond patterns $G_d(V_d, E_d, T_d, F_d)$. When notations are applied to layers, they act for all the elements in that layer. Notations $f(layer2_{in})$, $f(layer2_{out})$ show combined inflow and outflow of all nodes in layer 2. Both circular and diamond patterns always have a source node, a support node and at lest two layer 2 nodes.

Notation	Definition
$G(V, E, T, F)$	Directed weighted input graph
$G_d(V_d, E_d, T_d, F_d)$	Diamond pattern
$G_c(V_c, E_c, T_d, F_d)$	Circular pattern
δ	Duration constraint for pattern
$e(u, v)$	Set of edges from u to v
$f(e)$	Flow on edge e
$f_t(e)$	Flow on edge e at timestamp t
v_{in}	Incoming edges of node v
v_{out}	Incoming edges of node v
$t(e)$	Timestamp of edge e
$f(v_{in})$	Incoming flow of node v
$f(v_{out})$	Outgoing flow of node v
N	Number of unique nodes
M	Number of unique edges
$\sigma(a, b, c)$	Standard deviation of (a, b)

Table 3: Notations used in problem defining and their definitions

4.1 Problem statement: circular patterns



Source	Target	Amount	Timestamp
A	B	10	1
B	C	10	2
C	D	10	3
D	A	10	4

Figure 7: Example of a circular pattern along with the table explaining the edges of the pattern.

- **Definition:** A circular graph pattern $G_c(V_c, E_c, T_c, F_c)$ is a sub-graph of $G(V, E, T, F)$, where N $\geq$ 4, M $\geq$ 4. For the source node and support node in V_c, in-degree $\geq$ 1 and out-degree $\geq$ 1. For each of layer 2 nodes in V_c in-degree = 1 and out-degree = 1.
  In a circular pattern, E(source node, support node) cannot exist but E(source node, layer 2 nodes) and E(layer 2 nodes, support node) always exists.
- **Problem statement:** Given a graph $G(V, E, T, F)$, find $G_c(V_c, E_c, T_d, F_d)$ where each of the following condition is true.
- *Condition 1*

$$\sigma(f(layer1_{in}), f(layer1_{out}), f(layer2_{in}), f(layer2_{out}), f(layer3_{in}), f(layer3_{out})) < \phi \quad (1)$$

Where

$$\sigma = \sqrt{\frac{\sum_i^N (f_i - \mu)^2}{n}} \tag{2}$$

and ϕ is the deviation threshold. For Equation 2, n is the number of data points used for standard deviation calculation, μ is the mean for the data points and f_i represents a flow.
Condition 1 in Equation 1 ensures the amount transferred between different nodes of a circular pattern is similar and under deviation threshold ϕ.

- *Condition 2*

$$t(layer1_{out}) < t(layer1_{in}) \tag{3}$$

Condition 2 in Equation 3 ensures the source node first sends the amount before receiving it back. Timestamp conditions are implemented only on source node, as there can be multiple transactions between source node and layer 2 nodes.

We do not set any upper limit on N or M for circular patterns. Only flow and structural conditions are defined to find circular patterns in the dataset. This helps to explore complex pattern as shown in Figure 28.

4.2 Example: circular pattern

The structure of circular pattern and conditions can be better understood with the circular pattern $G_c(V_c, E_c, T_c, F_c)$ shown in Figure 7. For the circular pattern shown in Figure 7 graph network components nodes, edges, timestamps, and flow are defined as

$$V_c = \{A, B, C, D\}, \tag{4}$$

Equation 4 shows the nodes involved in a circular pattern

$$E_c = \{(A, B, 1), (B, C, 2), (C, D, 3), (D, A, 4)\} \tag{5}$$

Equation 5 illustrates edges that define the relationship between nodes.

$$T_c = \{1, 2, 3, 4\} \tag{6}$$

$$F_c = \{10, 10, 10, 10\} \tag{7}$$

Equation 6 shows the timestamps and Equation 7 shows the weights on the edges. referencing to Figure 7, Conditions in Equations 1, 3 can be explained as follows.
Condition 1

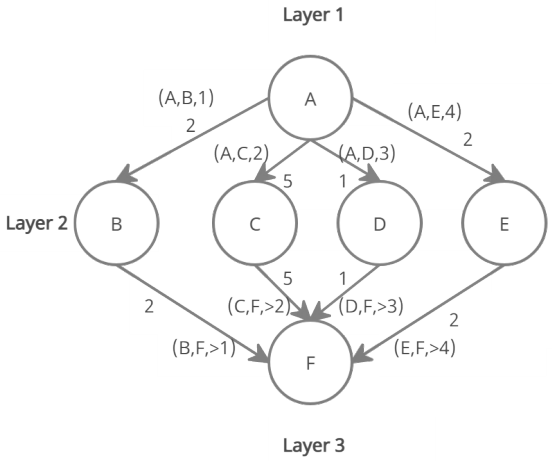
$$\sigma(f(A_{out}), f(B_{out}), f(C_{out}), f(D_{out}), f(A_{in})) < \phi. \tag{8}$$

Equation 8 implements $f(A_{out}) \approx f(B_{out}) \approx f(C_{out}) \approx f(D_{out}) \approx f(A_{in})$. Equation 8 ensures flow observed for each node remains similar.
Condition 2

$$t(A_{out}) < t(A_{in}). \tag{9}$$

Equation 9 Ensures source node A, first sends the amount before receiving any of it back.

4.3 Problem statement: diamond patterns



Source	Target	Amount	Timestamp
A	B	2	1
A	C	5	2
A	D	1	3
A	E	2	4
B	F	2	3
C	F	5	5
D	F	1	4
E	F	1	6

Figure 8: Example of a diamond patterns along with the table explaining the edges of the pattern.

- **Definition:** A diamond graph $G_d(V_d, E_d, T_d, F_d)$ is a sub-graph of $G(V, E, T, F)$, where N $\geq$ 4 and M $\geq$ 4. For the source node in V_d, in-degree = 0 and out-degree $\geq$ 2. For the support node in V_d in-degree $\geq$ 2 and out-degree = 0. For each of layer 2 nodes in V_d in-degree = 1 and out-degree = 1.
  In a diamond pattern, E(source node, support node) cannot exist but E(source node, layer 2 nodes) and E(layer 2 nodes, support node) always exists.

- **Problem statement:** Given a graph $G(V, E, T, F)$, find $G_d(V_d, E_d, T_d, F_d)$ where each of the following condition is true.

- *Condition 1*
  $$\sigma(f(layer1_{out}), f(layer2_{out}), f(layer3_{in})) < \phi \tag{10}$$
  Where
  $$\sigma = \sqrt{\frac{\sum_i^N (f_i - \mu)^2}{n}} \tag{11}$$
  and ϕ is the deviation threshold. For Equation 11, n is the number of data points used for standard deviation calculation, μ is the mean for the data points and f_i represents a flow.
  Condition 1 in Equation 10 ensures the amount sent out by source node is the same amount received by sink node. If there is any change in the amount, then it should be under the deviation threshold ϕ.

- *Condition 2*
  $$t(Layer1_{out}) < t(Layer3_{in}) \tag{12}$$
  $$t(Layer2_{in}) < t(Layer2_{out}) \tag{13}$$

The condition in Equation 12 ensures that sink node does not receive any amount before being sent by source node. With Equation 13, a condition is set for layer 2 nodes that ensures layer 2 nodes receive an amount before sending it to the sink node.

Similar to circular patterns, we do not put any structural limit on diamond patterns. Upper bound N and M is not defined which helps in discovering complex patterns.
Graph components of a diamond pattern $G_d(V_d, E_d, T_d, F_d)$ shown in Figure 8 are defined as follows

$$V_d = \{A, B, C, D, E, F\}, \tag{14}$$

$$E_d = \{(A, B, 1), (A, C, 2), (A, D, 3), (A, E, 4), (B, F, 3), (C, F, 5), (D, F, 4), (E, F, 6)\} \tag{15}$$

$$T_d = \{1, 2, 3, 4, 3, 5, 4, 6\} \tag{16}$$

$$F_d = \{2, 5, 1, 2, 2, 5, 1, 1\} \tag{17}$$

4.4 Example: diamond patterns

Referring to Figure 8 the conditions for diamond pattern can be defined as follows

Condition 1

$$\sigma(f(A_{out}), f(B_{in}), f(C_{in}), f(D_{in}), f(E_{in})f(B_{out}), f(C_{out}), f(D_{out}), f(E_{out}), f(F_{in})) < \phi \tag{18}$$

Equation 18 ensures the flow observed in all three layers is similar as shown below.

$$f(A_{out}) \approx \sum \{f(B_{in}), f(C_{in}), f(D_{in}), f(E_{in})\} \tag{19}$$

$$f(F_{in}) \approx \sum \{f(B_{out}), f(C_{out}), f(D_{out}), f(E_{out})\} \tag{20}$$

Condition 2

$$t(A_{out}) < t(F_{in}) \tag{21}$$

$$t(B_{in}) < t(B_{out}), t(C_{in}) < t(C_{out}), t(D_{in}) < t(D_{out}), t(E_{in}) < t(E_{out}) \tag{22}$$

Equation 21 ensures sink node F does not receive any amount before being sent by source node A. Using Equation 22 a condition is set for layer 2 nodes that ensures each node in layer 2 first receives an amount from source node before sending it to sink node.

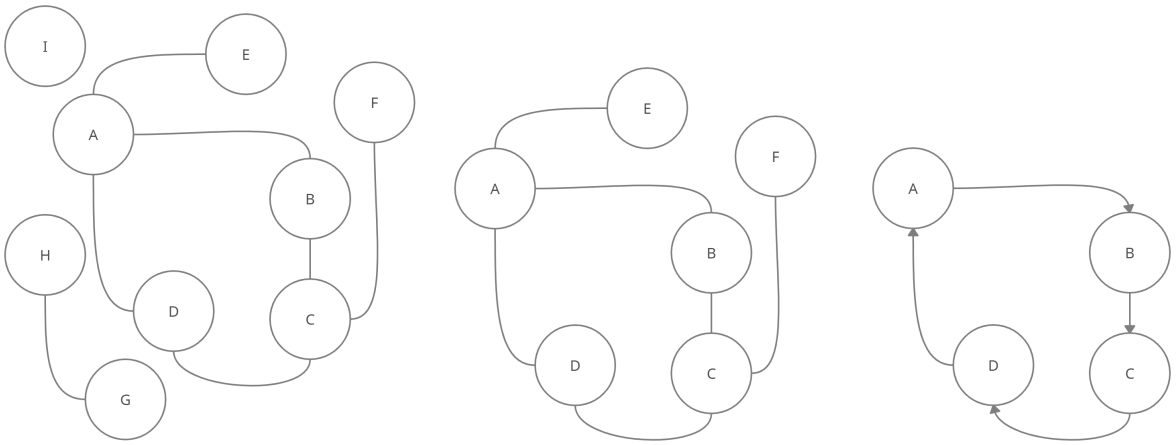
5 Methods

In this study, we designed two algorithms to find circular and diamond patterns. Given the input of a data split $G(V, E, T, F)$, algorithm diamond track finds $G_d(V_d, E_d, T_d, F_d)$, and circle track finds $G_c(V_c, E_c, T_c, F_c)$ that fulfil the conditions in Chapter 4. For the search of the patterns, algorithms take a hybrid approach of combining BFS and DFS. The hybrid approach is executed by searching for all patterns for a source node, but parallelly finding all patterns for a combination of a source node and a support node. Figure 9 (a) shows an example input received by the algorithm. For the given input data, algorithms first find a subset of data containing a pattern as shown in Figure 9 (b). An unclean pattern shown in Figure 9 (b) is a sub-graph that holds structural information about all the node interactions of interesting nodes, but can also have some non-interesting edges and nodes. Secondly, algorithms get rid of unnecessary part of the subset, resulting in the pattern shown in Figure 9 (c). A clean pattern graph in Figure 9 (c) is a sub-graph of only interesting nodes and does not have the noise of other non-interesting nodes and edges. All the edges that are part of a clean pattern satisfy the time and flow threshold conditions. In the final step, the algorithm checks the patterns for conditions in Chapter 4 and saves or discards them. Although Figure 9 (c) shows a circular pattern as a product of clean stage. Steps involved in finding diamond patterns are identical.

Pseudo-code of the circle track in Algorithms 1, 2, 3 and diamond track in Algorithms 4, 5 explains all the steps, using an iterative loop, as the process is the same for the multiple elements of the data set. However, the implementation of the algorithm, should use least possible iterative loops until a short subset of the data is taken. Furthermore, not having a dependency on multiple iterative loops ensures the algorithm's speed remains competitive and enables fast discovery of patterns.

In terms of time, our algorithms do not scale very well. When compared to frequent mining algorithm in [8] our algorithms take more time to find patterns. Inefficiency of algorithms is due to not having a structural limit for both circular and diamond patterns and low edge weight threshold. Primary purpose of creating these algorithms is to find patterns previously explored and also to explore some more similar but complex patterns. Unlike [8], we do not aim to create scalable and efficient algorithms. Empirical run time is illustrated in Chapter 6 Table 11.

In this chapter, we present detailed working of circle track algorithm in chapter 5.1 and diamond track algorithm in Chapter 5.2. We also present our novel evaluation method, *relevance evaluation,* in Chapter 5.3



(a) Overall (b) Unclean (c) Clean

Figure 9: In this image summarized idea of algorithms is seen. The workflow of the algorithm goes from left to right. Figure (a) example of the overall graph of weeks transactions is seen. In Figure (b) An unclean pattern is shown, and in Figure (c) a clean pattern having only interesting nodes and edges is shown.

5.1 Algorithm Circle track

In this algorithm, we search for the circular patterns $G_c(V_c, E_c, T_c, F_c)$ for a source node. Source nodes are selected from the candidate list. A candidate list is created based on the interaction of a sender node with receiver nodes in $G(V, E, T, F)$. For every source node, a support node is selected in an iterative way. For a pair of source nodes and support nodes, all circular patterns are searched in a single scan. Algorithms workflow is broken down in three steps. In the first step, algorithms generates a candidate list and a support list. In the second step, the algorithm takes a subset of data with probable circular pattern in it. In the last step, the subset is cleaned to find the circular pattern.

5.1.1 Node selection

The first step of the algorithm aims to find candidate nodes that possibly create circular patterns. In this first step, we also generate a list of support nodes. Each node from candidate nodes is a potential node for Address 1, and each node from support nodes is a potential node for Address 3 shown in Figure 1 (a).
To find candidate nodes that may create circular patterns, we use parts transaction input and transaction output of the data. As displayed in Algorithm 1 line 4 using transaction input and transaction output, an intersection is taken for senders and receivers of transactions. The result of this intersection is further filtered by setting a condition that a node should be first a sender and then a receiver later in time. This condition fulfils the first goal of finding candidate nodes. In the next step shown on line 5 a node $Address1$ is selected sequentially. The selected node acts as a source node and sink node for the circular pattern. The next step is to find incoming neighbours $(degree\_1\_in)$ and outgoing neighbours $(degree\_1\_out)$ for the selected node. In Algorithm 1 these two neighbour finding steps are executed in line 7 and line 8. Further, for each node in $(degree\_1\_in)$ incoming neighbours are calculated marked as $(degree\_2\_in)$ and for $(degree\_1\_out)$ outgoing neighbours are calculated marked as $(degree\_2\_out)$. Taking the intersection of $(degree\_2\_in)$ and $(degree\_2\_out)$ generates the list of support nodes. The

generation of support nodes is a crucial step as it helps to establish a circular structure of a pattern. Steps for finding $(degree\_2\_in)$, $(degree\_2\_out)$ and support nodes are executed from line 9 to line 16. List of support nodes is marked as $(degree\_2\_commons)$ in Algorithm 1. Another important step is to save nodes in the order of their discovery. Saving nodes in order of their discovery helps later in the cleaning step of the algorithm. In Algorithm 1 line 6 initiates a list that helps to save nodes.

Algorithm 1 Circle_track (Node selection)

```
1: Input — Block-chain data consisting of Sender, receiver, weight, timestamp, i.e., G(V, E, T, F)
2: Output — List of support nodes
3: while nodes in-degree_2_commons < 2 do
4:     Potential_candidates = Source ∩ Target
5:     Address 1 = Potential_candidate[i]
6:     Node_track += Address 1
7:     degree_1_in = in_neighbours(Address 1)
8:     degree_1_out = out_neighbours(Address 1)
9:     for node in degree_1_in do
10:        degree_2_in = in_neighbors(degree_1_in[node])
11:    end for
12:    for node in degree_1_out do
13:        degree_2_out = out_neighbors(degree_1_out[node])
14:    end for
15:
16:    degree_2_commons = degree_2_in ∩ degree_2_out
17: end while
18:
```

5.1.2 Data subset selection

The second step aims to create a subset of the data that has a circular pattern. To create this subset, the algorithm takes the input of support nodes generated in the first step. From the input, a node is sequentially selected which is marked as $Address3$ in Algorithm 2 line 3. Similar to $Address1$ in the first step, we calculate incoming and outgoing neighbours for $Address3$. This step is illustrated in Algorithm 2 line 4 and 5 in which incoming neighbours are saved as in_circle and outgoing neighbours are saved as out_circle. Compared to Figure 1 (a), $Address3$ in line 3 is the same as Address 3. Calculation of in_circle and out_circle finds nodes that are potential candidates for Address 2 and Address 4 in Figure 1 (a).
Same as Algorithm 1 $Node\_track$ saves nodes and neighbours in order of in_circle, $Address3$ and then out_circle. The structure of Figure 1 (a) demonstrates that all the nodes that are part of the circular pattern are both senders and receivers. Hence, using nodes saved in $Node\_track$ and a condition that states nodes should be senders and receivers, a subset of data is taken. Based on the interaction of nodes, there is a chance of this subset having noisy nodes. A subset taken from $G(V, E, T, F)$ reduces pattern search time but requires the additional step of cleaning the subset.

Algorithm 2 Circle_track (Data subset selection)

```
1: Input Blockchain data, support nodes.
2: Output Subset of data having only nodes of interest.
3: Address 3 = degree_2_commons[i]
4: in_circle = in_neighbours(Address 3)
5: out_circle = out_neighbours(Address 3)
6: Node_track += in_circle
7: Node_track += Address 3
8: Node_track += out_circle
9: for Node in node track list do
10:     for Row in Blockchain data do
11:         if node in "source" or node in "Target" then
12:             save the Row in data subset
13:         end if
14:     end for
15: end for
16:
```

5.1.3 Data cleaning

The subset contains a circular pattern, but it also has noisy edges and nodes not contributing to a circular pattern. The primary purpose of this step is to get rid of noisy nodes and edges. The secondary purpose of this step is to apply conditions stated in Chapter 4.
Going back to the $Node\_track$ list, as we have saved the visited nodes in order, the algorithm creates pairs of nodes. Based on these pairs, the data set is filtered. This filtering yields us the circular pattern without any noisy edges. The step executed in Algorithm 3 from line 3 to line 7 removes transactions that are not part of pairs created. As a result of removing transactions, the primary idea of all the nodes in the circular pattern being senders and receivers gets compromised. Hence, steps conducted in algorithm 3 from line 12 to line 20 help further remove noisy nodes. The combination of these three cleaning steps yields a clean circular pattern which is further checked for conditions discussed in Chapter 4.
After filtering step, if a pattern does not have a circular shape or does not satisfy the conditions from Chapter 4, the algorithm goes back to Algorithm 2 line 3 and changes the selected node for $Address3$ from degree_2_commons, and the rest of the steps remain the same.

Algorithm 3 Circle_track(Data cleaning)

```
1: Input — Noisy Circle Dataframe
2: Output — G_c(V_c, E_c, T_c, F_c)
3: for node in data subset do
4:     Create permutation of node and next node.
5:     for Row in Data do
6:         if permutation not in Row  then
7:             Remove Row
8:         end if
9:     end for
10: end for
11:
12: for Row in subset do
13:     if node is in "target" and not in "Source" then
14:         Remove row
15:     end if
16: end for
17:
18: for Row in subset do
19:     if node is in "Source" and not in "target" then
20:         Remove row (Step 6)
21:     end if
22: end for
```

5.2 Algorithm Diamond track

Steps involved in diamond track are identical to circle track, but with few changes in the traversing direction for the graph $G(V, E, T, F)$. The aim of diamond track algorithm is to find $G_d(V_d, E_d, T_d, F_d)$ in $G(V, E, T, F)$. In the first step, a candidate list a created based on in-degree and out-degree of nodes in $G(V, E, T, F)$. To preserve the diamond structure, all the nodes above out degree of 2 are candidate nodes. Also in this first step, the algorithm creates a list of support nodes. In the second step, a subset of data is taken that contains a diamond pattern. Using a cleaning step, all unnecessary parts of a subset such as noisy nodes and edges are removed.

5.2.1 Get diamond location

The aim of the first step in the algorithm is to generate a list of candidate nodes that may create a diamond pattern. Also, this first step generates a list of support nodes.
For creating a list of candidate nodes for diamond shape, we drop all the nodes that have less than 2 outgoing neighbours. As Figure 1 (b) shows, to qualify as a diamond shape, a pattern should have two more or more splits. Dropping all the nodes with out-degree less than 2 creates the list of candidate nodes. In Algorithm 4 candidate set is generated on line 4. Each node from $Candidate\_list$ acts as a source node.
To create the list of support node, we first find outgoing neighbours of $Address1$. The list of outgoing neighbours is sorted and split into 2 parts, marked as $out\_neighbours\_pt\_1$ and $out\_neighbours\_pt\_2$. Sorting this list rearranges the outgoing neighbours and makes finding the diamond pattern easier in later steps. Steps of finding outgoing neighbours, sorting, and

splitting are executed on lines 7, 8 and 9 respectively. For each split created, we find outgoing neighbours for each element in the split. Finding neighbours for each element in the split generates two more lists marked as out_deg2_split1 and out_deg2_split2 in Algorithm 4. Steps of finding outgoing neighbours for each split are shown on lines 11 to 17. Taking intersections of out_deg2_split1 and out_deg2_split2 generates list of support nodes. Each node from the list of support nodes is a candidate node for sink node, i.e., Address 5 shown in Figure 1 (b). Having a non-empty list of support nodes shows that there is a chance that a diamond pattern might occur for selected $Address1$. Each node from $out\_neighbours$ acts as a probable layer 2 node.

Algorithm 4 Diamond_track(Get diamond location)

```
1: Input Blockchain data consisting of sender, receiver, weight, and timestamp, i.e. G(V, E, T, F)
2: Output List of support nodes.
3: while nodes in-degree_2_commons < 2 do
4:     Candidate_list = Nodes[out_degree(Nodes) > 2]
5:     Address 1 = Select(candidate_list).
6:     node scanned += Address 1.
7:     out_neighbours = out_neighbours(Address 1)
8:     out_neighbours = sort(out_neighbours).
9:     out_neighbours_pt_1, out_neighbours_pt_2 = Split(out_neighbours)
10:    parts = list(out_neighbours_pt_1, out_neighbours_pt_2)
11:    for part in parts do
12:        for node in part do
13:            out_deg2_split = out_neighbours(part).
14:        end for
15:        Save neighbours
16:    end for
17:    (Returns 2 lists out_deg2_split1, out_deg2_split2)
18:    degree_2_commons = out_deg2_split1 ∩ out_deg2_split2
19: end while
20:
```

5.2.2 Mine diamond

From the first step executed by Algorithm 4, we have a source node $Address1$, probable layer two nodes in two lists out_deg2_split1, out_deg2_split2 and a possible sink node list $degree\_2\_commons$. Given the input of these three, the second step executes similar steps as Algorithm 4. The output of this step is a subset of $G(V, E, T, F)$ that may contain a noisy diamond pattern.

To generate this noisy diamond, the main task of this step is to filter out nodes from out_deg2_split1 and out_deg2_split2 that contribute to diamond formation. To execute this filtering, Algorithm 5 selects a node from $degree\_2\_commons$. For this selected node incoming neighbours $in\_diamond$ are calculated as shown in Algorithm 5 line 5. Following the same steps as Algorithm 4, the list $in\_diamond$ is sorted and split in two parts $in\_diamond\_pt1$ and $in\_diamond\_pt2$. As we have previously sorted and split out_neighbours in Algorithm 5 in this step we can take intersection of $in\_diamond\_pt1$ with $out\_neighbours\_pt\_1$ and

$in\_diamond\_pt2$ with $out\_neighbours\_pt\_2$. These two intersections create lists of layer 2 nodes for the diamond, as shown in lines 7 and 8. As long as one of $side\_1$ or $side\_2$ are not empty, we can continue further to save nodes and fetch a subset of data from $G(V, E, T, F)$. As observed in Figure 1 in a diamond pattern, all the nodes are senders in at least one transaction except for the sink node. Moreover, except for the source node, all the other nodes are receivers in at least one transaction. Based on this information, a transaction subset is selected from the primary dataset $G(V, E, T, F)$.

Algorithm 5 Diamond_track(mine diamond)

```
1: Input Block chain Data, source node, sink node, degree_2_commons
2: Output Noisy subset of Data containing Diamond
3: node_track = Save(source node)
4: Address 5 = degree_2_commons[i]
5: in_diamond = in_neighbours(Address 5)
6: in_diamond_pt_1, in_diamond_pt_2 = Sort & split (in_diamond)
7: side_1 = in_diamond_pt_1 ∩ out_neighbours_pt_1
8: side_2 = in_diamond_pt_2 ∩ out_neighbours_pt_2
9: node_track = Save side_1 and side_2
10: node_track = Save Address 5
11: for node in node_track do
12:     get and save transactions of node
13: end for(Returns data noisy data subset having only nodes contributing to diamond)
```

5.2.3 Clean diamond

In the cleaning step, the primary goal is to remove noisy edges and nodes and generate a clean pattern using the subset created in Algorithm 5. The cleaning process is simplified by the nodes saved in order of discovery. As the nodes are saved in sequential order of their discovery, it is possible to create permutations of two nodes. The subset can now be filtered, having only specific permutations created. This filtering creates an exact diamond pattern $G_d(V_d, E_d, T_d, F_d)$ as required. The cleaning algorithm for diamond pattern is similar to the Algorithm 3. Following the same steps as Algorithm 3 diamond patterns are tested in this step for the conditions defined in Chapter 4. Failing to find a diamond pattern at this stage, the algorithm goes back to Algorithm 5 and selects a new node for $Address5$.

5.3 Relevance evaluation

In this chapter, we explain our evaluation method, *relevance evaluation*. We first explain the general idea, the problem with existing methods, approach and important part of LSH. Later in Chapter 5.3.1, we elaborate on steps involved in relevance evaluation. Using two examples in Chapter 5.3.2, we explain the effectiveness of this method.

- *General idea:* Once patterns are discovered, there is a challenge of classifying them as usual or unusual observations. As stated earlier, usual patterns might be the result of common transactions, and unusual patterns result from malicious activity. Even if we assume all patterns are unusual observations, there is still a need to evaluate the risk associated with the discovered patterns. Given that numerous graph patterns are found

in a data split, we want to evaluate their risk based on their structure and weights. Relevance evaluation is an evaluation method for the discovered patterns. The primary idea behind this implementation is that if nodes of a pattern have transactions with a limited or low number of nodes, then the circular or diamond pattern created by them is more likely to happen. However, if nodes of a pattern are involved in multiple transactions or have transacted with many nodes in the overall graph, then the chances of creating an exact circular or diamond patterns are low.

- *Existing methods and problems:* The existing randomization tests evaluate the chance of observing a pattern in a randomized dataset. Moreover, some randomization tests only permute weights in a pattern and do not consider the structure of the pattern while calculating significance [8]. The approach in the existing randomization test is well suited for patterns with small sizes. However, in this study, we have not limited the maximum number of nodes in patterns. Hence, some patterns can be large and have numerous unique nodes and edges. The chance of observing such large patterns in the existing randomized test is very low. With relevance evaluation, we created a method that evaluates the chance of observing $similar$ structure like the discovered pattern and observing weights in the order of the same as the discovered pattern. Relevance evaluation implements a weighted randomization test that combines the approach of randomization test in [8] and weighted adjacency matrix in [18].

- *Implementation approach:* For the randomization, we use nodes from the unclean pattern saved by algorithms. Non-interesting nodes in the unclean pattern are the nodes that are not part of a clean pattern. For example, nodes E and F are non-interesting nodes in an unclean pattern, as shown in Figure 9 (b). Non-interesting nodes are the real wallet addresses from the data connected with nodes in a clean pattern. Nodes in the clean pattern have transacted with non-interesting nodes; however, those transactions are not part of the clean pattern. Random graphs patterns created with non-interesting nodes give an idea about all possible transaction patterns that could have happened for nodes in a clean pattern. If a pattern does not differ between the unclean and clean stages, relevance evaluation still has some effectiveness due to randomization of edges and weights. We keep the number of transactions, i.e., number of edges, in random graphs the same as the clean pattern. Moreover, each edge in a random graph is assigned with a weight observed in the clean pattern. Weights are selected randomly for every edge in a non-repetitive way. With this approach of implementation, relevance evaluation answers the following question
  Given the nodes in the unclean pattern create random graphs and random graph nodes are connected with the same number of weighted edges as the clean pattern, how similar are the random graphs compared to the clean pattern
  To answer this, the relevance evaluation conducts three tests; similarity test, skew test, and kurtosis test. Each of these tests assigns an evaluation score to a pattern. Results of the relevance evaluation are based on similarity comparison of a pattern with 1000 random patterns.

- *LSH aspect:* LSH computes similarity by calculating the Euclidean distance between a discovered pattern and each random pattern. We selected LSH due to its fast computation speed and accuracy. A higher number of nodes in an unclean pattern can create many candidate pairs for the random graph generation. When represented as weighted

vectors, patterns can result in high dimensional vectors. The total number of possible pairs is given by the formula, n^r, where n is the number of nodes in an unclean pattern and r is always 2. For example, if the unclean stage of a pattern has a noise of 100 nodes, then there will be 10k possible pairs. As the relevance evaluation uses 1000 random graphs, the matrix representing random graphs as weighted vectors would be of size $1k\ X\ 10k$. Hence, we use LSH for faster and accurate calculation of similarity. The output of LSH similarity calculation is a list of Euclidean distances between discovered patterns and random patterns. The relevance evaluation uses the list of Euclidean distances to conduct various statistical tests.

5.3.1 Workflow

Relevance evaluation can be divided into 3 steps, creating random graphs, calculating similarity scores and conducting statistical tests.

1. **Generation of random graphs:** For creating random graphs, we use unclean patterns. Based on the number of unique nodes in an unclean pattern, we create all possible pairs of two nodes. While creating pairs, we do allow repetition of the node in the same pair. This repetition allows an address to be a sender and a receiver in the same transaction. As we have observed in Bitcoin data, some transactions have the same address as sender and receiver; hence, we consider the same while creating random pairs. However, we do not allow such repetition in case of Ethereum data as no transactions with same sender and receiver are observed in Ethereum network. The pairs created; represent all possible transactions that could have happened for all the nodes present in a clean graph. As patterns are directed graphs, pairs are also created in a directed way. For example: (b, a) is not the same as (a, b), and both should be in possible combinations. we create 1000 random graphs using the pairs generated. These random graphs have the same number of transactions as the clean pattern. The high number of random graphs prevent results from being a random observation.

2. **Similarity score calculation:** To calculate similarity scores, the first step is to convert clean pattern and randomly generated patterns into weighted vectors. Randomly generated Patterns are converted into weighted vectors using the weights observed in a clean pattern and a pair of nodes present in them. In the second step, from the observed weights, we assign a random weight to each edge in randomly generated patterns [18] [8]. Similarly, the clean pattern is also converted to a weighted vector. In the final step, each such vector is stored in the $nearpy$ engine. Using clean pattern as query, the vector Euclidean distance between each randomly generated vector is calculated. These Euclidean distances are the similarity scores.

3. **Statistical test:** Based on these similarity scores, similarity test, kurtosis test and skew test are conducted.

    - similarity test computes the mean of similarity scores. Observing a higher mean illustrates low structural similarity between the clean pattern and all random patterns. Low structural similarity shows that, based on the overall transactions of the nodes in the unclean pattern, the occurrence of the observed clean pattern is unlikely to be random. The similarity test computes the mean of n similarity scores observed for n randomly generated patterns.

- The skew test measures the skewness of the similarity scores. Results of skew test can be positive or negative. A negative skew value shows that similarity scores are skewed to the left, i.e. most of the similarity scores show low similarity. In contrast, a positive skew value shows most of the observations from similarity scores have higher similarity. Using skew test helps in quantifying whether components of similarity score show overall high similarity or overall low similarity.
- The kurtosis test is the tail test and measures if the data is heavy tailed or lightly tailed. Value observed for kurtosis test can be positive or negative, where positive value shows heavy tailed distribution and negative value shows lightly tailed distribution. If the observed distribution is light-tailed, it illustrates that the similarity scores are concentrated in a particular range.

Observing results of similarity test, kurtosis test and skew test patterns can be sorted from usual observations to unusual observations. Results of kurtosis test and skew test can be interpreted together to assess the patterns in a better way. The following Table 4 shows the possible combinations and their interpretations for kurtosis test and skew test.

Observation	Interpretation
Positive skew Negative kurtosis	Pattern is the most likely candidate for usual transfer as it has high similarity with most random patterns and as it is lightly tailed it does not show low similarity with many patterns.
Positive Skew Positive kurtosis	The pattern is a potential candidate for usual transfer as has high similarity with some patterns. As it is heavy tailed, it has some similarity with most patterns.
Negative skew Positive kurtosis	The pattern is the potential candidate for unusual transfer as it has the low similarity with many of the random graphs created but also has some similarity with few patterns.
Negative skew Negative kurtosis	The pattern is a most potential candidate for unusual as it has the least similarity with most random patterns.

Table 4: Observed combination of skewness test and kurtosis test and their interpretation

5.3.2 Proof of concept

Using the following examples, we demonstrate the working of relevance evaluation. In this example, we consider two cases, one with low noise and one with high noise. This example also shows the structure of the structure of clean, unclean, and random patterns in Table 5 and Table 7. Along with this example of weighted vectors and similarity score distribution is illustrated in Table 6, Table 8 and Figure 12 respectively.

1. Case 1 : — Unclean graph considered in this example has moderate noise, which means it has few uninteresting nodes. This example has three noisy transactions. In Table 5 the transactions marked in italics format, i.e., edge number 2,4,7 are the noisy edges. Removing these edges results in a clean pattern as shown in Table 5. A total of 1000 random graphs are created by using all the unique nodes from the unclean pattern. Based on the edges, all the random graphs are converted into weighted vectors as illustrated in Table 6. The number of edges in each random graph is equal to the number of edges

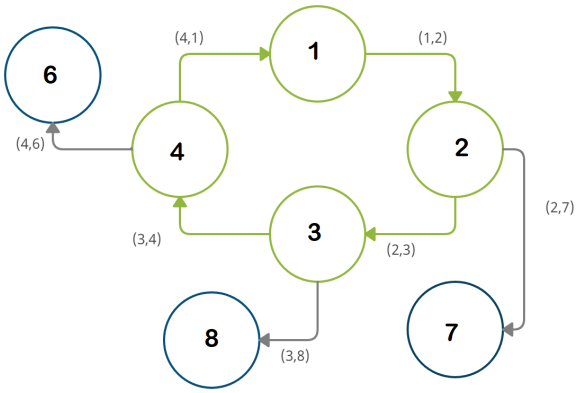
in the clean pattern. Figure 12 (a) shows the results of calculating vector similarity for this noisy pattern. A visual example of unclean pattern, clean pattern and randomly generated patterns for this case is demonstrated in Figure 10. In Figure 10, (a) nodes in blue are the non-interesting nodes and (c), (d), (e) are randomly generated graphs.

	Edge 1	**Edge 2**	**Edge 3**	**Edge 4**	**Edge 5**	**Edge 6**	**Edge 7**
Unclean pattern	(3,4)	*(3,8)*	(4,1)	*(4,6)*	(1,2)	(2,3)	*(2,7)*
Clean pattern	(3,4)	(4,1)	(1,2)	(2,3)	-	-	-
Random graph 1	(4,2)	(3,7)	(2,3)	(4,7)	-	-	-
Random graph 2	(1,3)	(2,7)	(4,7)	(3,4)	-	-	-
Random graph 1000	(7,2)	(4,7)	(1,7)	(2,1)	-	-	-

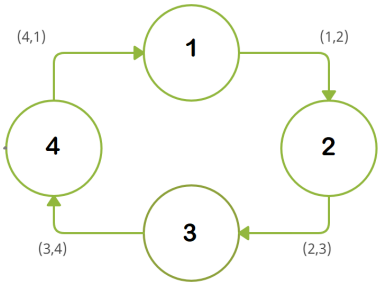
Table 5: Example pattern and random permutations for a pattern with high noise

	(1,2)	(1,3)	(1,4)	(1,7)	(2,1)	(2,3)	n such columns ->
Random graph 1	0	0	0	0	0	13	12,0,0,11,0,9
Random graph 2	0	13	0	0	0	0	0,0,0,11,0,13

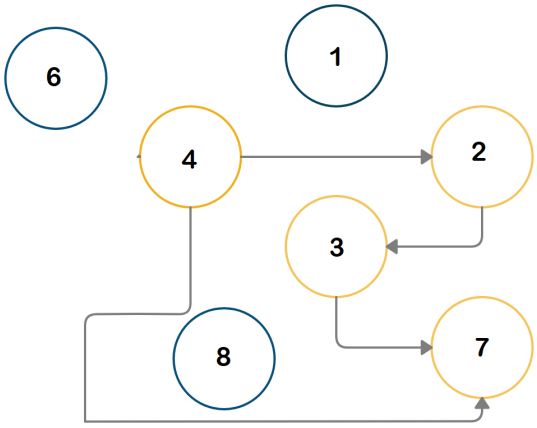
Table 6: vectors created for the Case 1



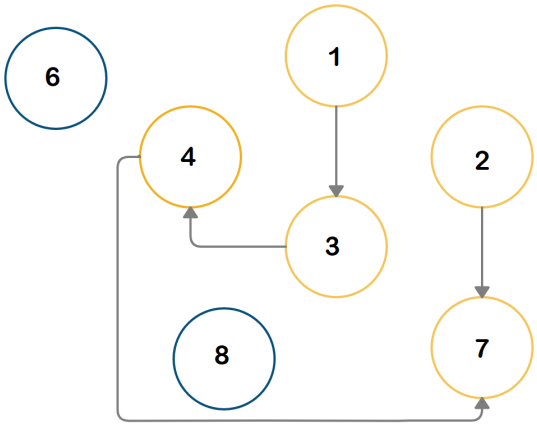
(a) Unclean pattern



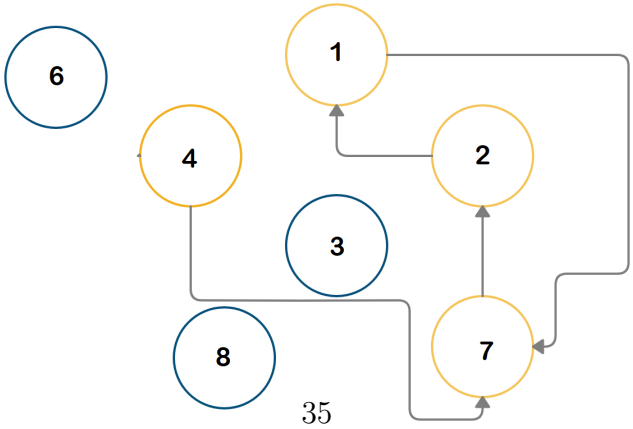
(b) Clean pattern



(c) Random graph 1



(d) Random graph 2



(e) Random graph 1000

Figure 10: Unclean pattern, clean pattern and random patterns from Case 1. Edges of all these patterns are shown in Table 5.

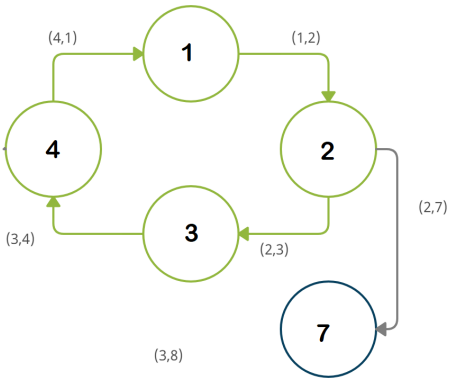
2. Case 2 : — The unclean graph considered in this example has low noise, which means it has very few non-interesting nodes. This example has only one noisy node marked in italics in Table 7. Vectors generated for the random graphs are sparser when compared to case 1. When we calculate vector similarity for this pattern, the result is shown in Figure 12 (b). Visualisation of patterns for this case is demonstrated in Figure 11. In Figure 11, (a) is the unclean pattern and (c), (d), (e) are the random patterns generated.

	Edge 1	**Edge 2**	**Edge 3**	**Edge 4**	**Edge 5**	**Edge 6**	**Edge 7**
Unclean pattern	(3,4)	(4,1)	(1,2)	(2,3)	*(2,7)*	-	-
Clean pattern	(3,4)	(4,1)	(1,2)	(2,3)	-	-	-
Random graph 1	(4,2)	(3,7)	(2,3)	(4,7)	-	-	-
Random graph 2	(1,3)	(2,7)	(4,7)	(3,4)	-	-	-
Random graph 1000	(7,2)	(4,7)	(1,7)	(2,1)	-	-	-

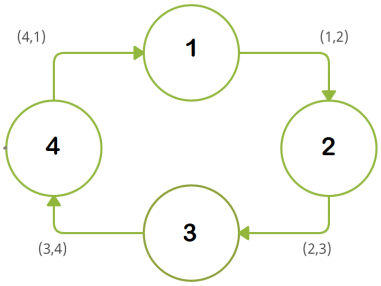
Table 7: Example pattern and random permutations for a pattern with low noise

	(1,2)	(1,3)	(1,4)	(1,7)	(2,1)	(2,3)	n such columns ->
Random graph 1	0	0	0	0	0	13	12,0,0,11,0,9
Random graph 2	0	13	0	0	0	0	0,0,0,11,0,13

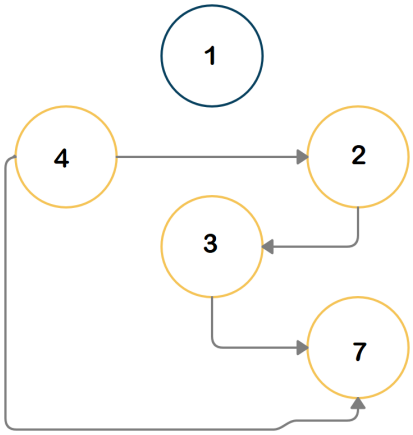
Table 8: vectors created for the Case 2



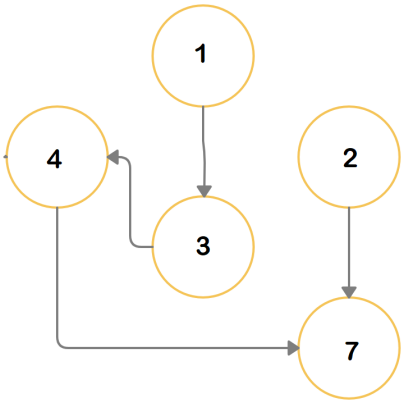
(a) Unclean pattern



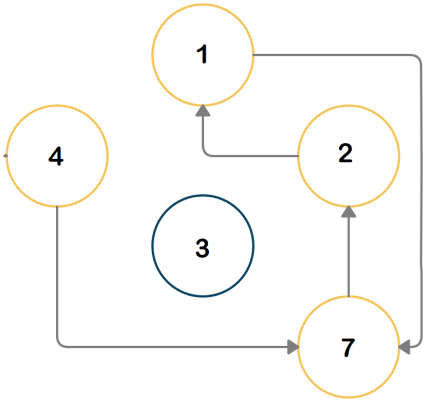
(b) Clean pattern



(c) Random graph 1

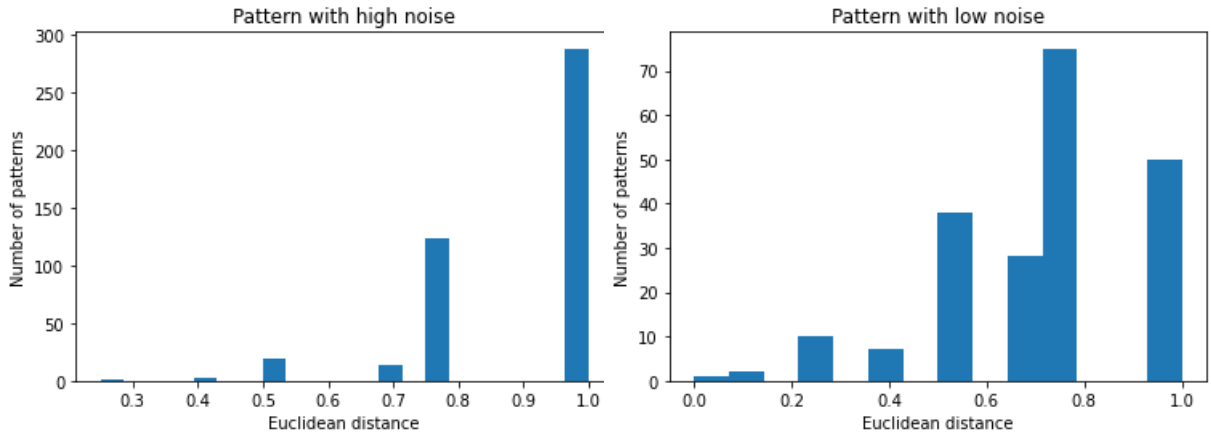


(d) Random graph 2



(e) Random graph 1000

Figure 11: Unclean pattern, clean pattern and random patterns from Case 2. Edges of all these patterns are shown in Table 7.



(a) The vector similarity score is high on average for a pattern with high noise

(b) Vector similarity score is low for a low noise pattern when compared to the high noise example.

Figure 12: These two figures show the comparison of results for a pattern with high noise and low noise. It is seen in Figure (a) that the pattern with high noise has less vector similarity with randomly generated graphs when compared to a pattern with low noise.

The vector similarity scores are the Euclidean distances between the query vector and the randomly generated vector set. The lower the Euclidean distance, the higher is the similarity. Mean observed for distribution in Figure 12 (a) is 0.93 and for distribution in Figure 12 (b) is 0.71. Using mean value, patterns can be ranked or by setting a threshold value they can be classified for being usual or unusual. The threshold value for mean value is selected based on the network being studied. For example, threshold value of mean in case of Bitcoin network should be lower than Ethereum network as it has denser structure than Ethereum. Setting lower threshold can result in increase in false positive rate. In this study, false positive rate is acceptable as in case of money laundering it will be better than having higher false negative rate. All the patterns having mean value more than threshold are termed as potential money laundering candidates.
Negative skew is observed if most of the Euclidean distances in the observed data are more than 0.5. The more the negative skew, the more is the chance of the pattern being unusual. On the other side, if the skew is positive, there is a high chance of the observed pattern being a usual pattern. Skew observed for the high noise pattern is -0.76, and for the low noise pattern, it is -0.26. Values of kurtosis test are -0.54 and -0.76 for high noise pattern and low pattern respectively. Negative values show that both of the patterns in this case are lightly tailed.

6 Results

In this chapter, we present results of both algorithms on Bitcoin and Ethereum datasets. Using Figures 14, 15, 17, 18 we demonstrate examples of the patterns found by algorithms. With Figure 16, We also show the effect of cleaning step executed in the algorithms. Figure 1 shows the ideal expected results. However, some pattern formations apart from ideal targeted patterns are observed. These patterns other than ideal structure are observed, as some patterns satisfy time and flow threshold conditions mentioned in Chapter 4 while not having the exact expected ideal structure as Figure 1. Nodes in patterns are displayed as numbers, however, in the Blockchain data of both cryptocurrencies, nodes are alphanumeric strings as previously

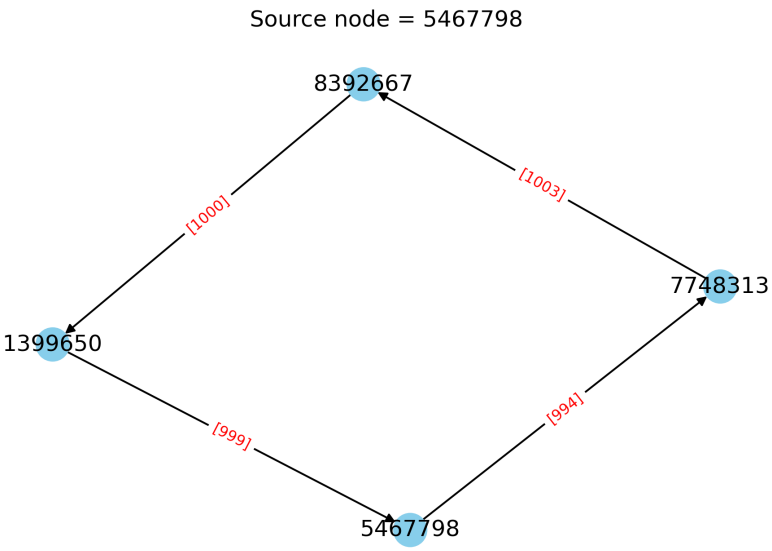
described. For simplicity of calculation and representation, all the nodes are mapped to unique numbers. Hence, each unique number represents a unique Bitcoin or Ethereum wallet. Each directed arrow shows the direction of transaction and the flow. Flow of a particular edge is mentioned in each figure in red font on the respective edge.

6.1 Patterns Discovered

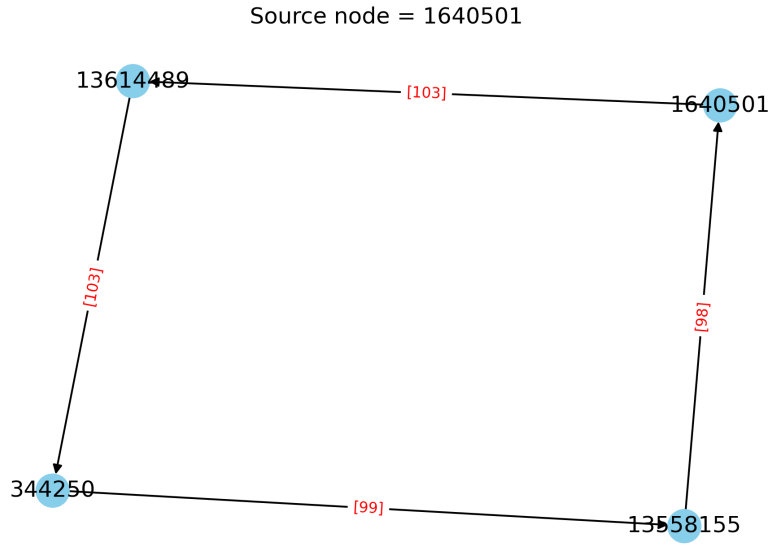
Ideal patterns show and describe pattern structures similar to Figure 1. Complex patterns show pattern structure for which N, M exceed when compared to patterns in Figure 1. For circular and diamond patterns, we show example of ideal patterns and complex pattern using Figures 14, 15, 17, 18.

6.1.1 Results of circular pattern

- **Ideal patterns:** Figure 14 shows an ideal pattern the algorithm yields. Patterns in Figures 14 (a), (b) are structurally similar to the circular pattern defined in Figure 1. The source node mentioned above each pattern in the figure is the address that initiates the pattern. As the Figure 14 is illustrating, circular patterns the source node is also the final receiver for the pattern. When compared to the example structure shown in Figure 1, source node is referenced as node Address 1 in example structure Figure 1 and as node A in the Figure 7. Arrows show the direction of the flow for both patterns in Figure 14. These patterns have four unique nodes involved, and only a single edge exists between a pair of two unique nodes. The amount sent by each address is shown in red font on the outgoing edge from that node. Amounts involved in the Figures 14 (a), (b) show a slight deviation, and adjustments made in amount are visible for both patterns. It can be observed that adjustments made in the amount of Figure 14 (a) do not show gradual increase or decay.



(a)



(b)

Figure 13

Figure 14: Ideal circular patterns observed in data with 4 nodes. Both patterns show expected circular structure where similar amounts being forwarded and received. The amounts are mentioned in red on respective edges.

- **Complex patterns:** Apart from ideal patterns shown in Figure 1, Some complex circular patterns are also discovered in both datasets. In a complex circular pattern, nodes are more than four, with a higher number of incoming and outgoing edges than an ideal pattern. The number of edges increases as there can be multiple transactions between two nodes. If a circular pattern is considered a single loop, then for a complex pattern, multiple such loops can exist. Figure 14 shows the simple structure where an amount is forwarded via single transactions between pairs of nodes. Although sending a large amount in a single transaction would help save transaction fees, a large amount can also

be sent using multiple individual transactions. The algorithm can track such patterns, and an example of one such complex pattern is shown in Figure 15. Figure 15 illustrates a pattern with 9 nodes. Multiple amounts are written on the edges for multiple transactions done between a pair of nodes.
Node 5467798 is the source and final receiver for all the transactions. Node 344250 is the support node in this pattern. If this pattern is compared with Figure 1 (a), node 5467798 would be `Address 1` and node 344250 would be `Address 3`. All the other nodes would be similar to nodes `Address 2` and `Address 4`. For this complex pattern, the flow involved is observed as follows.

$$\sum f(Node\_5467798_{out}) \approx \sum f(Node\_344250_{in}) \approx \sum f(Node\_344250_{out}) \approx \sum f(Node\_5467798_{in}) \tag{23}$$

This equation takes in account summation of inflow amounts and outflow amounts to put deviation threshold conditions. Time threshold with flow conditions in such a case is defined as follows.

$$f_t(Node\_5467798_{out}) > f_t(Node\_344250_{out}). \tag{24}$$

Such condition ensures the flow at a timestamp t for outgoing transactions of node 5467798 is greater than the flow of outgoing transactions of node 344250. As all transactions are going to be passed through node 344250, this conditioning ensures the source node first sends the amount before receiving any of it back. These conditions of time and flow ensure a meaningful structure for observed patterns.

In case of Figure 15 amount sent out is

$$962 + 96 + 934 + 544 + 73 + 50 + 41 + 99 + 39 + 460 = 3298. \tag{25}$$

Total of 10 outgoing transactions take place in between node 5467798 and node 344250 via 5 nodes in between. Amount received back at the node 5467798 is

$$1499 + 248 + 525 + 449 = 2721. \tag{26}$$

The difference between the amount sent and amount received is 577 Ethereum tokens, and the deviation is 288 Ethereum tokens. Based on the price average price of Ethereum at the time of this pattern, the difference would be ≈ 500 dollars in terms of fiat currency.

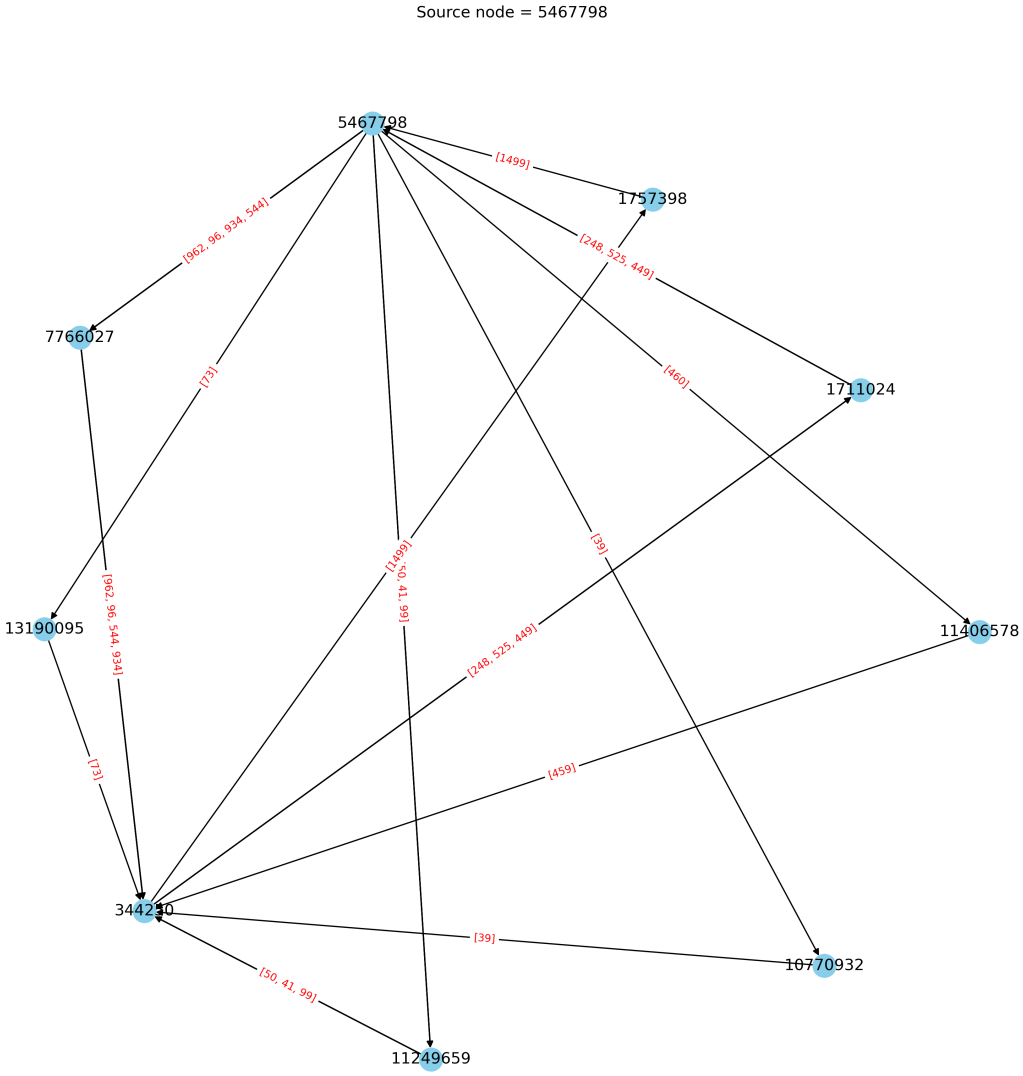
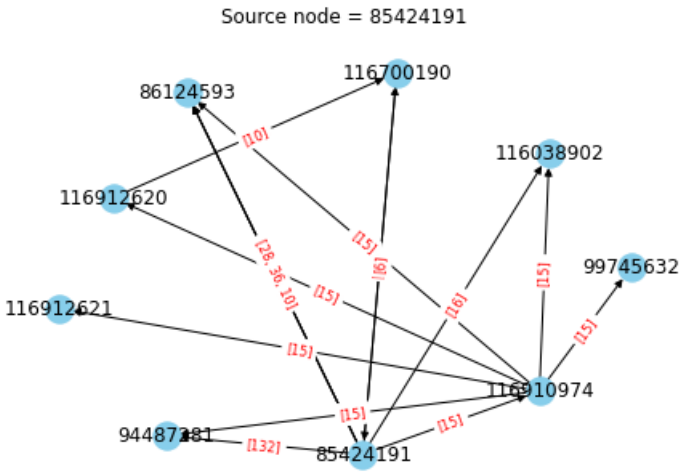
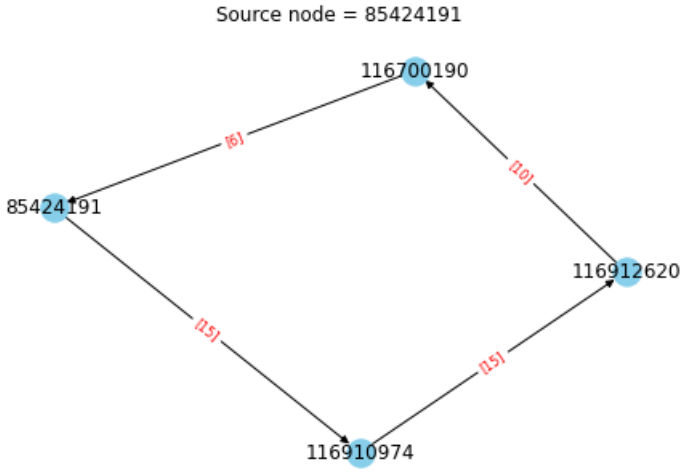


Figure 15: Example of a complex circular pattern having more than 1 loop and multiple transactions for the pair of source node and support node. The source node is 5467798 and the support node is 344250.

- **Effect of cleaning step:** Figure 16 demonstrates the effect of the cleaning step in the algorithm. Figure 16 (a) demonstrates the unclean pattern and (b) is the same pattern in clean stage. The unclean pattern in this Figure 16 is similar to the unclean pattern shown previously in Figure 9 (b). Compared with the clean pattern 16 (b), noisy nodes and edges are observed in Figure 16 (a). Nodes, such as node 9974, which do not forward any amount to any other nodes are removed. Node 116910974 has a circular pattern in unclean pattern with nodes 116910974, 94487381, 8524191. Selected source node 8524191 is also a part of this circular pattern occurring for node 116910974. However, as the source node selected is node 85424191, all other circular patterns are removed. This circular pattern will be tracked when the selected source node is 116910974. For the pattern in Figure 16, the search deviation threshold set was high enough to accept the flow 15, 10, 6, 15. Putting a low deviation threshold algorithm would discard the pattern and does not save it.



(a)



(b)

Figure 16: Example of a pattern in the unclean stage (a) and in the clean stage (b). In the clean pattern, the algorithm gets rid of all the unnecessary and noisy edges and nodes.

6.1.2 Results of diamond pattern

- **Ideal patterns:** Figure 17 shows the diamond patterns mined for the Ethereum data. The source node mentioned in Figure 17 is shown as node Address 1 in Figure 1 and as node A in Figure 8. The source node mentioned in the Figure initiates the pattern and divides a certain amount of Ethereum tokens in multiple layer 2 nodes. Layer 2 nodes for both Figures 17 (a), (b) are all the nodes except for the source node and sink node. As mentioned in the methods chapter, there is no maximum limit on the number of nodes in layer 2. Figure 17 (a) has 4 nodes in layer 2 while for Figure 17 (b) there are 8 nodes in layer 2. In the Figure 17 only a single outgoing transaction takes place in between the source node and each node in layer 2 and also each node in layer 2 and sink node. A minimal deviation is observed in the Figure 17 for the amounts being transferred. The amount received by a layer 2 nodes and the amount sent by a layer 2 nodes is almost equal in the case of both patterns shown in Figure 17. For Figure 17 (a) flow is observed

is

$$f(Source\_node_{out}) \approx \sum f(Layer2\_nodes_{in/out}) \approx f(Sink\_node_{in}) \tag{27}$$

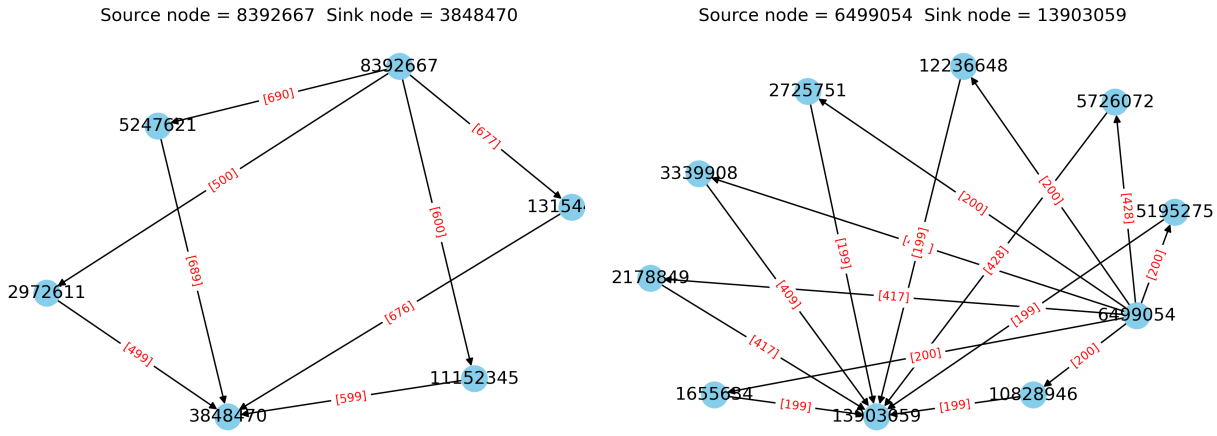
Where

$$Node\_8392667 \in Source\_node \tag{28}$$

$$Node\_274621, Node\_2972611, Node\_13154488, Node\_11152345 \in Layer2\_nodes \tag{29}$$

$$Node\_3848470 \in Sink\_node \tag{30}$$

Equations 27,28,29,30 demonstrate cryptocurrency tokens first being split into equal or unequal amounts among multiple addresses and then received at a common address later in time. Hence, From the patterns shown and flows involved in these patterns, it is observed that the algorithm yields expected diamond patterns.



(a) (b)

Figure 17: Example of diamond patterns. These two examples of patterns show the ideal expected diamond patterns discovered for Ethereum dataset.

- **Complex patterns:** Similar to the circular patterns, some complex patterns are also observed during the search for diamond patterns. Unlike complex circular patterns, the structure of complex diamond patterns is identical to the ideal structure in Figure 1 (b). The number of layer 2 nodes is high in complex diamond patterns, along with multiple edges between a pair of nodes. Figure 18 shows an example of such a complex diamond pattern occurring in the Ethereum network. This complex pattern is one of the largest patterns discovered in the Ethereum dataset. It consists of 181 unique nodes, 358 unique edges and a total of 1530 transactions. The size of the nodes is set as per their degree, and the size of edges is set as per their weight. All the green edges are the outgoing transactions from node 5467798. All the black edges are the incoming transactions for node 8590805. Weight is mentioned on every edge, and the font size is as per the amount of weight involved. Some large transactions, such as transactions with 8088 Ethereum tokens, is observed in Figure 18.

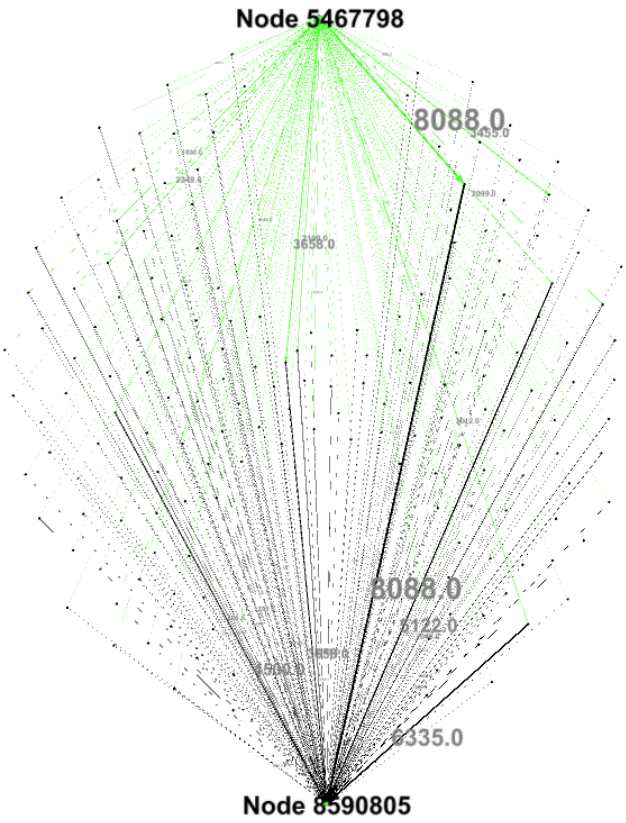


Figure 18: Ethereum: Example of complex diamond pattern having 181 nodes and 358 unique edges and total of 1530 transactions.

6.1.3 Quantitative evaluation

This subsection shows the number of patterns observed based on two metrics, the number of unique nodes N in patterns and deviation observed in patterns. Table 9 shows the number of circular and diamond patterns in both datasets.

- **Unique nodes in patterns:** Based on Table 9 it is observed that the total number of circular patterns is significantly less than diamond patterns in the Bitcoin dataset. Most of the circular patterns observed in the Bitcoin dataset have four unique nodes. Although some circular patterns have more than four unique nodes, there are no patterns with more than 64 unique nodes. The total number of circular patterns observed in the Ethereum dataset is slightly less than the Bitcoin dataset. More than 90% of circular patterns in Ethereum dataset have four or fewer unique nodes. However, very few circular patterns are of substantial size in Ethereum data.
  General observation for diamond patterns in Table 9 states that the occurrence of diamond patterns is frequent in the Bitcoin dataset but not in Ethereum dataset. Diamond patterns with $N \leq 4$ are less than patterns with $4 < N \leq 8$ for both datasets. Large diamond patterns such as $8 < N \leq 64$ and $N > 64$ also have a significant percentage compared to total patterns.

	Number of circular patterns				Total circular patterns	Number of diamond patterns				Total diamond patterns
Dataset	$N \leq 4$	$4 < N \leq 8$	$8 < N \leq 64$	$N < 64$		$N \leq 4$	$4 < N \leq 8$	$8 < N \leq 64$	$N > 64$	
Bitcoin	10k	1.2k	822	0	12.5k	84.5k	93.8k	44.1k	13.4k	236.1k
Ethereum	10.1k	205	219	18	10.5k	304	1.3k	1.3k	74	3k

Table 9: Table demonstrates the number of patterns and unique nodes in pattern for both datasets. N is the number of unique nodes.

- **Flow deviation in patterns:** Table 10 shows the number of patterns based on the deviation observed. For circular patterns in the Bitcoin dataset, the deviation observed for almost 40% of patterns is less than 10. Rest 60% patterns are divided in deviation range of $10 < \phi \leq 50$, $50 < \phi \leq 100$ and $\phi > 100$. Deviation for circular patterns in Ethereum Dataset is high on average when compared to Bitcoin dataset. About 50% circular patterns show deviation $\phi > 100$.
  Deviation observed for diamond pattern in Table 10 illustrates that both datasets show high deviation for diamond patterns. There are very few diamond patterns in both datasets that have deviation $\phi < 10$.

	Number of circular patterns				Number of diamond patterns			
Dataset	$\phi \leq 10$	$10 < \phi \leq 50$	$50 < \phi \leq 100$	$\phi > 100$	$\phi \leq 10$	$10 > \phi \leq 50$	$50 > \phi \leq 100$	$\phi > 100$
Bitcoin	4.9k	3.6k	1.6k	2.2k	523	6.1k	1.4k	227.9k
Ethereum	1.2k	3.1k	1.2k	4.8k	49	7	29	3k

Table 10: Number of patterns and deviation observed for the patterns. Patterns are split based on deviation of the flow, where ϕ is the deviation.

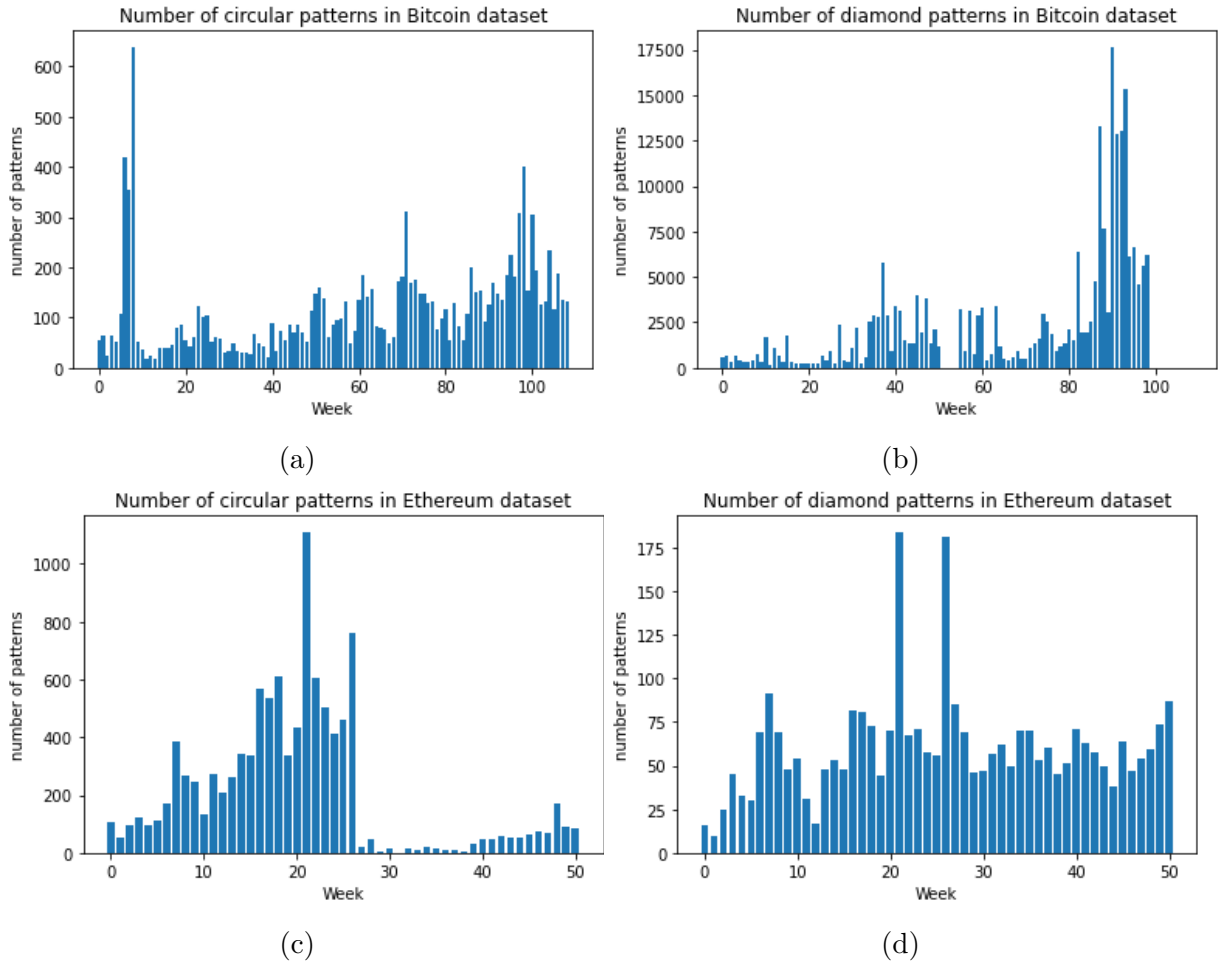
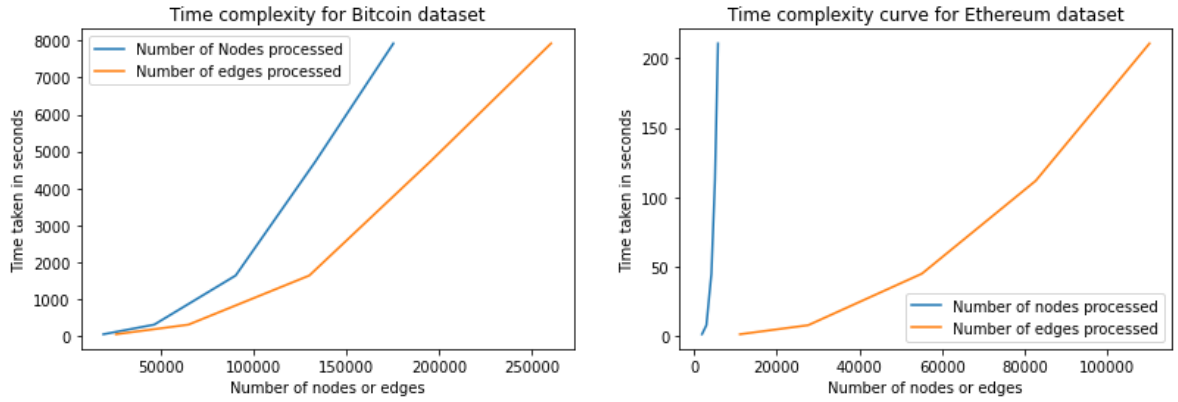


Figure 19: Number of pattern in every data split, i.e. every week. The Bitcoin dataset shows gradual rise for both patterns. There is a sharp drop in Ethereum dataset for circular patterns after week 27. However, no such drop is observed for diamond patterns in Ethereum dataset.

- **Time complexity analysis:** Table 11 shows the time taken by algorithms and nodes and edges processed in that time. Compared to Bitcoin, splits of Ethereum data have less number of unique nodes. However, the combination of fewer nodes and a high number of edges creates dense graph structures in Ethereum data. Hence, the time complexity for both datasets is different. To find all the complex patterns, algorithms iteratively search the dataset. A high number of support nodes in the case of Ethereum make the algorithms slow. Regarding edges, we can observe quadratic time complexity for both datasets in Figure 20. The time complexity of both algorithms is similar as they are built on a similar approach.

	Bitcoin					Ethereum				
Unique nodes	19k	46k	90k	133k	175k	1.8k	2.9k	4.1k	4.9k	5.7k
Number of edges	26k	65k	130k	196k	260k	11k	27k	55k	82k	111k
Time in seconds	49	310	1639	4721	7922	1.4	7.9	44	111	200

Table 11: Time complexity of algorithms for both datasets. Table shows time taken in seconds and nodes and edges processed in that time.



(a) (b)

Figure 20: Time complexity curves for both datasets.

6.2 Relevance evaluation

In this subsection, we present the result of the relevance evaluation. The relevance evaluation is conducted for every pattern observed. For a set of patterns observed for a data split, the relevance evaluation ranks them from most unusual to most usual by evaluating the risk associated with them. In this chapter, we present results for one example split. As mentioned earlier, the relevance evaluation consists of three different tests. All three tests assign their rankings to patterns. In most cases, all three tests commonly term a pattern as most unusual. One such example of an unusual diamond pattern is shown in Figure 21. The pattern in Figure 21 shows an exceptionally high number of nodes within a week's time frame. In some cases, multiple transactions have taken place between two nodes. Amounts being sent and received are almost identical with little or no deviation.
Figure 22 (a) shows the distribution of Euclidean distances for this most unusual pattern. The mean observed for this pattern is 0.991. The high mean value shows that most randomly generated patterns do not show similarity in structure or weights observed. The kurtosis value is 11.5, and the skewness test value is -3.07. A high positive kurtosis value and a negative skew value show that most random patterns are not similar. This observation of positive kurtosis and negative skew goes well along with the very high mean value observed. Figure 23 demonstrates possible usual patterns. All three tests select different patterns as potential candidates for usual patterns. Compared to pattern in Figure 21, patterns in Figure 23 are less complex and have low weights.

Example of usual patterns is demonstrated in Figure 23. Unlike unusual patterns, for usual patterns choices of the tests are different. Similarity distribution for Figure 23 is illustrated in Figure 22 (b). Compared to Figure 22 (a) it can be seen that this distribution has lower mean, Although the skew is negative, the kurtosis value is positive as it is a heavy tail distribution. Comparing Figure 22 (a) and (b) difference between similarity distributions of possible usual pattern and unusual pattern is evident. Figure 23 (b) and (c) are usual patterns for skew test and kurtosis test respectively. Compared to unusual pattern, both Figure 23 (b) and (c) have low weights and simple structure.

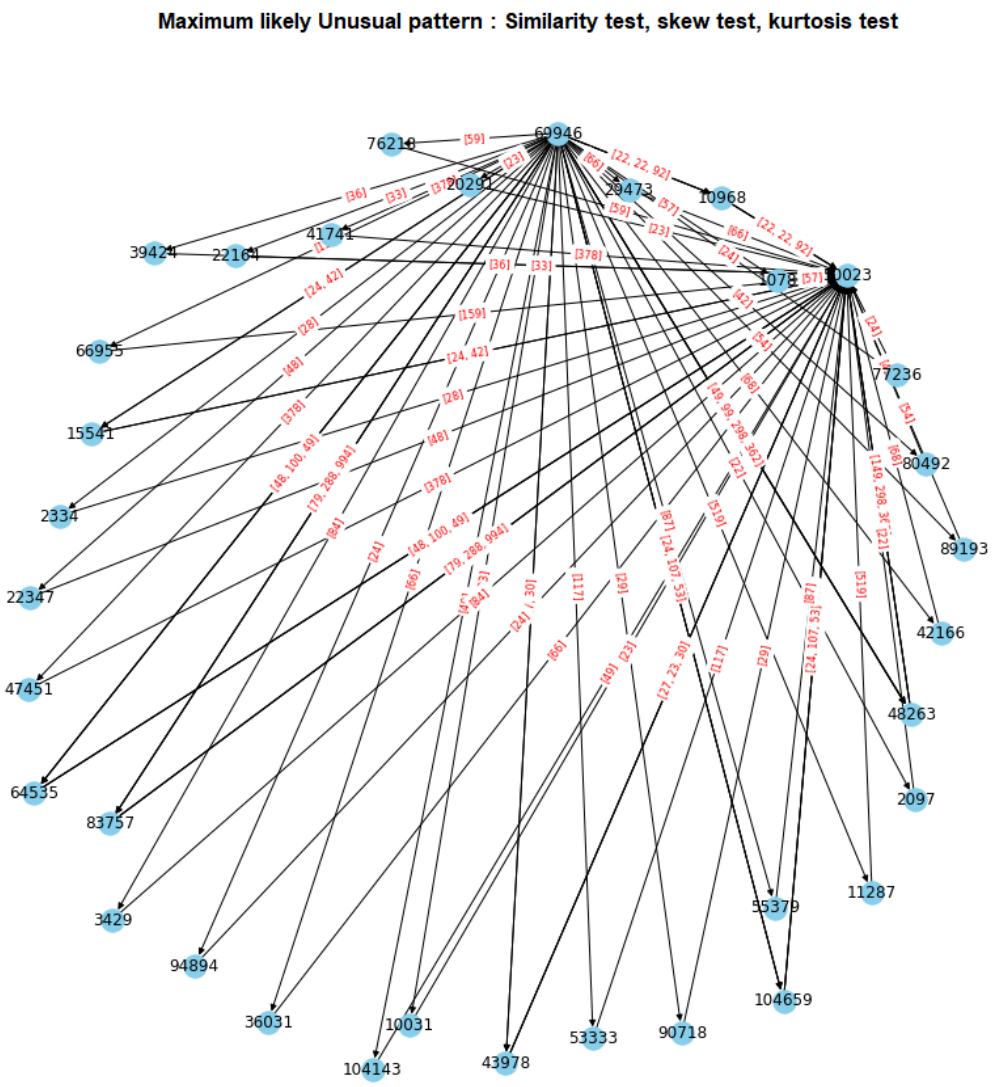
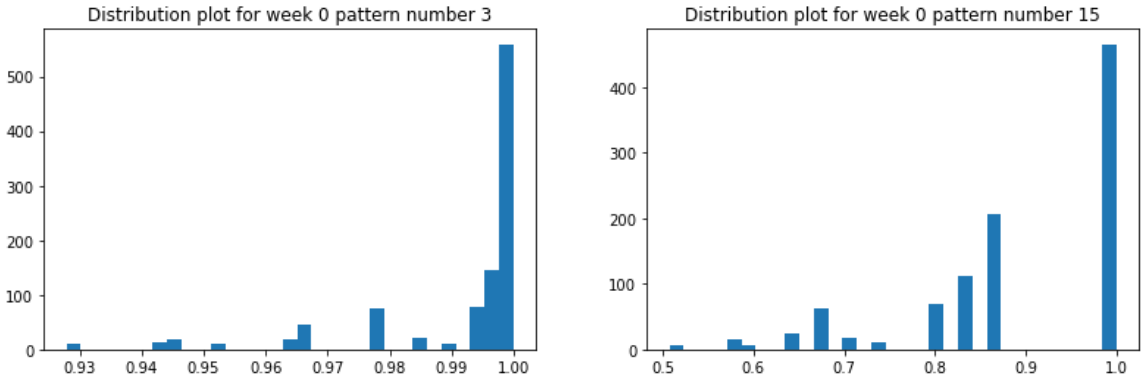
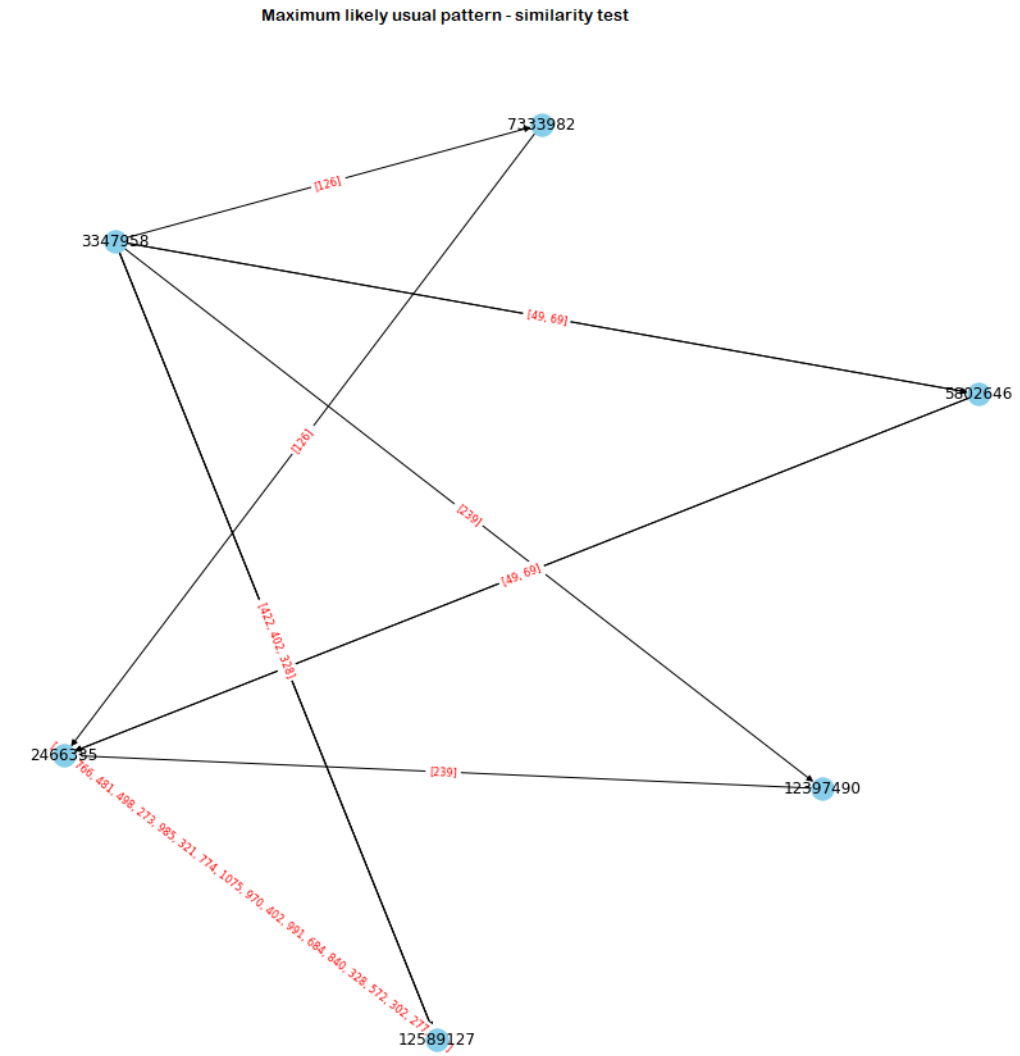


Figure 21: Most unusual pattern for a week. It is observed that it has diamond structure and amounts being sent and received between node 30023 and node 69946. Very little deviation is observed in the amounts.



(a) (b)

Figure 22: Distribution of Euclidean distances seen for most unusual pattern (a) and a probable usual pattern (b).



(a)

(b)

(c)

Figure 23: Patterns termed as usual by three tests, (a) usual patterns as per similarity test, (b) as per skew test and (c) as per kurtosis test. Unlike unusual pattern, results for usual pattern vary for all three tests.

6.2.1 Quantitative evaluation

In this subsection, we present a number of patterns observed for each threshold value of similarity test, skew test, and kurtosis test. Tables 12, 13, 14 show results for similarity test, skew test and kurtosis test respectively.

- Similarity test:

	Number of circular patterns					Number of diamond patterns				
	$\alpha \leq 0.6$	α >0.6	α >0.75	α >0.95	α >0.99	$\alpha \leq 0.6$	α >0.6	α >0.75	α >0.95	α >0.99
Bitcoin	1	12.4k	12.4k	1.6k	295	20.8k	18.8k	79.3k	7.4k	412
Ethereum	0	10.5k	10.4k	663	209	1	3k	3k	1.4k	312

Table 12: Similarity test results: Table shows number of patterns for different thresholds for similarity test. α is the threshold for mean of similarity scores.

Table 12 shows for circular patterns in Bitcoin dataset almost no pattern show mean less than 0.6. The mean of most of the circular patterns in Bitcoin dataset is in the range 0.6 to 0.75. A significant number of patterns have mean more than 0.95 and a few patterns show mean value more than 0.99. Circular patterns in Ethereum dataset have similar structure for results compared to Bitcoin dataset. No patterns show mean less than 0.6 in case of Ethereum dataset. Most of the patterns fall in the range of 0.6 to 0.95. Compared to total circular patterns observed in the Ethereum dataset, small percentage of patterns shoe mean more than 0.99.
Large number of diamond patterns in Bitcoin dataset show mean less than 0.6. Similar to circular patterns, most of the patterns are in the range of 0.6 to 0.95. Compared to total number of diamond patterns, less than 1% patterns show mean more than 0.99.

- Skew test:

	Number of circular patterns					Number of diamond patterns				
	$\beta \leq$-10	β <-5	β <0	β >0	β >5	$\beta \leq$-10	β <-5	β <0	β >0	β >5
Bitcoin	10	93	12.1k	302	0	207	29	156k	53k	33
Ethereum	115	4	10.4k	118	0	85	1	3k	10	0

Table 13: Skew test results: Number of patterns observed for different threshold of skew test. It can be observed that most patterns have skew in the range -5 to 5. β is the threshold used for the skew test.

Table 13 shows results of skew test for circular and diamond patterns on both datasets. In case of the skew test, results are identical for both patterns in both datasets. Most of the patterns show negative skew and only few patterns have positive skew. Only in diamond patterns for Bitcoin data, few patterns showed a positive skew.

- Kurtosis test:

	Number of circular patterns						Number of diamond patterns					
	$\gamma \leq$-10	$\gamma <$-5	$\gamma <$0	$\gamma >$0	$\gamma >$5	$\gamma >$10	$\gamma \leq$-10	$\gamma <$-5	$\gamma <$0	$\gamma >$0	$\gamma >$5	$\gamma >$10
Bitcoin	0	10	5055	7.4k	794	258	0	0	167k	41k	13k	719
Ethereum	0	0	4.01k	6.5k	676	253	0	0	939	2.1k	504	261

Table 14: Kurtosis test results: Results of kurtosis test for different thresholds. Almost no patterns have high negative kurtosis value. γ is the threshold used for kurtosis test.

Results of the kurtosis test in Table 14 having high negative value are identical for circular patterns and diamond patterns in both datasets. None of both patterns show high number of patterns, high negative value of kurtosis associated with them. For circular patterns, about 40% of total patterns have kurtosis value in between -5 to 0. Rest of the circular patterns are divided in the range of 0 to 10. A significant number of patterns show kurtosis value higher than 10 for circular patterns.
For diamond patterns, no patterns show a high negative kurtosis value. Large number of diamond patterns in Bitcoin dataset have values between -5 to 0. In contrast, diamond patterns in Bitcoin dataset mostly have values more than 0.

7 Discussion

This chapter presents an interpretation of the results observed for the circular and diamond patterns in both datasets. We also discuss the link of the results with multiple aspects of Blockchain and cryptocurrencies.
Based on Figure 14 it is observed that the circle track algorithm yields the expected results of circular patterns. Conditions formulated in Chapter 4 for flow and timestamps find expected circular patterns. Figure 14 illustrates that the flow in the patterns is adjusted in every transaction. Adjustments for flow in Figure 14 do not show gradual increase or decay. These adjustments can be due to the price volatility of cryptocurrencies. Hence, calculating deviation based on layers and not using $<$ as a condition for amounts works well. As described in the subsection 5.1 in the algorithm chapter, a pair of source nodes and support nodes are processed in parallel while finding their neighbours. This parallel search serves as an advantage as it assists the algorithm to track all the patterns occurring for a pair in a single search. This advantage of parallel search helps in searching for patterns slightly more complex than patterns shown in Figure 14. Multiple circular patterns observed in Figure 15 are results of a hybrid approach of BFS and DFS in circle track algorithm.
Searching diamond patterns while following the conditions from Chapter 4 is slightly complex, as there is no limit on layer 2 nodes in a diamond pattern. Figure 17 shows algorithm diamond track can find targeted diamond patterns while obeying flow and timestamp conditions. Based on flows involved in Figure 17 it is observed that putting conditions based on flow and timestamp works well and yields expected diamond patterns. Not having a limit on layer 2 nodes helps to track large diamond patterns as shown in Figure 18. Such large diamond patterns are only observed after using a high ϕ value. As the number of transactions is high, the deviation observed in such patterns is also high.

In Table 9 number of unique nodes N are described for each pattern type in both datasets. Based on Table 9 it can be seen that average N in circular patterns in 4. However, the average N in the diamond patterns exceeds 4. These results about the average size of N are relevant as any suspicious activity would try to be hidden as much as possible. Having numerous layer 2 nodes in diamond patterns helps to split the amount in tiny chunks that are prone to unnoticed. Hence, most diamond patterns in both datasets consider splitting data in more than four but less than 64 layer 2 nodes. Figure 19 shows the number of patterns observed every week. In the Bitcoin dataset, both diamond and circular patterns are on a linear rise. Few weeks show a sudden rise for both patterns. The sudden rise in patterns is likely to be correlated with the price of Bitcoin, as shown in Figures 27, 26, 25, 24. Patterns in Ethereum dataset do not show any trend. The sudden drop for circular patterns in the Ethereum dataset is unexplained. Such drop also does not show any correlation with Ethereum Blockchain metrics like number of transactions, number of users, network hash rate (Refer to Ethereum Blockchain metrics). About 50% of total circular patterns in the Bitcoin dataset observed in Table 10 show a very low deviation. This observation is also linked with the price of Bitcoin. Due to the exponential rise in the price of Bitcoin, adjustments of flow in a transaction are reducing. For example, when the Bitcoin price was 1000 dollars, an adjustment of 10,000 dollars in a transaction would show a change of 10 Bitcoins. However, when the Bitcoin price was 10,000 dollars, an adjustment would show a difference of only 1 Bitcoin. However, In the case of diamond patterns, the deviation observed is very high. Both Bitcoin and Ethereum datasets show that most of the diamond patterns have significant deviation. Hence, based on these results, it can generally be claimed that diamond patterns show high deviation than circular patterns. High deviation in diamond patterns could most likely correlate with the number of nodes in layer 2 or the number of transactions in a pattern.

Results of relevance test in Figure 21 show that all three tests, i.e., similarity test, skew test and kurtosis test, point towards the same pattern for being unusual. Figure 22 (a) shows most of the Euclidean distances have high dissimilarity for the pattern shown in Figure 21. Further, the distribution is also seen to be lightly tailed. Hence, based on Figures 22, 21 it observed that statistical tests can mark correct patterns as unusual. Opposite to this, in Figure 23 it can be observed that patterns marked as safe usually have less complex structure compared to Figure 21. While classifying patterns, all three tests point towards different patterns as usual patterns. As each test evaluates separate metrics, the choice of their classification for usual patterns differs. Based on Figure 23 (a) it can be observed that there are numerous transactions between $Node\_12589127$ and $Node\_12589127$. Along with this, weights involved in Figure 23 (a) are significantly high. Still, the relevance evaluation terms the pattern as safe; hence it is observed that results of the relevance evaluation are not biased towards any factor of a pattern like the number of unique nodes, the number of transactions or total flow. Relevance evaluation can successfully find the most unusual pattern in a set of results. The combination of weighted vectors with randomisation works better than only randomising weights for circular and diamond patterns.
Based on the results of this study in Chapter 6 it was observed that circular patterns and diamond patterns are evident in transaction networks of Bitcoin and Ethereum. Based on the total number of patterns and size of patterns, it is hard to deny that money laundering is happening in Bitcoin and Ethereum. The linear rise in the number of patterns and total patterns for Bitcoin shows that Bitcoin is more prone to be associated with suspicious activities than Ethereum. Observing network, exponential rise, and users' total number, preference to Bitcoin

for laundering money is likely. Observation of large size of patterns and number of patterns for both Bitcoin and Ethereum dataset, signal towards some algorithmic service being involved in assisting suspicious transaction activities.

For future work, some changes in approach and methods can be done to find more relevant results. Only one time constraint of 1 week was considered in this study. Rather than considering a large overall size of data, a small portion of the data can be analysed for multiple time windows. Results of such analysis are most likely to yield larger patterns. Although both algorithms can find exact targetted patterns, scalability and efficiency were not considered as a priority in this study. Both scalability and efficiency can be improved by parallelising the execution of code. As both algorithms were implemented without having any significant dependency on supporting packages, PySpark could be a solution for parallelising the algorithms. Graph network metrics such as centralities were not considered in this study. Integrating such metrics can help improve the performance of the algorithms. Relevance evaluation can be assisted with a more statistical test such as KS test and t-tests. Also, the number of random patterns in the relevance evaluation can be optimised. This optimisation should help observe ideal normal distribution for Euclidean distances of usual patterns and make classification easier.

8 Conclusion

Using graph pattern mining, we analysed Blockchain transaction data of two cryptocurrencies, Bitcoin and Ethereum. We created two algorithms that can find money laundering patterns, such as circular and diamond patterns. We tried to combine multiple existing approaches available in graph pattern mining to create our algorithms. Based on results of this study, it can be confirmed that money laundering patterns such as circular patterns and diamond patterns are evident in both Bitcoin and Ethereum Blockchains. Numerous suspicious patterns of different sizes and monetary value were observed for multiple short time windows in both datasets.
We also used LSH to create a novel approach for testing relevance of the discovered patterns. Relevance evaluation conducted in this study shows it has effectiveness in classifying or ranking patterns as usual or unusual observation. Results of the relevance evaluation show that combined randomization of weights and edges is an effective way to assess the quality of the patterns.
Compared to total number of transactions, there is only a small portion of total transactions that is associated with suspicious activities. However, the size of both circular and diamond patterns is significant and often involves complex pattern structures. We found linear growth in Bitcoin patterns, but no such trend was observed for Ethereum patterns. To summarise this study, we analysed datasets of Bitcoin and Ethereum transactions for money laundering patterns previously tested on fiat currencies like USD and Euro. Creating 2 algorithms and a novel concept of relevance evaluation, we contributed a framework to find and evaluate suspicious graph patterns in Blockchain networks.

References

[1] C. Cachin et al. "The Transaction Graph for Modeling Blockchain Semantics". In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 1070.

[2] Priyanka Sain and Shalini Puri. "Detection of money laundering accounts using data mining techniques". In: Mar. 2018.

[3] Chen Zhao and Yong Guan. "A GRAPH-BASED INVESTIGATION OF BITCOIN TRANSACTIONS". In: *Advances in Digital Forensics XI*. Ed. by Gilbert Peterson and Sujeet Shenoi. Cham: Springer International Publishing, 2015, pp. 79–95. ISBN: 978-3-319-24123-4.

[4] Xiangfeng Li et al. "FlowScope: Spotting Money Laundering Based on Graphs". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 34 (Apr. 2020), pp. 4731–4738. DOI: 10.1609/aaai.v34i04.5906.

[5] Deepak Garg and Hemant Sharma. "Comparative Analysis of Various Approaches Used in Frequent Pattern Mining". In: *International Journal of Advanced Computer Science and Applications* 1 (Sept. 2011). DOI: 10.14569/SpecialIssue.2011.010323.

[6] Rakesh Agrawal, Tomasz Imielinski, and Arun Swami. "Mining Association Rules between Sets of Items in Large Databases". In: *IN: PROCEEDINGS OF THE 1993 ACM SIGMOD INTERNATIONAL CONFERENCE ON MANAGEMENT OF DATA, WASHINGTON DC (USA*. 1993, pp. 207–216.

[7] Stanley Oliveira and Osmar Zaïane. "Privacy Preserving Frequent Itemset Mining". In: *Proceedings of the IEEE ICDM Workshop on Privacy, Security and Data Mining* (June 2003).

[8] Chrysanthi Kosyfaki et al. *Flow Motifs in Interaction Networks*. 2018. arXiv: 1810.08408 [cs.SI].

[9] Rolf Wegberg, Jan-Jaap Oerlemans, and Oskar Deventer. "Bitcoin money laundering: mixed results? An explorative study on money laundering of cybercrime proceeds using bitcoin". In: *Journal of Financial Crime* 25 (Mar. 2018), pp. 00–00. DOI: 10.1108/JFC-11-2016-0067.

[10] Cazabet Remy, Baccour Rym, and Latapy Matthieu. "Tracking Bitcoin Users Activity Using Community Detection on a Network of Weak Signals". In: *Complex Networks & Their Applications VI* (Nov. 2017), pp. 166–177. ISSN: 1860-9503. DOI: 10.1007/978-3-319-72150-7_14. URL: http://dx.doi.org/10.1007/978-3-319-72150-7_14.

[11] R. Beck. "Beyond Bitcoin: The Rise of Blockchain World". In: *Computer* 51.02 (Feb. 2018), pp. 54–58. ISSN: 1558-0814. DOI: 10.1109/MC.2018.1451660.

[12] Nikhil Vadgama and Paolo Tasca. "An Analysis of Blockchain Adoption in Supply Chains Between 2010 and 2020". In: *Frontiers in Blockchain* 4 (2021), p. 8. ISSN: 2624-7852. DOI: 10.3389/fbloc.2021.610476. URL: https://www.frontiersin.org/article/10.3389/fbloc.2021.610476.

[13] Jie Shen et al. "Identity Inference on Blockchain using Graph Neural Network". In: *CoRR* abs/2104.06559 (2021). arXiv: 2104.06559. URL: https://arxiv.org/abs/2104.06559.

[14] Ting Liu et al. “An Investigation of Practical Approximate Nearest Neighbor Algorithms”. In: *Proceedings of the 17th International Conference on Neural Information Processing Systems*. NIPS’04. Vancouver, British Columbia, Canada: MIT Press, 2004, pp. 825–832.

[15] Alexandr Andoni and Ilya Razenshteyn. *Optimal Data-Dependent Hashing for Approximate Near Neighbors*. 2015. arXiv: 1501.01062 [cs.DS].

[16] Shikhar Gupta. *Locality Sensitive Hashing*. Apr. 2019. URL: https://towardsdatascience.com/understanding-locality-sensitive-hashing-49f6d1f6134.

[17] Dorit Ron and Adi Shamir. “Quantitative Analysis of the Full Bitcoin Transaction Graph”. In: *Financial Cryptography and Data Security*. Ed. by Ahmad-Reza Sadeghi. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 6–24. ISBN: 978-3-642-39884-1.

[18] Sadia Tariq, Muhammad Saleem, and Muhammad Shahbaz. “User Similarity Determination in Social Networks”. In: *Technologies* 7.2 (2019). ISSN: 2227-7080. DOI: 10.3390/technologies7020036. URL: https://www.mdpi.com/2227-7080/7/2/36.

[19] Stephen Ranshous et al. “Exchange Pattern Mining in the Bitcoin Transaction Directed Hypergraph”. In: *Financial Cryptography and Data Security*. Ed. by Michael Brenner et al. Cham: Springer International Publishing, 2017, pp. 248–263. ISBN: 978-3-319-70278-0.

[20] Andrea Fronzetti Colladon and Elisa Remondi. “Using social network analysis to prevent money laundering”. In: *Expert Systems with Applications* 67 (2017), pp. 49–58. ISSN: 0957-4174. DOI: https://doi.org/10.1016/j.eswa.2016.09.029. URL: https://www.sciencedirect.com/science/article/pii/S0957417416305139.

[21] Ramiro Camino et al. “Finding Suspicious Activities in Financial Transactions and Distributed Ledgers”. In: Nov. 2017, pp. 787–796. DOI: 10.1109/ICDMW.2017.109.

[22] Ahmad Aljabr, Avinash Sharma, and Kailash Kumar. “Mining Process in Cryptocurrency Using Blockchain Technology: Bitcoin as a Case Study”. In: *Journal of Computational and Theoretical Nanoscience* 16 (Oct. 2019), pp. 4293–4298. DOI: 10.1166/jctn.2019.8515.

9 Appendix

9.1 Terminology and definitions

Term	Explanation
Address	Alphanumeric string which acts as an account number for receiving and sending cryptocurrencies
Clean pattern	Pattern with only selected nodes
Coin treasury transaction	A transaction in which cryptocurrency miners are rewarded
Edge	A link between two addresses created by transaction
Flow	Amount of cryptocurrency tokens transferred
Master node	A server where Blockchain data is stored
Miners	Computers that validate the transactions and add them in the Blockchain
Node	A wallet address represented in a graph.
Unclean pattern	Pattern with selected nodes as well as some noisy nodes

Table 15: Terms and their meaning

9.2 Additional results

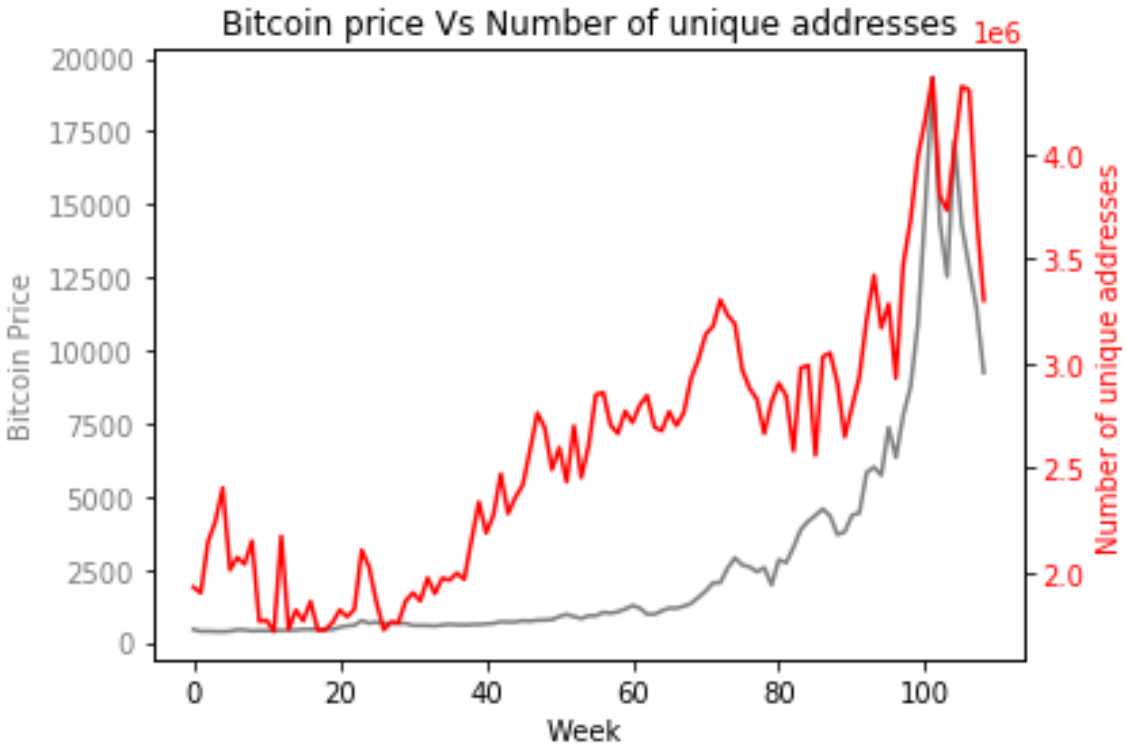


Figure 24: Number of unique addresses vs price.

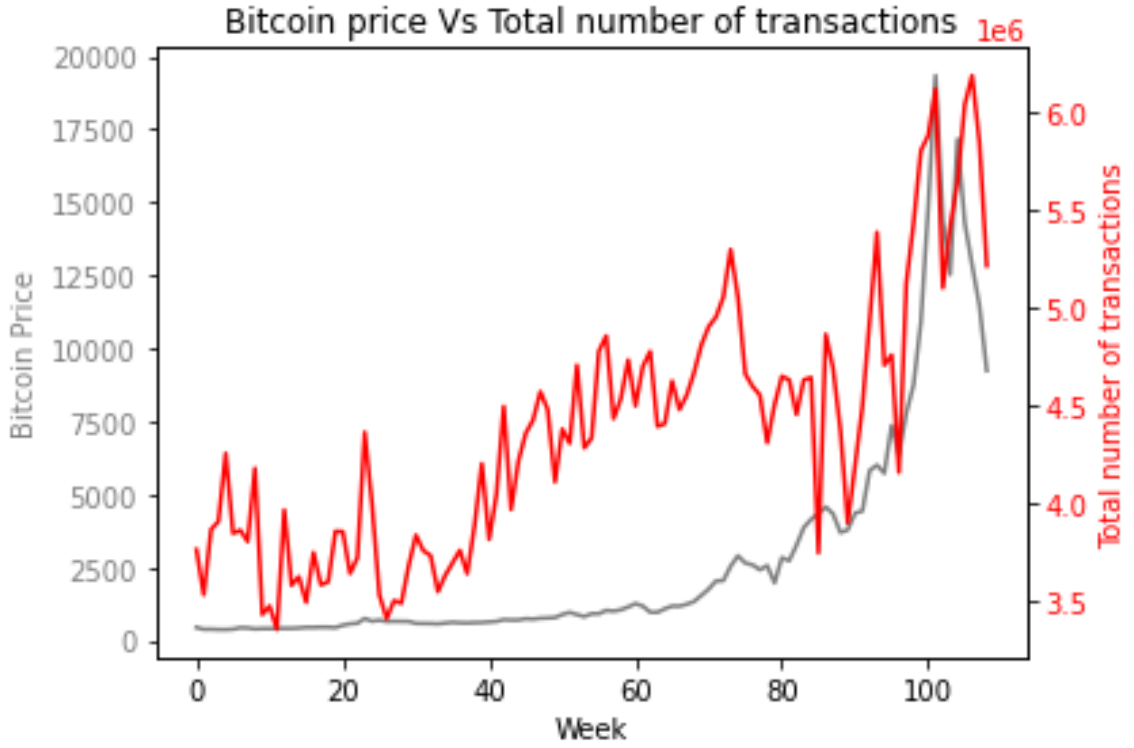


Figure 25: Number of transactions vs price.

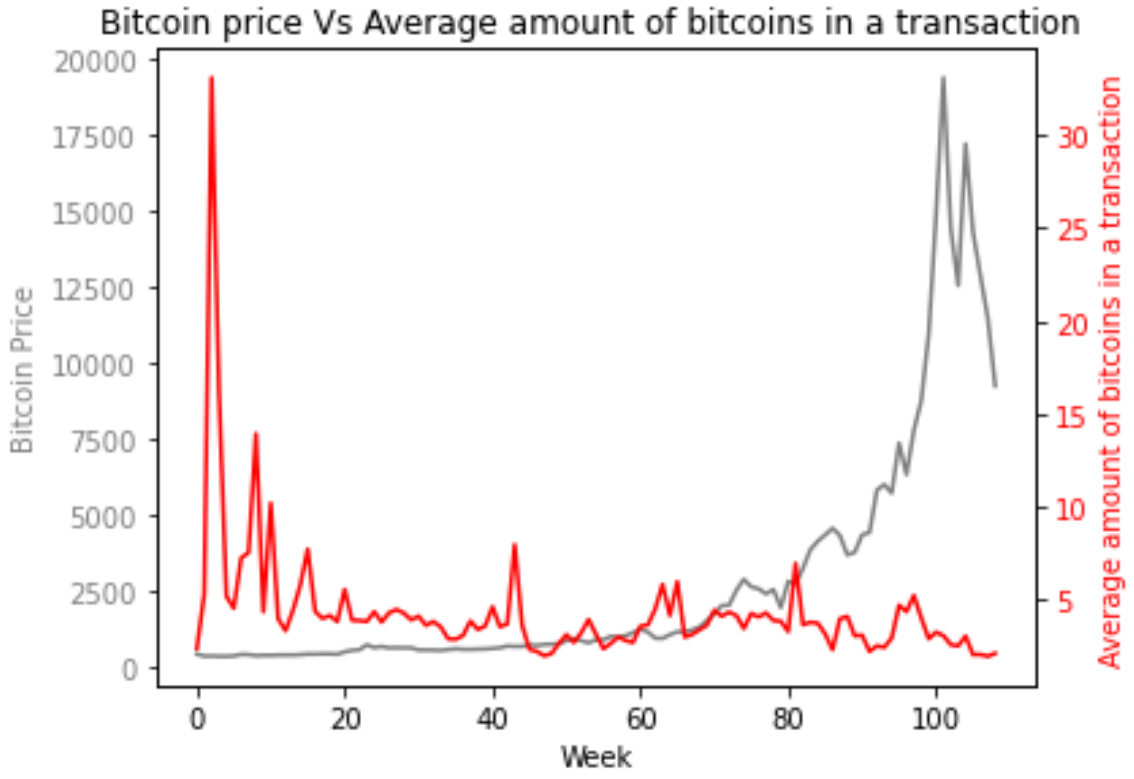


Figure 26: Average amount of transactions vs price.

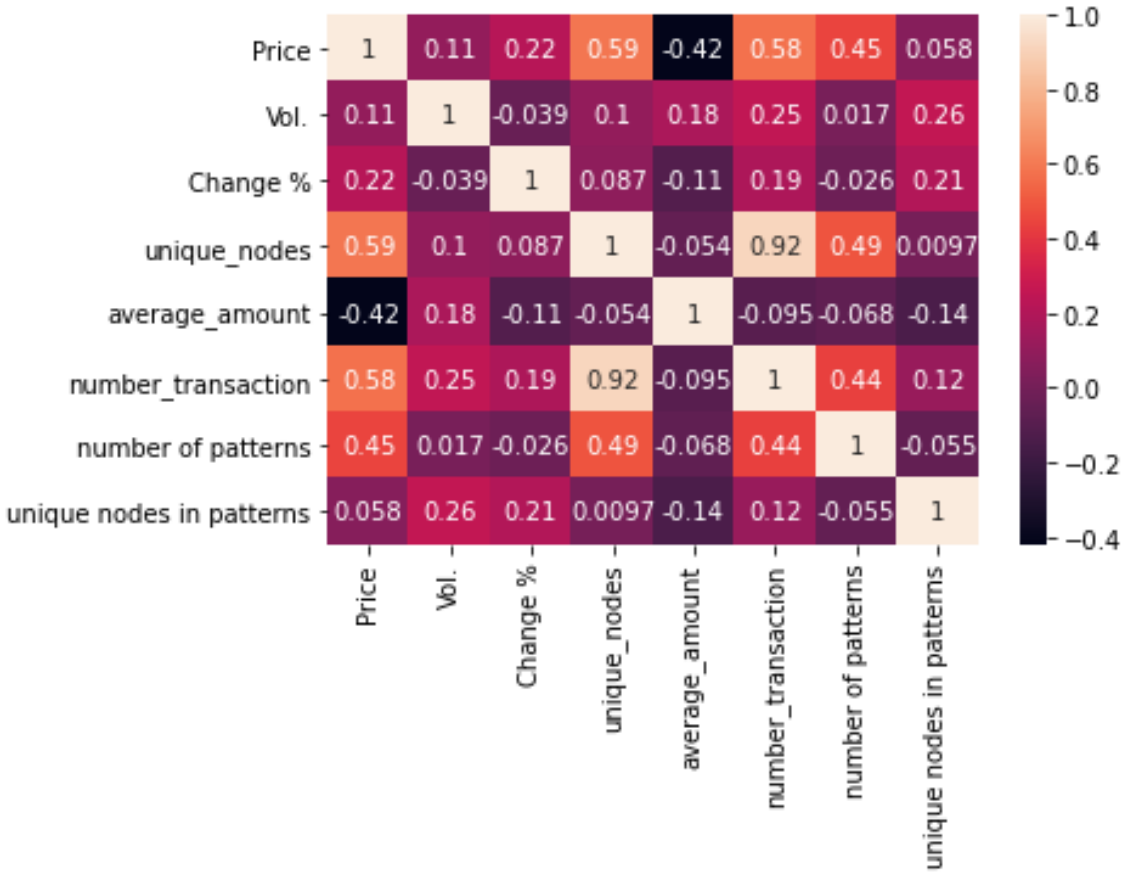


Figure 27: Pearson correlation between different Blockchain metrics and patterns.

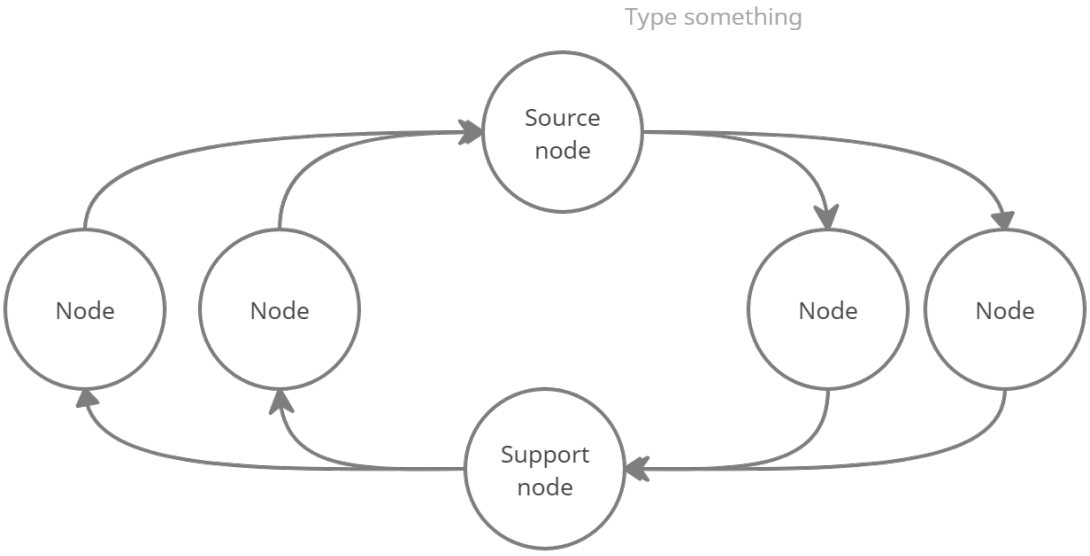


Figure 28: Example of a complex circular pattern having more than 4 unique nodes.

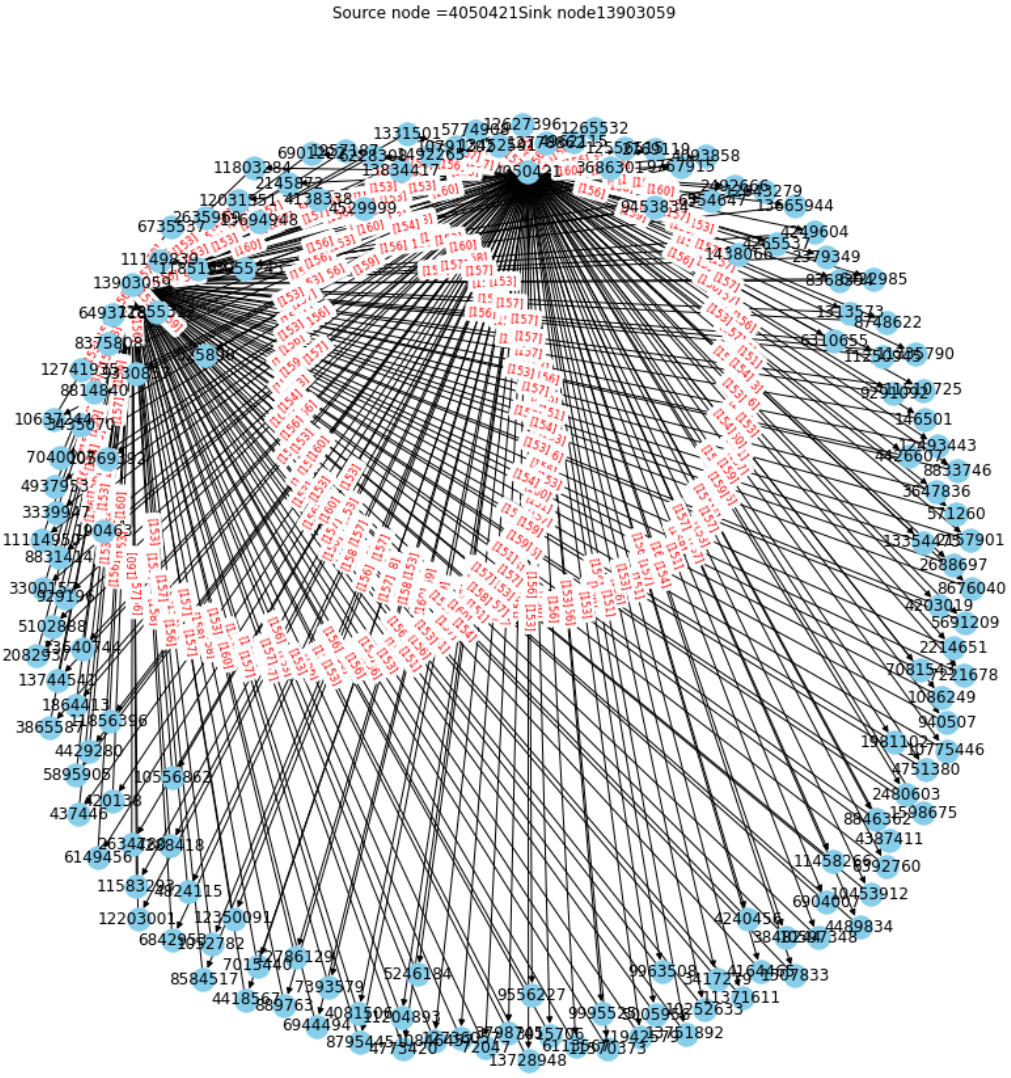


Figure 29: Complex diamond pattern.

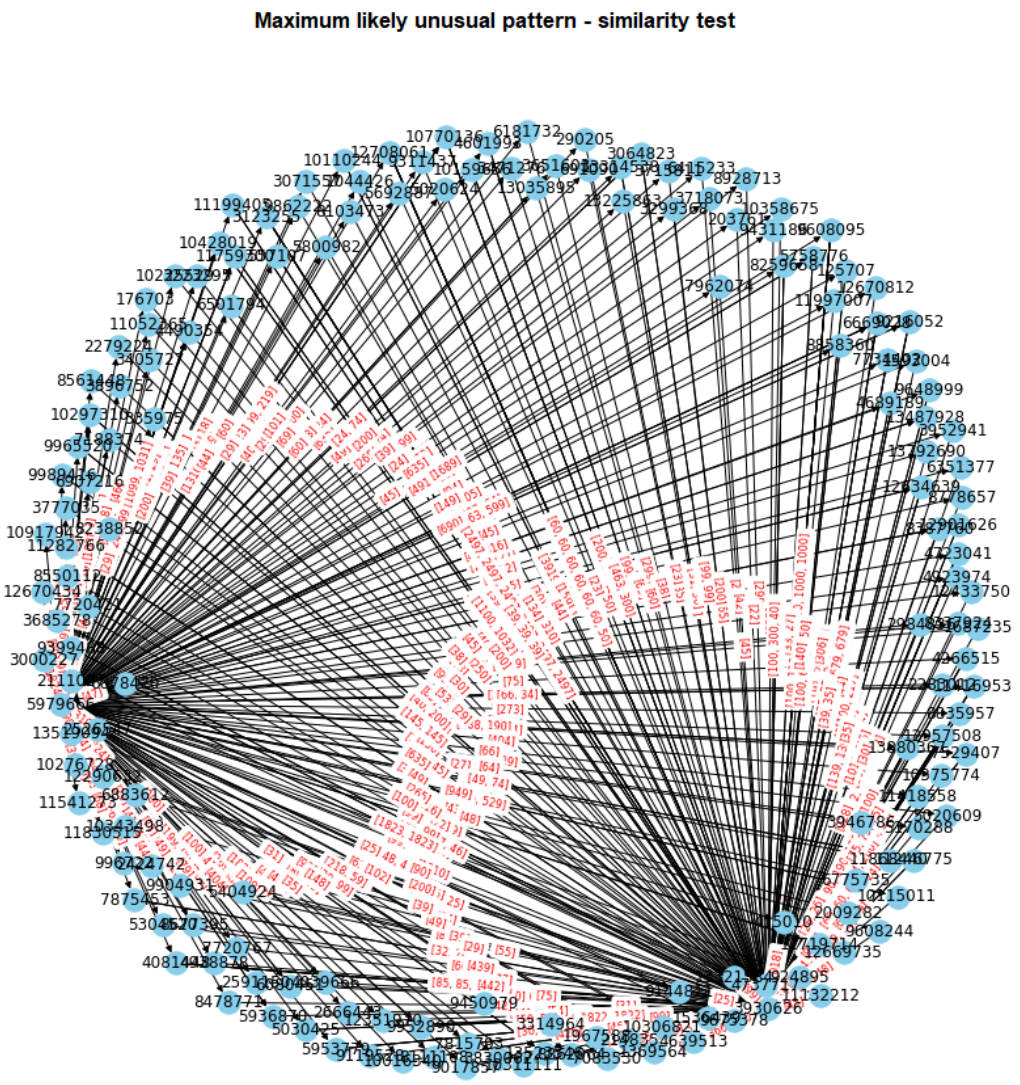
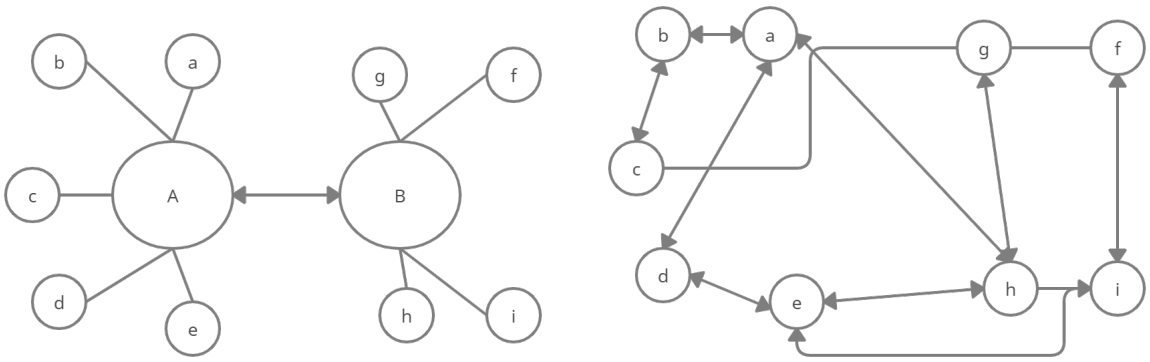


Figure 30: Unsafe pattern according to similarity test



(a) (b)

Figure 31: Figure (a) shows centralised network structure and (b) shows decentralised network structure. Transactions in (a) have to pass through centralised entities A, B. In contrast, in (b) transactions can happen peer to peer. Blockchain is a decentralised network structure, hence probability of transaction patterns is more.