|  |
| --- |
|  |
| Development of a Multi-party Collusion (Circular Trading) detection algorithm |
| Supervisors – Dong Yuan and Uwe Sontowski |
|  |
| **Alexander Ling - 430391570** |
|  |

|  |
| --- |
|  |

Table of Contents

[List of Figures 3](#_Toc484016208)

[Abstract 4](#_Toc484016209)

[Acknowledgments 5](#_Toc484016210)

[1. Introduction 6](#_Toc484016211)

[2. Background and Literature Review 7](#_Toc484016212)

[2.1 Background on the stock market 7](#_Toc484016213)

[2.2 Types of stock market 7](#_Toc484016214)

[2.2.1 Exchanges 7](#_Toc484016215)

[2.2.1 OTC market 8](#_Toc484016216)

[2.2 Background on Market Manipulation 8](#_Toc484016217)

[2.3 Types of Market Manipulation 9](#_Toc484016218)

[2.4 Motivation for market surveillance 10](#_Toc484016219)

[2.5 Difficulty of manipulation detection and detection strategies 11](#_Toc484016220)

[2.6 Collusion 12](#_Toc484016221)

[3 Design and Implementation 13](#_Toc484016222)

[3.1 Defining the problem - Circular Trading 13](#_Toc484016223)

[3.2 Problem formulation 15](#_Toc484016224)

[3.3 Algorithm – Collusion Checking 16](#_Toc484016225)

[3.3.1 Group net volume calculation for > 2 parties 17](#_Toc484016226)

[3.3.2 Other thresholds 20](#_Toc484016227)

[3.3.3 Checking previous collusion alerts 21](#_Toc484016228)

[3.3.4 Check volume domination 21](#_Toc484016229)

[3.4 Algorithm – Group Building 22](#_Toc484016230)

[3.4.1 Idea – Cycle searching 22](#_Toc484016231)

[3.4.2 Efficient group building – Pruning Nodes 23](#_Toc484016232)

[3.4.3 Efficient group building – Pruning Edges 25](#_Toc484016233)

[3.4.4 Group Building design and implementation 27](#_Toc484016234)

[3.4.5 Data Collection 29](#_Toc484016235)

[3.5 Unit Tests 30](#_Toc484016236)

[4 Analysis and Results 30](#_Toc484016237)

[4.1 Time and memory complexity 30](#_Toc484016238)

[4.2 Results 33](#_Toc484016239)

[4.3 Run time against real market data 34](#_Toc484016240)

[5 Conclusion and future work 36](#_Toc484016241)

[6 References 37](#_Toc484016242)

[7 Appendices 38](#_Toc484016243)

# List of Figures

[Figure 1. Example of sell side of the order book 8](#_Toc484014446)

[Figure 2. Example of circular trading 13](#_Toc484014447)

[Figure 3. Example of graph 15](#_Toc484014448)

[Figure 4. Example of 3 party collusion 16](#_Toc484014449)

[Figure 5. Examples of 0 group net volume 17](#_Toc484014450)

[Figure 6. Example used to prove group net volume calculation is acceptable 18](#_Toc484014451)

[Figure 7. Example of unwanted collusion triggers 20](#_Toc484014452)

[Figure 8. Low net value without cycle 22](#_Toc484014453)

[Figure 9. Low ranking participants impact on total trade volume 23](#_Toc484014454)

[Figure 10. Low edge volume cut-off 25](#_Toc484014455)

[Figure 11. Psuedo-code for group building algorithm 26](#_Toc484014456)

[Figure 12. Example used to demonstrate group building 27](#_Toc484014457)

[Figure 13. Profiler results sorted by time spent 31](#_Toc484014458)

[Figure 14. Pseudo-code for depth values 32](#_Toc484014459)

[Figure 15. Run time on real markets 34](#_Toc484014460)

[Appendix A. 2 Party collusion example result 0](#_Toc484014461)

[Appendix B. 2 Party collusion example result graph 0](#_Toc484014462)

[Appendix C. 4 Party collusion example result 0](#_Toc484014463)

[Appendix D. 4 Party collusion example result graph 0](#_Toc484014464)

# Abstract

The purpose of this study is to investigate how to detect a type of market manipulation called circular trading which has seen use in some stock exchanges around the globe. After investigation an algorithm must be designed and implemented to find and alert on cases of this type of market manipulation. There was a previously defined algorithm used for detecting 2-party collusion, however in this paper it is expanded on to support up to N participants. This study was conducted as a NASDAQ research project and has results based off of real market data from various NASDAQ exchanges. The results of the research and implementation have found various cases of plausible but unconfirmed manipulation in all tested exchanges. These tested exchanges include BIST (Borsa Istanbul Exchange) and ASX (Australian Stock Exchange). Although the developed algorithm can be run on any number of parties it is inefficient when run across large amounts of market data with a value of n >= 6. Moreover there are clear memory issues during parts of the algorithm causing crashes for n >= 6. Further research and optimization is suggested to improve the run time and memory usage of the algorithm, however for cases where n <= 5 it is possible to find good results which signify cases of legitimate circular trading.

# Acknowledgments

This thesis became a reality with the help and support of many individuals over the course of the past year.

Firstly I would like to extend my thanks to my family and friends for the constant support and encouragement to continue working and researching even when I couldn’t muster up further effort to continue. They helped greatly simply by always being there for me.

I would also like to express my utmost gratitude towards my supervisor Ian Park and manager Uwe Sontowski from NASDAQ who provided me ideas for the algorithm and constantly provided good feedback. Their knowledge and experience as software engineers and finance gurus helped me enormously when designing the logic of the algorithm. Their feedback of my code also helped largely to prevent many possible issues with my program.

Finally I would like to thank my university supervisor Dong Yuan for his help with the actual writing and structure of my thesis.

# Introduction

Market manipulation is a deliberate attempt to interfere with fair trading in stock markets and it is a crime that is present in markets around the globe. There are many different types of market manipulation and it is an illegal act that may result in unfair results for traders. A trader is any investor or group of investors buying and selling in a market. The market can be manipulated in many ways; one such method would be for a group of large stock holders working together to increase or decrease the price of a certain security, causing unfair prices to other individuals. For example if a cartel decided to buy a large amount of a certain security, thereby decreasing the supply and driving the price upwards. They could then sell what they bought at the higher price while normal traders would be forced to buy at the higher price. Another example would include releasing false information causing uninformed individuals to buy stocks out of being misinformed. Traders will naturally gravitate to the market with the highest integrity in hopes that the effect of market manipulation has been eliminated. As a result all stock markets would definitely strive to have an environment where the likelihood of market manipulation is minimal.

Clearly all kinds of market manipulation are a large problem and all trade activity must be monitored in stock markets to prevent any kind of unfair trading. This leads to the topic of this paper which is to develop a circular trading collusion detection algorithm. Circular trading collusion is one method of market manipulation which involves two or more traders trading large volumes between themselves with a small net change of stock ownership. It may also be done by one person with access to multiple accounts on the exchange. This is done to generate what looks like high activity for the security, which may then generate interest in the security and thus increase the demand and price of said security. Moreover there would be minimal loss for the participants involved as they are simply trading within themselves with no net change in volume. This would generate artificial market activity and falsely create the look of a popular liquid security, attracting other traders. This manipulation may be enacted by both small and large groups of participants which would make it far more difficult to detect as every group of participants must then be analyzed. This leads to the main idea for this topic, which is to create an n-party collusion detection algorithm which can detect collusion between any of the parties in the market.

This will be created as an ‘alert’ which is the name of the detection algorithms that have been built by Nasdaq. The code will be done in a language called ALICE which is an internal language based off of C++ that the alerts are coded in. The alerts look over the market data and compile information based on the events in the market. This compiled information is then examined by the algorithms and suspicious activity will trigger the alert. Note that although alerts trigger on suspicious activity, the traders may not be doing anything illegal; it may just seem like suspicious activity. The alerts run on software native to Nasdaq and this software can also be used to visualize the results of the findings in a graphical format.

# Background and Literature Review

In this section I will examine different sources to firstly give a background of the market and how it can be manipulated. I then discuss why the market should be monitored and look at strategies of market manipulation detection.

## Background on the stock market

The stock market is a market in which the shares (also referred to as stocks) of companies are traded through exchanges or over-the-counter (OTC) markets. Previously stock markets were mainly in physical locations however due to the advancement of technology most trading is now done online as well as in a physical location. The stock market allows companies to register their stocks onto the market and then allow participants to gain equity of these companies through the exchange of money. These participants include both individual investors and institutional investors, and also publicly traded corporations trading in their own shares.

Participants on the market may buy and sell shares of companies to other traders and this trading is mostly done to gain equity in a company to gain profit. If a company is profitable the price of the shares will increase and investors who own these shares will also profit. Hence it is clear that investors are trading on the market with the intention to make profit by buying low and selling high.

Buyers on the market *bid*their desired price for a stock while sellers *ask* for a specific price for a stock. Trades occur when the bid and ask prices match where the stocks are traded on a first-come-first-serve basis if there are multiple traders bidding/asking at the same price. There are mainly two types of orders:

* Market Order – The trader automatically is executed immediately and the stocks are traded at the best available bid/ask price.
* Limit Order – The trader specifies their own price and amount to be traded. These orders may take longer to trade if there are no ask/sell orders at the desired price.

This paper will be referring to limit orders as the algorithm used must analyze price and market orders do not have a set price.

## Types of stock market

### Exchanges

Stock exchanges are places or organizations in which people may trade stocks, derivatives, and other types of securities through a centralized source. Two examples of popular stock exchanges are the NYE and Nasdaq, these two organizations act as the middle-men when two traders want to exchange stocks. All traders that wish to trade through the exchange must follow the exchange rules and usually results in less risk for the traders compared to Over The Counter markets. Moreover market data can be more easily monitored on exchanges which allows for the detection of market manipulation. I will mainly be focusing on the use of my algorithm in an exchange rather than OTC markets as market data is required to find cases of circular trading

### OTC market

An OTC market is a decentralized market without a central physical location and the majority of trading is done by internalization. This means that securities are traded directly between the traders without any mediators like in an exchange. Participants on the market trade through various communication systems such as phone, email, and other electronic trading systems. A simple example of an OTC market is the foreign exchange market. In OTC markets there is “minimal availability of trade executions market data to conduct a proper comparison and identify true outliers” [4] This results in difficulty when monitoring the market for market manipulation techniques compared to looking for market manipulation signs over an exchange. This also concerns my research topic as I will be mainly conducting my data analysis over exchange market data, specifically Nasdaq’s market data.

## Background on Market Manipulation

Market manipulation is an illegal action done by individual or groups of traders in markets to gain an unfair advantage in trading compared to other traders in the market. Market manipulation strategies usually come with a large gain with little or no risk to the trader/cartel (A cartel is a group of traders). All stock markets may be manipulated in different ways and while the issue has decreased on main exchanges, it is still a large issue in over-the-counter (OTC) market in the US, and also in new emerging markets. [1] This is largely due to rules in the exchanges which prevent some types of market manipulation and also new technology which allows the exchanges to detect suspicious activity. There are multiple methods to manipulate the market and some famous market manipulations include comers, and short squeezes. [2] More market manipulation methods exist and may be mixtures of different types of methods which make these strategies far more difficult to detect. These market manipulation tactics often involve “large traders” which are traders whose trades will vastly change the price of a security due to the size of the trade. For example (refer to Figure 1) if a trader puts in a market order for 350 LTC this will match all of the sell orders until the price goes up to $25, causing the price to be manipulated from $22.50 to $25.

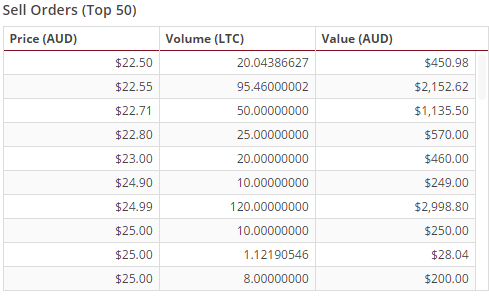


Figure 1. Example of sell side of the order book

Furthermore market manipulation methods are common among parties which have access to insider information as they may use this information unfairly. Although using non-public information to gain an unfair advantage in trading is illegal it is difficult to detect if the traders act with caution. Hence it is clear that these “corporate officers and board members who know of new products or inventions” are suspicious and their actions must be monitored to avoid market manipulation tactics. These particular individuals must “adhere to special regulations when purchasing stock in their companies to avoid penalties. “ [3] These rules are put in place to prevent insiders (individuals with private information) from manipulating the market. This is further emphasized by Jarrow as he states that “potentially informed parties such as corporate insiders, brokers, underwriters, large shareholders, and market makers are likely to be manipulators. “ [2]

## Types of Market Manipulation

There are many different types of market manipulation which can be classified into “classification of stock manipulations into three basic categories: information-based, action-based and trade-based”. [5] The main focus of this paper will be the trade based manipulation which is defined by Allen and Gale to be a “trader attempting to manipulate a stock price simply by buying and then selling (or vice versa), without releasing any false information or taking any other publicly observable action designed to alter the security’s value”. [6] As such popular trade based manipulation techniques often involve large traders [2] as they can move the market and stock prices more easily by executing large trades. This is important to the topic of this paper as one of the criteria for finding a suspect is the volume traded.

To get an idea of why market manipulation is a large issue we will go over a few of the techniques and the results from successfully using these techniques. Below is a discussion of two popular techniques. Further in the document we will be discussing collusion which is the topic of the research.

**Short Squeeze -** First we must define a “short” which is when an individual borrows stocks from a broker to sell. In a short sell the borrowed stock is sold and the individual must later buy the same amount of borrowed stocks to pay the broker back (usually with a small interest). Hence it is clear that investors who are short selling would want the stock price to go down so they can expend less to pay back the broker. A **short squeeze** is a situation where the price of the stock is manipulated to move sharply higher, causing short sellers to panic and buy the stocks while the prices are low to pay back the brokers. This pressure causes the price to be pushed even higher, causing the price of the stock to rise artificially.

**Cornering the market –** Cornering the market is a technique which is more prevalent in smaller exchanges as it requires a trader to gain “sufficient control of a particular stock, commodity, or other asset to allow the price to be manipulated”. [3] By having the greatest market share for a product traders may highly influence the price of the product and may do so without much risk. For example a trader continuously buys a product and pushes up the value of the product (short sellers are driven out of the market via a short squeeze) until it reaches a satisfactory level. After inflating the price to a desired level the manipulator may then begin to sell the shares at the inflated price.

## Motivation for market surveillance

It is evident that due to possible manipulators and the abundance of different market manipulation techniques that surveillance is needed in the markets. Without surveillance many market manipulators will be free to do as they please and likely gain unfair profits. While some argue that it is difficult to distinguish manipulative intent from legitimate intent, Pirrong [7] argues that market power manipulation definitely has an effect on prices and quantities. It is argued that “manipulated prices and quantities can be reliably distinguished… even if fundamental market conditions are unusual”. Pirrong argues that manipulation can be detected and is clearly distinguishable from natural market activity by examining the soybean market [8]. Hence we will assume for this paper that market manipulation can be detected.

If we assume this is true we may look at examples of market manipulation such as the Chinese stock exchange in 2013 which included the Shanghai Stock Exchange and Shenzhen Stock Exchange. It was found that “regulators had litigated 21 manipulation cases by 2003.” Moreover it was also speculated that was” merely a tip of the iceberg, since hundreds (even thousands) of abnormalities of price movements, for no proper reasons regarding fundamentals, could be observed during that period.” [5] Evidently market manipulators may gain the upper hand in the markets at the expense of the fair traders.

Moreover the amount of strange movements created by market manipulators may negatively also influence natural trading behaviors and market liquidity. This further reduces market trading integrity which may be a concern for the organizations that are running the markets and result in fewer customers. This is further emphasized by Cumming, Johan, and Li who state that “trading activity decreases if exchanges fail to adopt surveillance procedures and regulations that assure market integrity” [10].

There are clearly solutions other than looking through the market for manipulators, such as limiting the power of large traders and extra regulations against these large traders. However this is simply deterrence and does not actually solve the problem of market manipulation. The best solution would be to detect market manipulation with a high accuracy as this could detect and remove these traders completely from the market, deterring future manipulators.

## Difficulty of manipulation detection and detection strategies

The detection manipulation on the market is difficult and may not always produce correct results. In some cases, what seems to be market manipulation may instead be a trader acting strangely or outlying behavior from an investor. There does not exist much literature on detection techniques that are applicable in real markets, however Palshikar and Apte use graph clustering algorithms to detect collusion set. [5,8]. Abrantes-Metz and Addanki also developed a model “ to detect manipulation in commodities market and apply the model to the case of Hunt Brother’s manipulation of silver”. [5] Sun et al. also proposes the “analysis of trading networks of stocks to identify fraudulent traders, since they find the trading networks of manipulated stocks exhibit significantly higher degree–strength correlation than that of non-manipulated ones.”[5]

A few other authors also attempt to detect market manipulation however it is reviewed in paper [5] that these algorithms take too many “assumptions to facilitate model formulation” and the assumed behaviors are “far from reality”. It is argued that manipulators will not be as simple as to “pump-dump”. I.e. the manipulators do not just conduct their illegal activities at one time. A smart manipulator will weave their illegal activity with normal market behavior, making the manipulation more difficult to detect. It is clear that the detection of market manipulation is quite difficult and potential manipulators must be examined further before reaching any conclusions. However this is a problem inherent to all market manipulation detection strategies and is exceedingly difficult to solve and this paper will not attempt to solve this problem. This paper will mainly be focused on the market manipulators who don’t attempt to hide their illegal activities with legal activities.

The detection of market manipulation may also be more complicated due to the various types of methods that may be employed by the manipulators, some of which may still be unknown. Each method may also require different methods of detection, for example for detecting short squeezes we must look at large volumes of sell orders while the detection of insider trading may be based off of what information is released to the public and the trades before the information is released.

## Collusion

Collusion in the market is secret or illegal actions taken by individuals or cartels in cooperation to unfairly influence the market. Collusion may take many forms in different markets and the purpose of the colluding parties is usually to gain an upper hand in the market or to disadvantage another party in the market. Collusion is popular when there an oligopoly exists in the market; this is where a small number of companies own a large majority of the market share and thus have high level of influence over the prices.

An example of collusion is price fixing. This is where there are a small number of companies in the market who are offering the same product. Hence it is possible that these companies may work together to all set their prices to a relatively high price, forcing traders on the market to buy at the new higher price (vice versa for setting a low price). The unfairness of this behavior can be emphasized more when the colluding parties attempt to eliminate all other non-colluding companies in the market so that all individuals must buy at their fixed prices. This may be done through strategies such as advertising to limit knowledge of the product. Despite this there is also the possibility of tacit collusion, where no formal agreement is made which will be more difficult to convict.

Although collusion seems like a simple method to gain the upper hand in the market it is usually prevented by strict exchange rules and it is an illegal action with considerable punishment. Additionally the difficulty of successfully colluding is high, especially with many parties involved. It is difficult to get many parties who are committed and willing to cooperate as there is the risk of one party simply defecting and alerting the authorities of the collusion. Despite the difficulty of collusion there are still preventative measures in place such as forcing firms to “maintain a substantial amount of forward sales, the procompetitive effect can dominate the pro-collusion effect making it harder for firms to sustain collusion.”[11]

For this paper when we are attempting to detect collusion we will firstly be looking for smaller groups of collusion within the large traders of a security as the likely hood of large colluding groups is not high.

# Design and Implementation

In this section we will be analyzing the problem of this paper, why it is important, and going through different possible ways to solve the problem. Afterwards some of the code used to implement the design is shown and explained.

## Defining the problem - Circular Trading

The main aspect that my topic will be focusing on is a circular trading collusion detection system. Circular trading in a market can be defined as a form of collusion where buy/sell orders are traded within a group of participants with a low net change in stock ownership. The colluding participants will know the exact number of shares to trade and the exact time to trade them to each other. Participants within a group may utilize prior knowledge from within the group, allowing participants to enter orders which will be instantly covered by orders of the same size on the opposite side of the order book. An example is shown in **Figure 2** where 5 participants simply trade within the group.

The consequence of this type of collusion will be that trading volumes will be increased for the security. Hence other investors are more likely to become interested in the security, which could possibly drive the price upwards or downwards. This is especially true as traders often look for upward/downward trends in the market based off of the trading volume and they could easily for fall this artificial volume increase. To understand this we must first look inside the mind of a trader.

Clearly, a rising market should see rising volume. If the trade volume is increasing there is evidently an interest in the security which will continue to push the price higher as other traders will most likely notice the higher trade volume and seek to buy in the security. Even if there is a spike in the price of a security, if it is accompanied by a low trade volume it is unlikely that traders will buy into the security as there is a lack of interest. Moreover a price spike with low volume is usually not worth investing in as low volume means low liquidity, which means that the security may be difficult to sell later even if it reaches a higher price.

On the other hand, a price spike followed by a large increase in trade volume is a clear indicator that the price will continue to rise. Hence it is clear that by manipulating the volume of a security it could be vastly detrimental to normal traders and could easily benefit manipulators who want to increase the price of securities.

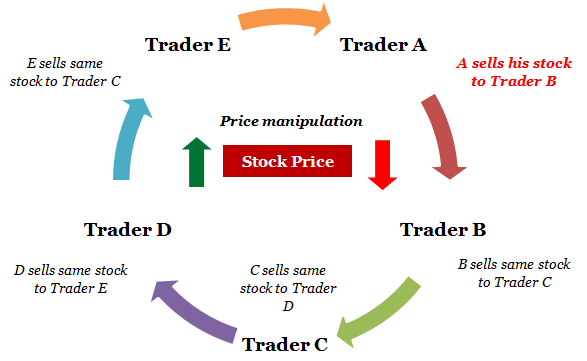


Figure 2. Example of circular trading

This type of strategy has been utilized by many in the past. One of which is Ketan Parek who used circular trading to scam the Indian stock market by inflating the prices of securities such as Zee Telefilms , Global Telesystems, Himachal Futuristic Communication (HFCL) and Silverline. He would “buy into smaller companies through private placement and then rig share prices” using circular trading. [12]

Another scheme involving circular trading was recently discovered in 2016 where Russians were able to funnel money offshore, using circular trading as a form of money laundering. A Russian broker named Igor Volkov consistently placed two orders simultaneously almost every day between 2011 and 2015. In one trade he would use the Russian Rubles to buy a blue chip stock for a Russian company he was representing. Meanwhile in the second trade Volkov would sell the same quantity of the same Russian stock in London. “In the second trade, Volkov—acting on behalf of a different company, which typically was registered in an offshore territory, such as the British Virgin Islands—would sell the same Russian stock, in the same quantity, in London, in exchange for dollars, pounds, or euros.” [13] Since both companies had the same owner the owner was effectively buying and selling to himself.

At first this appears pointless other than the Deutsche Bank earning commission for executing the buy/sell orders. In regards to the finances the clients all finished with roughly the same amount of money they had in the beginning. However, it was later found that these trades were not done with the intention of making a profit, instead these trades were done to take money out of Russia and put it into an offshore company (since both the Russian company and the offshore company belonged to the same owner). As a result the Rubles in Russia were successfully converted into dollars outside of Russia. This tactic allowed a person or cartel to move a large amount of money outside of Russia, successfully laundering the money.

This tactic was done with only one trader and was effective in terms of laundering large amounts of money (each order done by Volkov was usually for around $10 million worth of the stock) Hence it can be deduced that by using a larger group of people it would be possible to more easily employ this tactic without arousing any suspicions as each individual would be trading less money with each order entry.

## Problem formulation

This paper will be describing the problem using a directed graph.

As we want to search for collusion on each security per day clearly there will be a different graph for different securities and different days. If it is possible to solve the problem, this solution can then be applied to all markets, securities, and days.

First we define our directed graph which will contain the trading activity in the market on a certain day:

where is a participant who makes a trade on the day.

where is an edge pointing from participant who trade with each other on the day

When there is a trade an edge is created with endpoints and . If initiated the trade and buys from then the direction of the arrow will be pointing **towards** . An edge pointing to the opposite direction would be created if bought from . If on the current trading day both and buy from each other there will be two edges connecting the two vertices.

All edges will have a value associated with them which indicate the buy/sell volume traded between the two participants on the current day. These volumes can be the sum of multiple trades throughout the day. For example if bought 100 shares from and then later buys another 100 shares then the edge pointing from to would have , while as did not initiate any trades against . An example can be seen in **Figure 3** where Sam bought 100 shares from Anne during the day, while Anne bought 350 shares from Sam.

**Note** that the x in x100 exists as a stock market convention. It is used simply to denote that the number is used to represent a volume.

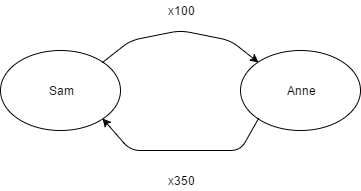


Figure 3. Example of graph

After constructing the graph based off of all the trades on the current day we then must search the graph for cases of collusion. Hence, we must search for **weakly connected components** of the directed graph and then analyze these components to check for collusion. A connected component is weakly connected if the undirected underlying component obtained by replacing all directed edges of the component with undirected edges is a connected component.

To analyze the weakly connected components for collusion we must check for a low net change in stock ownership in comparison to the total amount traded. Net traded volume can be defined as buy –sell volume. Using this formula with the data in **Figure 3** we get:

We take the net volume of the group as

Comparing this to the total amount traded we find that the net traded volume is of the total traded volume. Although we have no defined a set threshold percentage it is clear that the two participants in **Figure 3** are not colluding. Set thresholds will be discussed further in the algorithm section.

## Algorithm – Collusion Checking

In this section we will discuss the algorithm used and how it was designed to search for collusion for more than 2 participants. As defined in section 3.2 it is quite simple to search for collusion between two participants however this becomes more complicated when there are more participants. Three core conditions are stated which will be the main method of detecting collusion. If any group of participants satisfies the three conditions then a collusion alert will be triggered on the participants.

We will also discuss how to reduce the run time and how to remove noise from the market data so the results are more likely to be real cases of manipulation instead of it being normal trading activity.

## Group net volume calculation for > 2 parties

The calculation of the group net volume (which will also be referred to as **group net position**) becomes more complicated when there are more than two parties. The method used to calculate net volume in section 3.2 (buy - sell) would not be effective in the case of 3 or more parties. If we are to use the same method with the participants in **Figure 4** then we would have:

In this case there is confusion on what should be the groupnet volume. In the 2 party collusion example it would be acceptable to take the absolute value of one of the participant’s net volumes however in the example it can be seen that the net volumes can all differ from each other. As a result we must use a different metric for the net volume of the group of participants. On the other hand the net volume calculation for an individual participant may remain the same. From an individual’s point of view it is clear that their net volume should always be their total buy – sell volume.

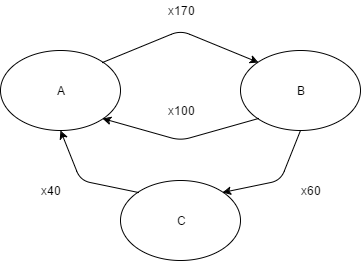


Figure 4. Example of 3 party collusion

A simple initial solution would be to sum of the net volumes of each participant to get the group net volume.

However, in the above scenario this would give us a group net volume of 0 which is incorrect as the value isn’t perfectly being passed around the group. A group should only have a net volume of zero when all volume is perfectly passed around the group without any volume left over.

Some clear cases of groups with 0 net group volumes can be seen in **Figure 5.** In the second graph it is clear that all volume is being passed around without any volume left behind as the traders are simply trading 100 shares around in a circle. Similarly in the first graph in **Figure 5** it can be observed that all participants are simply buying X amount from another participant and selling X amount back, and hence there is no net change in volume. As a result the sum of the net volumes of each participant cannot be used as the group net volume.

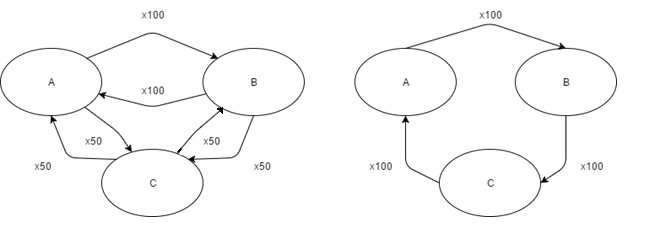


Figure 5. Examples of 0 group net volume

Another viable solution is to use sum of the absolute values of net volumes of each participant to get the group net volume.

Using this formula the scenario in **Figure 4** would not give a group net volume of 0 and instead result in a net volume of 60. Moreover the scenarios in **Figure 5** would still return a net group volume of 0 which is correct.

Consequently when looking at the case in **Figure 3** the group net volume would be 500 which seems far too high as this even exceeds the group total volume. However in this paper we will still use this formula to detect net volume. This is because we must note that the net volume calculation is only being used to compare against the total volume traded within the group. When attempting to detect collusion the group net volume must be less than a certain percentage of the total group volume. This percentage threshold will clearly be a low amount such as 10% or 20% such that only the most extreme cases of collusion are detected.

Assuming that 20% is used as the group net volume threshold, we will look at **Figure 6** to prove that this group net volume calculation is acceptable for use as long as it is being used to compare against total group volume.

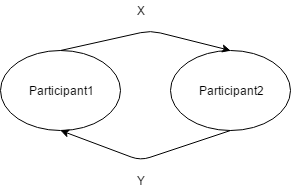


Figure 6. Example used to prove group net volume calculation is acceptable

Ignore x = 0 while y > 0 and vice versa as there is no collusion in these cases as there is only trading activity going one way. The case where x = 0 and y = 0 can also be ignored as if there is no trading activity in the group there can be no collusion.

The two participants must trade with each other for collusion to exist. Hence it can be observed that x and y **must** be relatively close in value before even passing the 20% threshold which would satisfy the graph for collusion. In the scenario in **Figure 3** there will be the threshold:

Even with a relatively high percentage of 40% the threshold would still be which is still not close to the value of 350 in the example.

As a result we may completely ignore these types of cases and continue to use the same function to calculate group net volume as it is only being compared to group total volume. Were the group net volume to be used elsewhere this may not be the case. The condition is below with x being the desirable percentage. A higher percentage will yield more alerts while lower percentage will give less alerts.

## Other thresholds

To detect worthwhile cases of collusion in markets other metrics will be needed to be checked within the group of participants. Detecting collusion cannot just rely on checking the group net volume. One case is the total group volume; it must also be checked against the total volume traded for the security. This is because a scenario where there is low group net volume with a low group total volume would not be deemed as important and should be ignored as noise.

For example if a security has many participants trading millions of shares each day then we should not be alerting when there is a group with 3 participants trading only 100 shares during the day. Clerarly there is no reason to suspect this group of participants for collusion as the amount they are trading is not enough to influence the market for that security. The algorithm being created is only concerned with participants with the trading power to manipulate the market. Hence there must be a total group volume threshold as well as a group net volume threshold.

is the sum of all edges pointing out of

is the sum of all edges pointing towards

For example using **Figure 4** this formula is used to get

The total group volume must then be compared to a percentage of the total volume traded in the security for the day. Hence to detect collusion another check must be included, the condition is shown below. Clearly a higher x value will give fewer alerts while a lower value will give more alerts.

Another threshold that must also be considered is the amount of trades between the participants. From what has been defined previously a group of 3 participants can be seen as colluding when they pass the group net position and total group volume thresholds. However, sometimes this may not be enough evidence to suggest real manipulation if it happens only once. If two large traders simply decide to buy and sell a large amount of a stocks to each other at one point during the day it could possibly be coincidence. On the other hand if they do this kind of trade behavior consistently then it becomes a lot more suspicious.

As a result there must be another threshold to check the amount of trades between the participants in the group. This check can be done after the participants in the group pass through the first two conditions defined before. The number of trades between participants will be recorded during the day and then checked at the end of the day to see if it passes the threshold.

Where X can be a value defined by the exchange using the algorithm. This value should not be hard coded into the algorithm and should be configurable, similarly to the percentage thresholds used in the two conditions before. If the exchange is interested in seeing all cases of possible collusion then the value could be set to 0. If exchanges want to see more realistic cases of collusion this value can be set to something higher, however this will clearly reduce the number of alerts that are triggered.

## Checking previous collusion alerts

The algorithm must find all cases of collusion for groups of 2 participants up to n participants. Hence this algorithm must start searching for collusion over 2 participants. After checking for collusion on all groups of 2 participants the algorithm may begin searching for collusion on groups of 3 participants. This is done so previous cases of collusion do not trigger future cases of collusion. This is explained in the left graph of **Figure 7** where it is clear that participants A and B should trigger a collusion alert. However the group A, B, C would also trigger an alert as the net volume does not change much with the addition of C trading a small amount of x10. Thus the algorithm must first check if group A, B, C contains a subgroup (such as {A, B}, {B, C}, {A, C}) of participants in which a collusion alert was previously triggered. Hence when the algorithm is analyzing groups it should ignore all groups with subgroups of participants that previously triggered the alert.

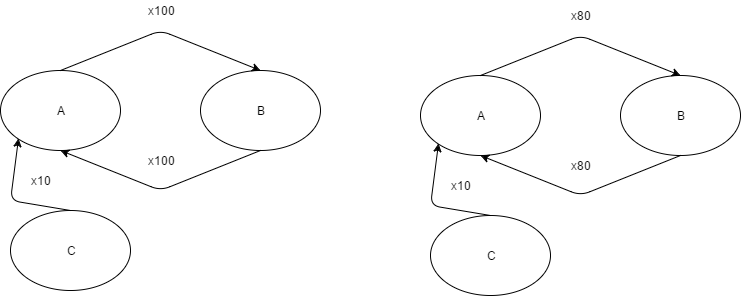


Figure 7. Example of unwanted collusion triggers

## Check volume domination

Another case where the 3 conditions would incorrectly identify a case of collusion can be seen in the right graph in **Figure 7.** In this scenario it can be seen that there is most likely a case of collusion between A and B as the net group volume is 0. However if we assume that the threshold for condition 2 is x165 then A and B will not trigger an alert as the x160 volume traded between A and B is less than the threshold of x165. On the other hand, the total volume of the group A, B, C is x170 which will pass the threshold of x165. Moreover the group net volume will not change much from the addition of participant C. As a result this will trigger a collusion alert when it should have been triggering from the group A, B.

In the algorithm a precaution must be taken against these kinds of scenarios by checking that the volume traded between a pair of participants does not dominate any of the other trading pairs within the group. This is done by looping through all existing pairs of participants in the group and then checking if the trade volume in each pair is greater than 2 times the sum of the trade volumes of all other pairs. In the example we would check the following conditions :

Since the pair AB dominates every other existing trading pair in the group then the algorithm should not trigger a collusion alert as one pair is dominating every other pair in trade volume.

## Algorithm – Group Building

After defining the three conditions used to detect collusion within a group of participants, the method to build these groups of participants from the market data must be designed. From what has been defined previously it is clear that building groups using a graph is a good idea as there will be lots of information to store and it can be easily input into a graph.

## Idea – Cycle searching

There can be many possible solutions to this problem using a graphical approach. Searching for cycles is a simple first approach that may come to mind as the money literally has to go around the group so that the net volume stays low. It can be seen in both **Figure 2** and **Figure 3** that a cycle seems to be needed for there to be collusion. If there was only one participant selling to the other then the group net position would not be a low value. Hence to detect 2-party collusion we may simply search for cycles between two parties. However when searching for collusion on 3 or more participants the problem will become more complicated than just searching for cycles in the graph. This is because the algorithm will strictly search for any group that satisfies the 3 conditions outlined in the previous section. Refer to **Figure 8** below and assume that condition 2 and 3 pass in this case. The group has a total group net value of 60 which must then be compared against the total group volume. In this case use a low 20% as the threshold

.

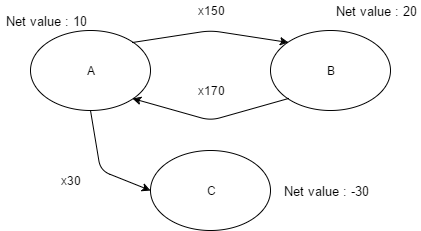


Figure 8. Low net value without cycle

Hence in this case all 3 conditions to satisfy collusion pass even though there is no 3 participant cycle in the group. As a result the algorithm used cannot simply search for cycles in the graph of trading activity. Unfortunately, if this is the case then there are no obvious patterns such as cycles to look for in the trading graph.

## Efficient group building – Pruning Nodes

It appears that the algorithm must search through all available market data and check every weakly connected component in the graph for collusion. The actual group building algorithm will be explained in a later section. It will involve using a graph traversal algorithm that is a combination of the depth first search and breadth first search algorithms. These two algorithms have running times of O(V + E) where V is the number of nodes while E is the number of edges in the graph. As a result it would be beneficial to first reduce the number of nodes and edges in the graph before building groups.

Assuming that the algorithm simply traverses over the whole graph, this would not yield an acceptable runtime on a complete graph with over 1000 participants (real markets frequently do have more than 1000 participants per security per trading day). A complete graph is a graph which has an edge connecting each node to every other node. If every node is connected to every other node then the total number of possible 2 participant groups would be . Meanwhile if searching for groups of three participants then the result would be . This would scale exponentially worse when searching for groups of more than 3 participants. Furthermore, to make matters worse some markets would have multiple securities with over one thousand participants trading on a given day. Given that all securities must be analyzed per day it would be an extremely inefficient idea to find every group of 1, 2, 3 … n participants for each security.

One approach to make the algorithm more efficient is to reduce the amount of data that the algorithm must go through. For example, by looking at the three conditions defined before it can be seen that only groups with high total trade volume will pass the 2nd condition. Hence it would be possible to exclude certain participants who do not trade a high amount of volume on the day.

Scripts were run on the ASX market to gather market data on the trade volume of all participants per security during the day. The total trade volume of each participant was stored and sorted on the total trade volume in descending order. This resulted in multiple lists of top trade volumes for the top trade participants per security each day. It was found that most of the top 25 participants in each security will be large traders and will often simply trade with each other. Moreover the participants trading during the day also seem to trade with these top participants as they usually open the best bids/asks with large volumes. However to be safe and to not miss too many cases of collusion the algorithm should analyze all trading between the top 30 participants during the day.

To show why this number was chosen we must refer to the results from the script. The script stored the for each security as a percentage. This would find out the participants’ total trade volume compared to the total security trade volume. These percentages were then all put into a normal distribution to analyze how the participants at a certain rank would compare to the total security trade volume. In **Figure 9** the top 30, 35, 40, and 50 ranked participants are analyzed by their total trade volume compared to the total security volume. It can be seen that over all securities the participant at rank 30 had a maximum total trade volume of 1.42% of the total security volume on the day. Similarly it can be seen that for 90% of the securities the top 30th participant only traded at a maximum of 0.93% of the total security volume. Moreover, the average participant ranked 30th in trade volume only traded 0.64% on average.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | Percentile | | | | | |  |  |  |
| Rank | 1 | 10 | 25 | 50 | 75 | 90 |  | MIN | MAX |
| Top30 | 0.0029 | 0.19 | 0.4 | 0.64 | 0.81 | 0.93 |  | 0.00002 | 1.428667 |
| Top35 | 0.0058 | 0.15 | 0.33 | 0.53 | 0.65 | 0.76 |  | 0.00006 | 1.14197 |
| Top40 | 0.0041 | 0.11 | 0.28 | 0.43 | 0.54 | 0.64 |  | 0.00001 | 1.071501 |
| Top50 | 0.0022 | 0.078 | 0.19 | 0.33 | 0.41 | 0.49 |  | 0.000002 | 0.714334 |

Figure 9. Low ranking participants trade volume compared to total security trade volume

These percentages are very small and should be ignored when considering collusion as the algorithm is searching for groups of participants with a total group traded volume over a certain percentage of the total security traded volume. Although the maximum values of over 1% may seem high, each participant usually trades with multiple other participants. As a result, the trade volume for that participant may not be solely concentrated within the group being analyzed for collusion. Hence in most cases the participant will be trading around 0.5% or less within a group being analyzed for collusion. This would clearly not be enough to pass condition 2 if the threshold is set to anything higher than 10%.

On the other hand if multiple participants who are trading within the 1% range are attempting to conduct circular trading then they will not be discovered. I.e. If 3 participants solely trade their 1% within a group of 3 participants who also do the same, they would not be discovered using the algorithm. However this would also mean that only 3% of the total security trade volume is being passed around the group, which is quite a negligible amount. If it is desired to detect cases of circular trading with these small amounts of volume then the algorithm can be tweaked to look at more of the top participants.

As a result the algorithm may generally ignore all participants that are not part of top 30 participants by trade volume. This will significantly lower the running times as there would be far fewer groups to analyze compared to analyzing all weakly connected components in the graph. In comparison to the groups from before, using 50 participants would lower the amount of groups to . Similarly with n = 3 the result would be a total of compared to the possible groups from before. This is a 400 times decrease in the number of groups for 2 participant groups and an 8477 times decrease in groups for 3 parties. It can be seen that this difference in number of groups would only get higher for higher n. Hence is clear that there would be a significant runtime boost when limiting the graph to only contain nodes and edges coming from the top X participants. As a result the graph will be pruned so that all nodes and edges that are not from the top X participants are ignored.

## Efficient group building – Pruning Edges

Another method can be used to further reduce the amount of edges that the algorithm must search through. As the total run time will be dependent on both the number of nodes and the number of edges it will be beneficial to also prune edges from the graph. Using the same approach it is also possible to limit the edges based on trade volume. The method used will prune the graph so that only edges with X volume will be included in the group building algorithm.

The reasoning behind this choice is to pick a minimum volume X that must be satisfied so that the edge is not pruned from the graph. Unlike the case with pruning participant nodes, the numbers of edges vary greatly with each security. The above formula is used to find the minimum volume needed for an edge based on the total security volume. This is needed as for most securities there are a large number of participants who do not trade a large volume and should be excluded from the groups due to their low volume trades. The top X approach used for pruning nodes is not used for edges as within a trading day there will often be many edges with relatively low volume and only a few edges with a large volume. For example if there was a graph with edge values:

If the top 30 edges are taken then the algorithm will attempt to analyze many low volume edges which will not contribute to collusion due to their low volume. However if the formula above is used then only edges with a trade volume equal to or greater than will be taken into account.

Note that these edges may not be filtered out by the node pruning in the previous section as large traders may also trade small amounts with each other during the day. This would cause these edges to remain in the graph after the node pruning.

An example of the formula is shown in **Figure 10**. The edges will be sorted in descending order based on their trade volume and the line is the X value, which is the minimum volume of .

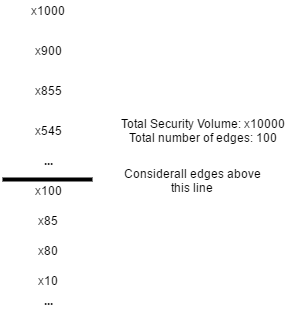
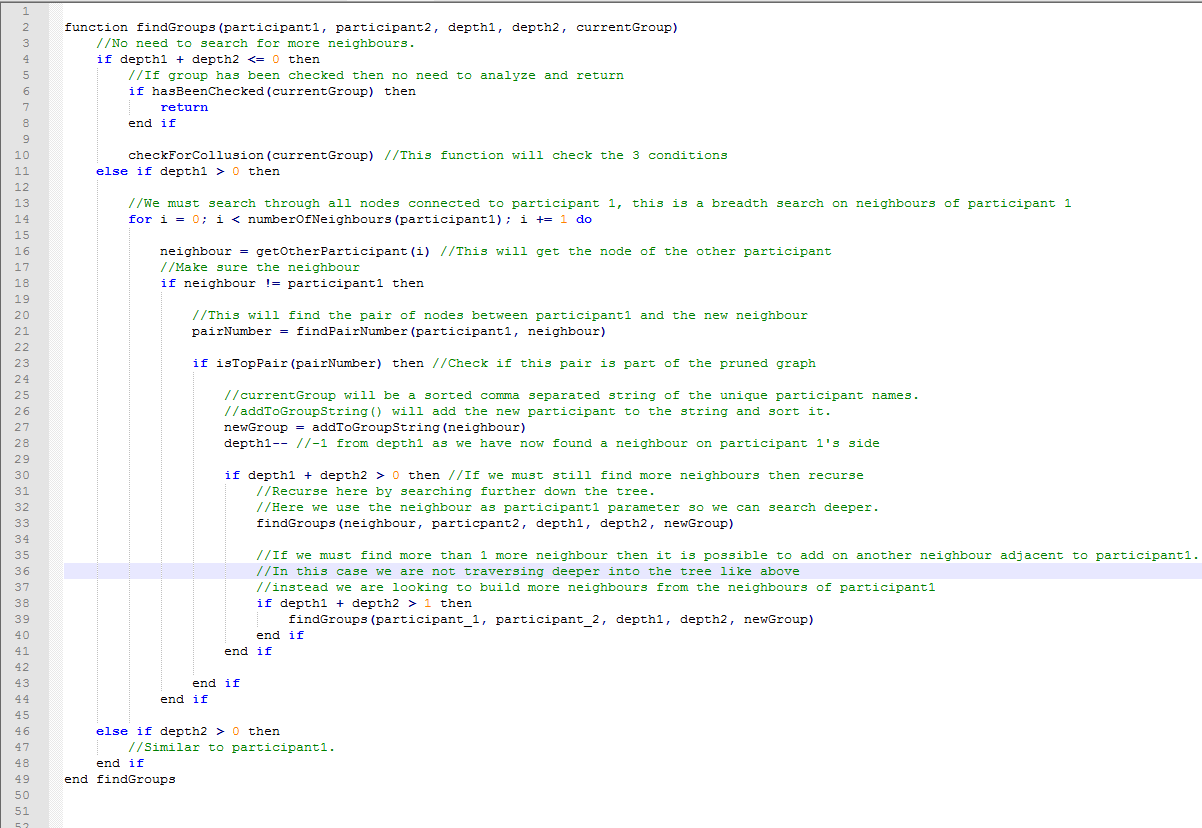


Figure 10. Low edge volume cut-off

There is a problem where the formula will give too low or too high of a value for the cut-off so there is a forced threshold that must be used. Looking at the formula it can be seen that too many edges will result in a low of a cut-off number, causing no edges to be pruned. Too few edges will result in too high of a cut-off number, causing some important edges to not be included. As a result it would be beneficial to enforce a strict upper and lower bound for the decimal given by . In the actual implementation the upper and lower bounds for this value will be to ensure all important edges are used.

## Group Building design and implementation

The actual algorithm used to search for groups will be quite simple as it is similar to a depth/breadth first search. The function to find groups will be recursive and will start from a pair of nodes to branch off and find other nodes connected to the pair of nodes to add to the group. These pairs of nodes are built after analyzing all the market data for the day and each pair of nodes is simply any two nodes connected to each other by an edge in the graph. To build groups over the whole graph this function must be run on every pair in the graph.

Figure 11. Psuedo-code for group building algorithm

The pseudo code for the group building algorithm is shown in **Figure 11.** The parameters of the function are the two participants to start searching from - *participant1* and *participant2.* The *depth1* and *depth2* parameters state how far to search from each of these participants. For example if depth1 has a value of 1 then the algorithm will attempt to find a neighbor node from participant1 and vice versa for depth2 and participant 2.The *currentGroup* parameter is a sorted comma separated string of the participant names in the group. As the algorithm uses recursion the base case is when depth1 and depth2 are both 0. This means that there is no need to search for more participants as the group has been built.

Using **Figure 12** as an example, if *findGroups(Participant1, Participant2, 1, 2,”Participant1,Participant2”)* was run then the algorithm would first start searching for neighbors from participant1 as depth1 > 0. It would then loop through all neighbors of participant1 and in this case it would take A as it is the first seen participant. There would then be a recursion and the function would then run

*findGroups(A, Participant2, 0, 2,”A,Participant1,Participant2”).* Since depth1 is now 0 the algorithm will look for more neighbors from Participant2’s side. In this case it would see D first and then run *findGroups(A, D, 0, 1,”A,D,Participant1,Participant2”)*. Now the search will start again from participant D as depth2 > 0. The search will then find G, resulting in *findGroups(A, G, 0, 0,”A,D,G,Participant1,Participant2”)*. Since both depths are now 0 then the algorithm would check group “A, D, G, Participant1, Participant2” for collusion using the 3 conditions.

Note that when *findGroups(Participant1, Participant2, 1, 2,”Participant1,Participant2”)* is run then the algorithm will loop through all neighbors of Participant1first. Therefore there will initially be 3 different groups built as there are 3 different neighbors of Participant1. The three lines of code below will be run as a result (This is the breadth first search component of the algorithm).

*findGroups(A, Participant2, 0, 2,”A,Participant1,Participant2”)*

*findGroups(B, Participant2, 0, 2,”B,Participant1,Participant2”)*

*findGroups(C, Participant2, 0, 2,”C,Participant1,Participant2”)*

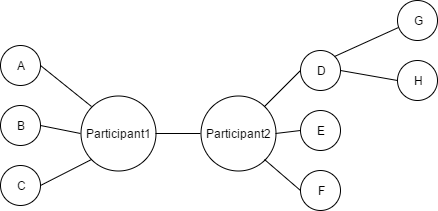


Figure 12. Example used to demonstrate group building

The depth first search component of the algorithm comes into play when the *findGroups* function is called again in line 33 as it changes the parameters so that the search starts from the neighbor. This can be seen in the example before as the search from Participant2 first reached D and then went down to find G.

On the other hand on line 39 there is another breadth first search here. This is needed due to the case where (refer to **Figure 12**)the current group is “A, Participant1, Participant2”. Without this line of code there would be no case where B is added to the group after running *findGroups(A, Participant2, 1, 0,”A,Participant1,Participant2”)*. This is because the algorithm will attempt to look for any nodes after A and clearly there are no neighbors with A other than Participant1 which is already in the group. Hence we must also continue searching from Participant1 to potentially add B and C to the group.

The participant string is sorted so that the same groups with different orders of names are not analyzed twice. For example a group “A,B,C” does not need to be analyzed if “A,C,B” was previously analyzed for collusion. Hence it would be more efficient to first sort the groups.

It must also be noted that the for loop (line 14 in **Figure 11**) loops through all neighbors of Participant1 and we would not want to include any participants which are already in the group. For example if we are looking at a group “A, Participant1” and then want to search for more neighbors from participant1 then A and Participant1 must not be put into the group string again. The *addToGroupString* (line 27 in **Figure 11**)function will take care of this problem by checking if the new participant (given as a parameter) is already in the group string.

It can be seen that this group building algorithm would rely heavily on the number of edges and nodes in the graph, which is why the graph must be pruned in the earlier stages using the methods described before.

## Data Collection

For the design of the algorithm to work correctly there must be code to collect the data during the day. This will be done by using the *on trade* clause in the ALICE language. Any code within the clause will execute on every trade during the day. By doing this it is possible to save all of the market data needed. The saved data will save all data into the pairs defined before. Hence every trade between two participants will have the trade volume added to the pair. The number of trades for each pair must also be counted to check for condition 3of the collusion checking algorithm. Each pair will also be assigned a pair number to easily loop through all edges in the graph. Moreover we must also track the traded volume for an individual participant as the participants will need to be sorted based on individual trade volume (must sort by individual trade volume instead of pair volume) to find the top X participants.

Pairs will be placed in a hash map with keys *security, buyer, seller* where buyer and seller are the buyer/seller participant names. With these 3 keys each trading pair can be uniquely identified every day. If there is a need to process over several days then a date key may be added to the hash map to identify pair data on a given day. By using the pair number all of the other data can be easily found as the data is placed into hash maps with the pair number as one of the keys.

## Unit Tests

Some basic unit tests were created to ensure that the core functionality of the algorithm is correct. Moreover if any further changes are made then it is possible to run the unit tests over the changed code to check for any simple errors. The unit tests generate sample market data and run the alert over the data. The output is then taken and compared to some expected output. If the output matches the expected output then the unit test will pass. Some but not all of the basic unit tests created are described below:

1. ALERT TRIGGER - Basic case of brokers B1, B2, B3 trading with each other with no net change in volume. At the end of the day these 3 securities have made up 100% of the traded volume in the security, but have a 0 net change in security ownership.
2. ALERT TRIGGER – There are a few unit tests created that are the same as the above scenario with the only difference being the participant type. (Broker, Trader, Client) The alert will give different output for different participant types.
3. NO ALERT TRIGGER – Similar scenario to the first scenario except that the total group volume is only 3% of the total volume traded within the security. Thus condition 2 will not pass and no alert will trigger.
4. NO ALERT TRIGGER – Similar to scenario 3 except that now the group net volume does not pass the threshold required.
5. NO ALERT TRIGGER – Similar to scenario 3 except that the number of trades within the participant group does not pass condition 3 of the algorithm.
6. ALERT TRIGGER – A more complicated case with multiple participants. N is set to 5 and the generated market data contains four different groups of participants which trigger the alert for n = 2,3,4,5.
7. ALERT TRIGGER – Certain markets may contain negative prices and hence a unit test similar to the first scenario except with negative prices is required.
8. NO ALERT TRIGGER – A scenario similar to scenario 6 except that one the 2 person group which triggered the alert is now a subset of the 3,4,5 participant groups. Hence the 3, 4, and 5 participant groups will not trigger the alert as there was a previous collusion alert triggered on a subset of its participants.

# Analysis and Results

## Time and memory complexity

To analyze the time complexity of the algorithm we must analyze the part of the algorithm which will be using the most time. The two main parts of the algorithm as discussed in the previous section are:

* Checking participant group for collusion
* Group building

Out of these two parts of the algorithm it can be seen that the group building will take a longer time as the graph must be iterated over multiple times to build all the groups needed. Meanwhile checking each participant group for collusion only requires the participant groups and iterating over the graph is not needed. This can be further proven when looking at the profiler results in **Figure 13**, these results were found by running the alert on a day on the BIST exchange. Running on other days also gave similar results. The columns are

* Count – Number of times the line of code is run
* Time Spent (seconds) – Time taken to complete running the line of code in seconds
* % Time Spent – The time spent running the line of code compared to total run time of the program
* Code – The code that was timed

The *findGroups()* function is the recursive function responsible for searching for all groups. It can be seen that the *findGroups()* function (rows 879 - 883) is run in 4 different parts of the code and the **% time spent** is summed up to around 32%.

On the other hand the *checkCollusion()* function which is responsible for analyzing a participant group for collusion only takes up 3% of the total time (row 878 in the figure). It can also be seen that *checkCollusion()* is run only 579753 times compared to the 6075702 times *findGroups()*  is run. Thus it is clear that finding the groups will take far more time than actually checking a group for collusion.

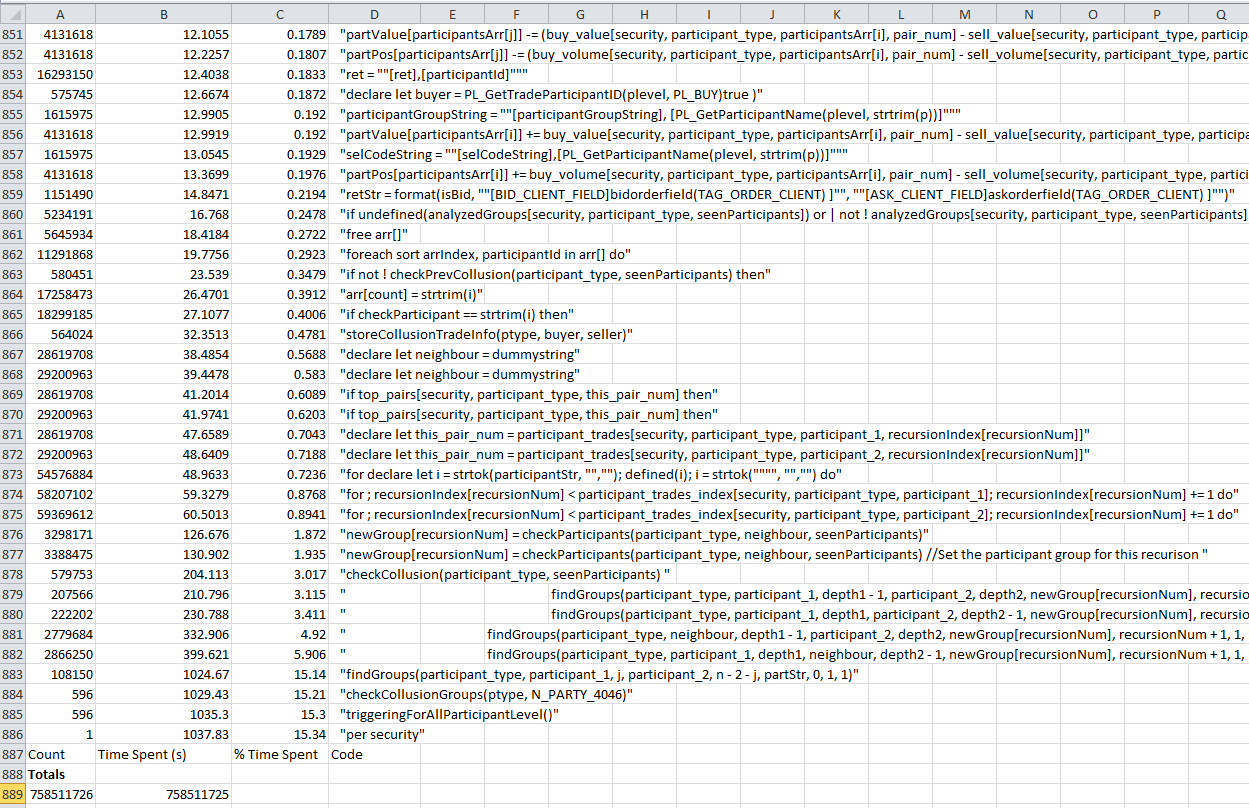


Figure 13. Profiler results sorted by time spent

A worst case time complexity analysis should also be done in terms of the graph. For a worst case analysis, assume that group building is done on a complete graph where each node is connected to all other nodes.

The first part of group building is to iterate over all pairs in the graph to start building the groups off of the pairs. However the graph will have a set maximum amount of nodes as the algorithm only takes the top X nodes. Hence the number of pairs will be a constant which is where X is a custom parameter set by the user to choose how many of the top X participants to analyze (Section 3.4.2).

After finding a pair of nodes to start from the depth values must be set so that the program knows which side to start searching for neighbors. For example refer to **Figure** 12; if a 4 participant group is needed then we may start searching for 2 participants on the side of participant 1 (only A, B and C can be added to the group as they are on the side of participant1). Then search for 1 neighbor from participant1 and 1 neighbor from participant2. Afterwards search for 2 participants from the side of participant2 (D, E, F, G and H are on the side of participant2).

Hence for each pair there is a loop to go through all possible depth values which is shown in the pseudocode in **Figure 14**.

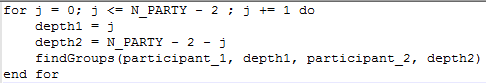


Figure 14. Pseudo-code for depth values

Hence for each pair of nodes, *findGroups()* will run times.

For *findGroups()* to add a node to the group it will loop through all available neighbors of each participant. In the pruned graph there will only be other where X is from the top X participants. Hence to find one neighbor it would take iterations. After finding a neighbor the algorithm will attempt to find another neighbor which is connected to the found neighbor, which will also take as the algorithm will again loop over all other top X participants (since it is a connected graph). In total this will take iterations to find 2 neighbors.

To build a group of N participants, N – 2 more participants need to be found. This will take iterations. However from above *findGroups()* will run n – 3 times for each pair. Furthermore there are pairs per graph. Hence the total run time is:

To get this final run time the constants must be ignored as they are negligible when the program is scaled to a high n.

## Results

This section will mainly be used to show some example results and to interpret them. The results were found by mainly testing on market data from the Borsa Istanbul and Australian stock exchanges. The results included in this paper were from the BIST exchange and can be seen in the **Appendices** section of the paper. There are multiple participant types such as brokers, traders, and clients. In most markets there will be a high number of clients and a lower amount of brokers and traders. However the results shown are from analyzing broker trade data for a more clear visualization of the graph as client graphs mostly have a high number of nodes and are also highly connected. The alerts are run and displayed on NASDAQ software. The top half of the figures display all alerts for the day while the bottom half of the figures display a graph of trading done during the day for a certain security.

When looking at the graph it can be seen that the highlighted circles are the participants who triggered the collusion alert. Moreover the size of the circles is proportional to the amount of trading the participant did during the day compared to the other participants.

**Appendix A and B** contain some examples of two party collusion alerts on the BIST market. The top part of Appendix A shows the alert text for the selected alert. Each alert will have a uniquely configured alert text based on the market information, such as the group total volume and group net volume. In the case of Appendix A it is shown that collusion is detected between brokers DSI and ACP on the security DXDTK.V. The alert text on the top of the image displays all relevant information to the alert, describing why it triggered. The custom parameters used were 15% and 65% for conditions 1 and 2 respectively; this information is all shown in the alert text. Other information which may be important such as an individual participant’s net position is also shown. In this case both participants bought and sold an equal amount to each other during the day and as a result they both ended up with a net position of x0.

In **Appendix B** it can be seen that during the day for the security DXDTK.V the two main trading participants were ACP and DSI (based off of the circle sizes). On the right side of Appendix B it is also possible to analyze the volume traded to every other participant during the day. However for this security it is clear that most of the volume was traded in between ACP and DSI. Other participants would not trigger the alert as their trade volume would be too low to pass condition 2. Similar examples for 4 party collusion alerts can be seen in **Appendix C and D**.

Naturally these results cannot be taken as clear cut cases of market manipulation and further investigation must be done on the participants before concluding that the participants were intentionally manipulating the market. Methods to further analyze suspicious participants is discussed in the future work section but are not implemented in the current algorithm.

One issue would be that participants that triggered the alert could simply be trading normally and coincidentally created a circular trading scenario with other participant. This noise could possibly be removed by analyzing over many days of market data. For example if there was a group of participants that triggered the alert every two days for a month then it becomes far more suspicious than if the group of participants triggered the alert once. Furthermore smart traders will often weave in illegal activities with legal activities so that they can’t be detected. By analyzing over several days it is possible that their legal activities will be outnumbered by their illegal activities and this noise can be removed.

## Run time against real market data

The main problem of the algorithm is its run time as it has an exponentially scaling time complexity; it should be tested against real market data to see if the algorithm can scale properly with large amounts of data. Some data was collected over the ASX and BIST markets and it is shown in **Figure 15**. From the results in section 4.1 it was concluded that the algorithm has an exponential run time. This is further emphasized in the graph of the run times. It is clear that the run time increases exponentially with N.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Market** | **N Party** | **Participant Type** | **Time (hours)** | **#Trades** | **Time per trade (seconds)** |
| BIST | 6 | Broker | 10.5425 | 575745 | 0.065919808 |
| BIST | 5 | Broker | 1.9036 | 575745 | 0.011902769 |
| BIST | 4 | Broker | 0.0722 | 575745 | 0.00045145 |
| BIST | 3 | Broker | 0.01138 | 575745 | 7.11565E-05 |
| BIST | 2 | Broker | 0.007638 | 575745 | 4.77586E-05 |
|  |  |  |  |  |  |
|  |  |  |  |  |  |
| BIST | 5 | Client | 1.26 | 575745 | 0.007878488 |
| BIST | 4 | Client | 0.0608 | 575745 | 0.000380168 |
| BIST | 3 | Client | 0.0113 | 575745 | 7.06563E-05 |
| BIST | 2 | Client | 0.0083 | 575745 | 5.1898E-05 |
|  |  |  |  |  |  |
| ASX | 6 | Broker | 23.3691 | 1520650 | 0.05532421 |
| ASX | 5 | Broker | 0.644125 | 1520650 | 0.001524907 |
| ASX | 4 | Broker | 0.13916 | 1520650 | 0.000329449 |
| ASX | 3 | Broker | 0.113 | 1520650 | 0.000267517 |
| ASX | 2 | Broker | 0.11601 | 1520650 | 0.000274643 |
|  |  |  |  |  |  |
| ASX | 6 | Client | 15.50416 | 1520650 | 0.036704683 |
| ASX | 5 | Client | 5.01861 | 1520650 | 0.011881101 |
| ASX | 4 | Client | 0.278 | 1520650 | 0.00065814 |
| ASX | 3 | Client | 0.1233 | 1520650 | 0.000291901 |
| ASX | 2 | Client | 0.11386 | 1520650 | 0.000269553 |

Figure 15. Run time on real markets

The cells in the table which are red are due to crashes in the program due to memory over allocation. This is due to the use of a recursive algorithm. As the local variables and state of the program are stored for each recursion, the memory use grows with each recursion and sometimes will exceed a certain limit and the program will crash. When n is increased to 5 and above it is obvious that the number of groups that can be built is very large. Although the number of nodes is limited to a top X number there is still a large difference between and as the choose function scales quickly with n. Moreover if a number greater than 30 is chosen for the top X then the run time becomes worse as the number of groups would scale factorially with X(which is comparable to exponential scaling). It should also be noted that the amount of trades per day seemingly does not impact the running time. The more impactful variables are the number of nodes and most importantly n.

# Conclusion and future work

It can be seen in this paper and through the example results that using the designed algorithm to detect circular trading is effective. The results show cases of collusion as defined in earlier sections (low net group volume and high group total volume). This is especially clear when looking at the results for 2 party collusion with two participants both having 0 net volume by the end of the day. The same applies to the alert on 4 participants (**Appendix C**) as all of the parties have a low net position within the group while the total group volume is a large part of the total security volume for the day.

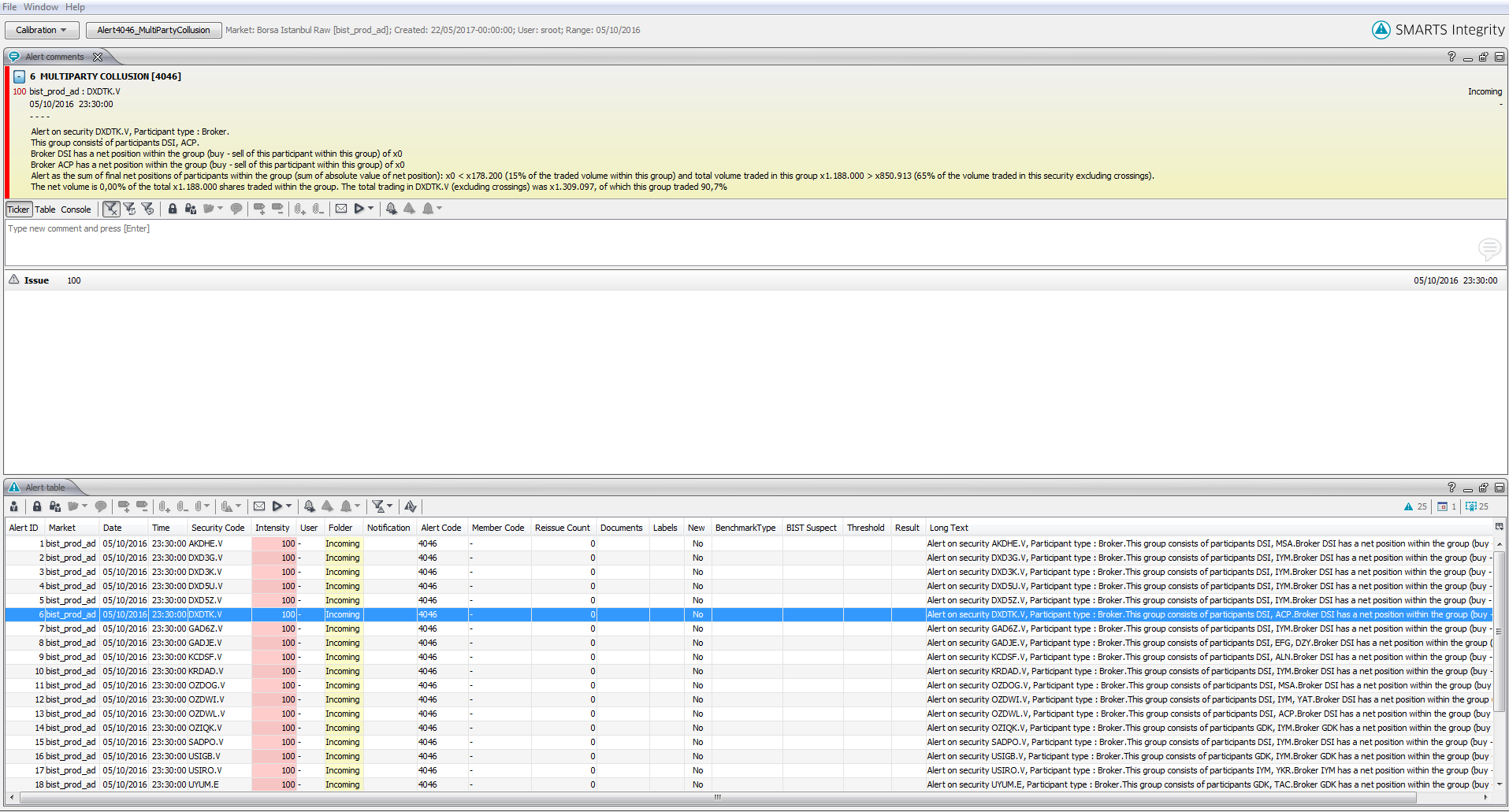
Undoubtedly there is much further work that can be done with this algorithm as there are a few short comings, especially relating to the run time. The run time could be improved on by improving the group building algorithm, or possibly by further pruning the graph to give less data to analyze. Another issue that must be improved on is the recursion crashing issue. These crashes could be solved by using an iterative algorithm instead of a recursive one.

Moreover the code was written in an internal language ALICE which is obviously not as extensive as any mainstream language and hence could be improved upon by using tools and libraries from other more sophisticated languages. This could benefit the time and memory issues with the solution described in this paper.

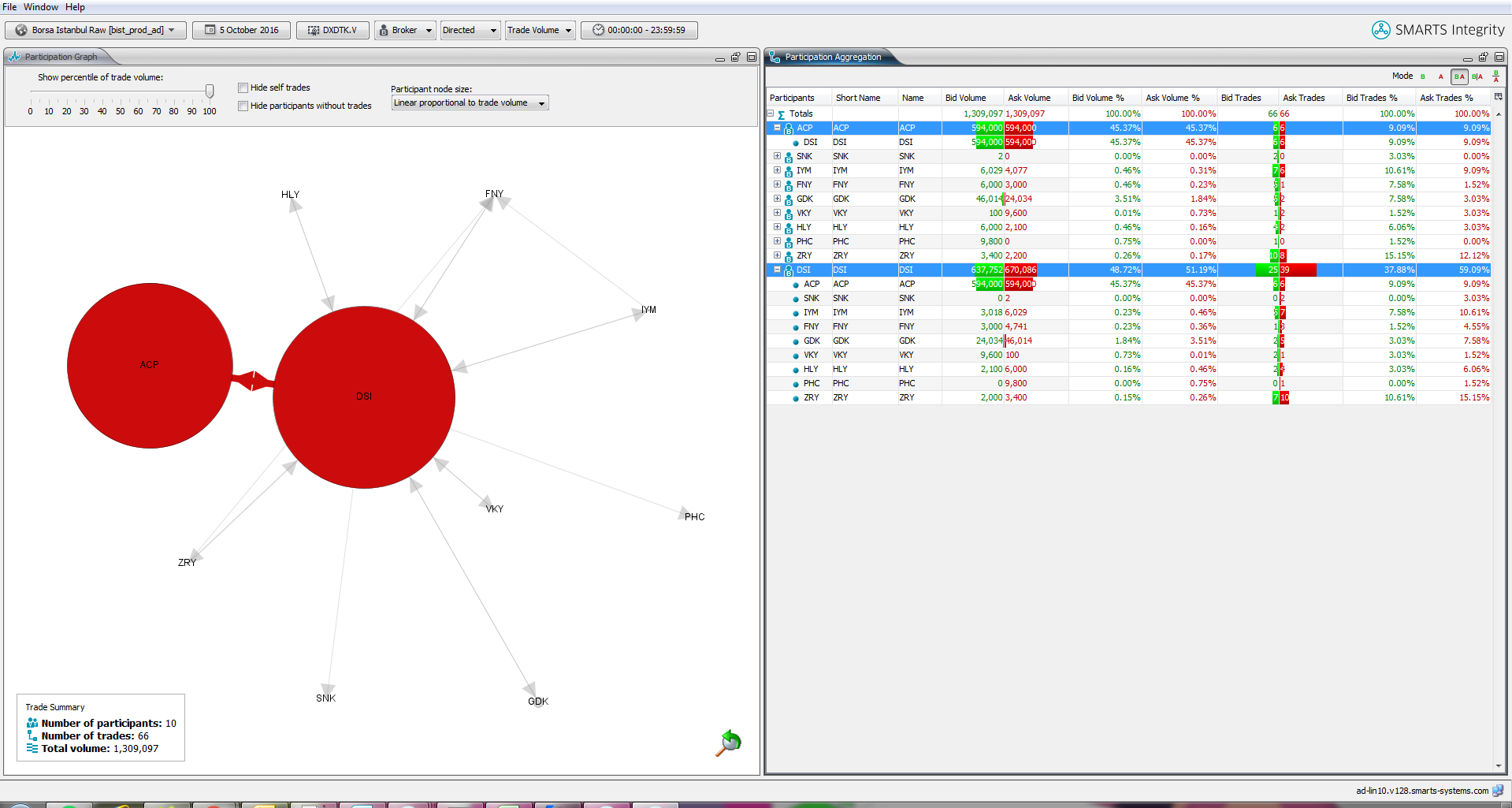
# References

1. Aggarwal, R., Wu, G. (2006). "Stock Market Manipulation." Journal of Business 79(4): 1915-1953.
2. Robert A. Jarrow (1992) . “Market Manipulation, Bubbles, Corners, and Short Squeezes” JOURNAL OF FINANCIAL AND QUANTITATIVE ANALYSIS VOL. 27, NO. 3, SEPTEMBER 1992 –
3. Investopedia. 2016. Investopedia. [ONLINE] Available at:  <http://www.investopedia.com/university/>
4. Nasdaq. 2016.  [ONLINE] Available at: <http://business.nasdaq.com/market-tech/market-participants/smarts-trade-surveillance-buy-side/otc-coverage>.
5. Liu, Ke; Lai, Kin Keung Yen, Jerome Zhu, Qing (Jan 2015). “A Model of Stock Manipulation Ramping Tricks” Computational Economics 45.1 135-150.
6. Allen, F., Gale, D. (1992). "Stock-Price Manipulation.” The Review of Financial Studies 5: 503-529.
7. Pirrong, C. 1997a. “The Inefficiency of U.S. Commodity law: Diagnosis and A Proposed Cure” 18 Research in Law and Economics 173-227
8. Pirrong, C. (2004). “Detecting manipulation in futures markets: The Ferruzzi soybean episode.” American Law and Economics Review, 6(1), 28–71.
9. Palshikar, G. K., & Apte, M. M. (2008). “Collusion set detection using graph clustering. Data Mining and Knowledge Discovery” , 16(2), 135–164.
10. Harris, F. R., Aitken, M., & Ji, S. (2011). “Trade-based manipulation and market efficiency after the introduction of real-time surveillance: A cross-market comparison.” Working paper, Wake Forest University.
11. Liski M. , Montero J.P 2005. “Forward trading and collusion in oligopoly” Working paper, Helsinki School of Economics and Montero
12. Mishra L. 2012 “Who's Ketan Parekh and why's he bad for the market?“. IndiaToday. Retrieved from: <http://indiatoday.intoday.in/story/ketan-parekh-stock-market-share-prices-sebi-k-10-stocks/1/208342.html>
13. Caesar E. 2016 “Deutsche Bank’s $10-Billion Scandal“. The New Yorker . Retrieved from: <http://www.newyorker.com/magazine/2016/08/29/deutsche-banks-10-billion-scandal>

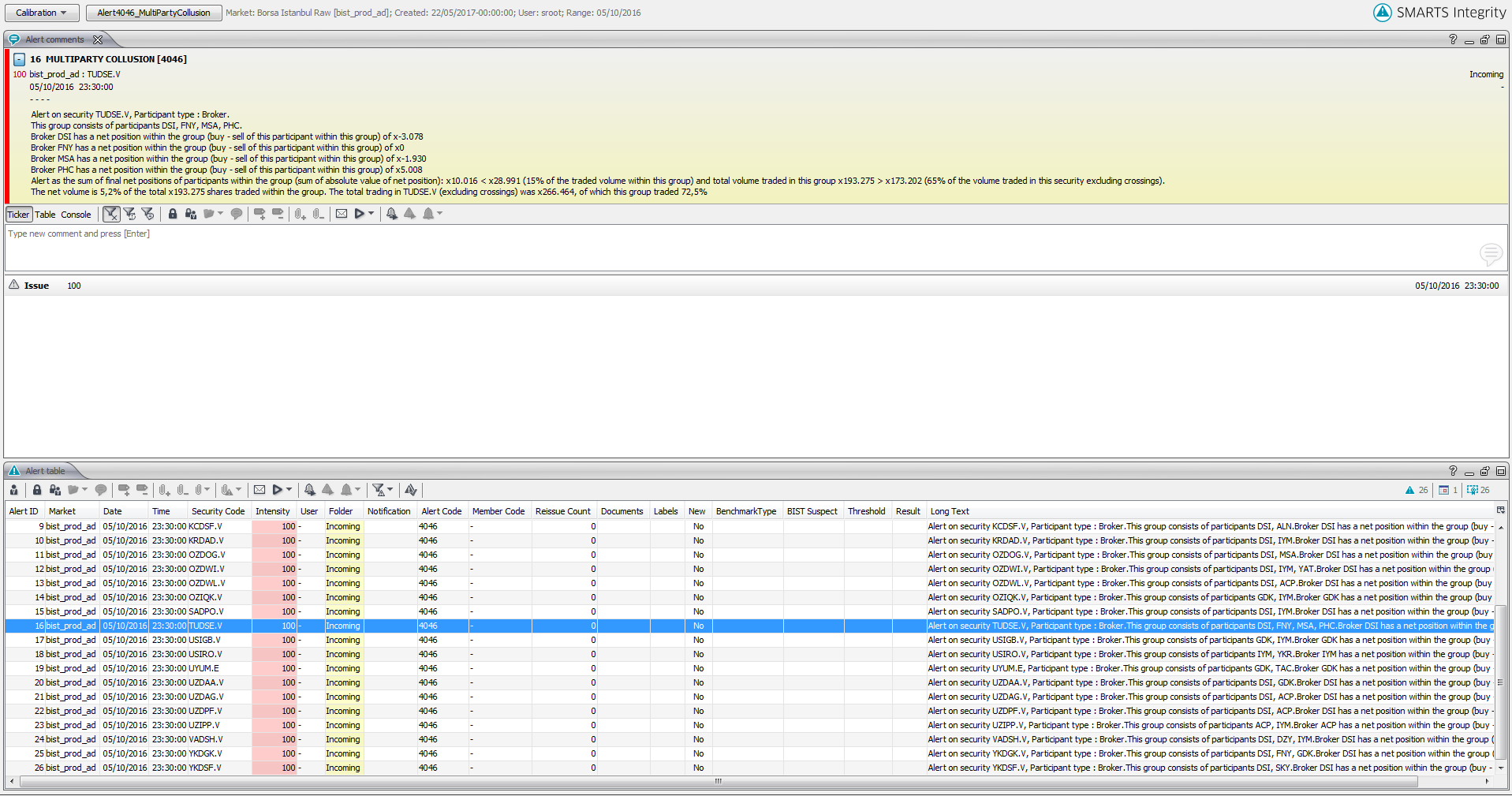
# Appendices



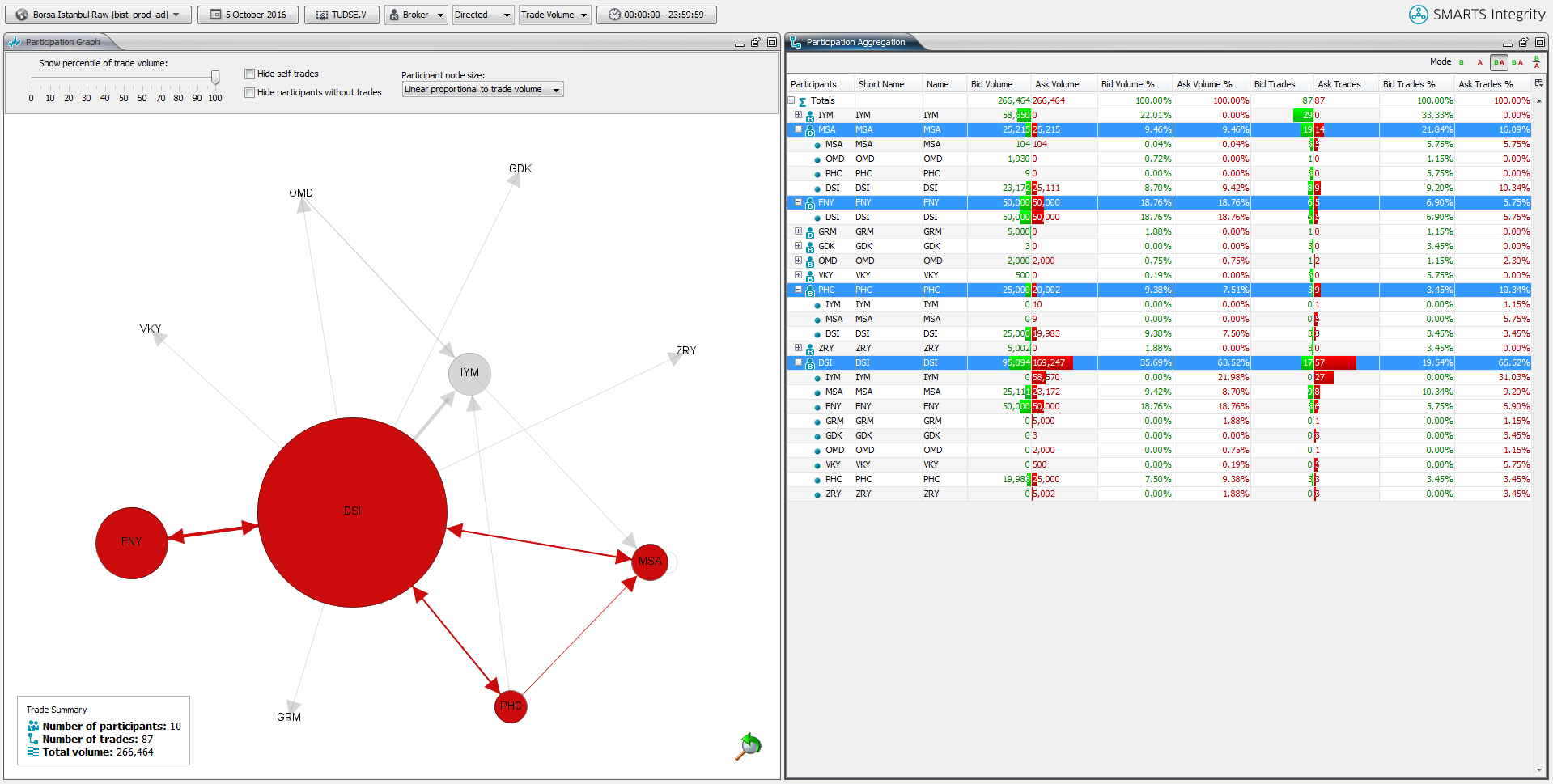
Appendix A. 2 Party collusion example result



Appendix B. 2 Party collusion example result graph



Appendix C. 4 Party collusion example result



Appendix D. 4 Party collusion example result graph