Dissertation Type: Research

DEPARTMENT OF COMPUTER SCIENCE

Studies of Response-Time Issues in Popular Trading Strategies via a Multi-threaded Exchange Simulator

Michael Rollins

A dissertation submitted to the University of Bristol in accordance with the requirements of the degree of Master of Engineering in the Faculty of Engineering.

Wednesday 19th August, 2020

Declaration

This dissertation is submitted to the University of Bristol in accordance with the requirements of the degree of MEng in the Faculty of Engineering. It has not been submitted for any other degree or diploma of any examining body. Except where specifically acknowledged, it is all the work of the Author.

Michael Rollins, Wednesday 19th August, 2020

Contents

List of Figures

List of Tables

List of Algorithms

List of Listings

Executive Summary

Financial exchanges have become an integral part of any modern economy. On these exchanges many billions of dollars of assets are exchanged between parties every year. Trades were formally executed by a pair of human traders, a buyer and a seller, each shouting their offers across a busy trading hall. In modern times traders have realised that by using automated trading algorithms, they can better extract profit from these markets. This has led to academics producing a series of papers proposing algorithms that they deem to best extract profit from financial markets. These papers have resulted in a "pecking order" or hierarchy from best to worst of how each algorithm performs in a variety of market scenarios. Previously, these algorithms have been tested on basic, single-threaded financial exchange simulators that abstract away many aspects of a real financial market. One of these simplifications is that the execution time of algorithms has not been able to affect their performance, each algorithm is given as much time as it needs to compute the price of the order it wishes to send to the exchange, before any other algorithm is given the opportunity to do the same.

In this project a new style of financial exchange simulator has been developed that uses multi-threading to allow multiple traders to execute their algorithms concurrently, allowing the execution time of these algorithms to have an impact to how they perform. This simulator is designed as an extension to Cliff's Bristol Stock Exchange (BSE)[2] and is therefore named the Threaded Bristol Stock Exchange (TBSE).

> The research hypothesis of this project is that automated trading algorithms with a slow execution time will perform less favourably on the multi-threaded financial exchange simulator TBSE than they would on a single-threaded simulator such as Cliff's Bristol Stock Exchange (BSE), because BSE is not multi-threaded.

As TBSE provides a test-bed much closer to real-world financial markets than BSE, my results from TBSE could overturn the conventionally accepted hierarchy of public-domain trading algorithms. Experiments were designed to compare six public-domain trading strategies AA, GDX, ZIP, ZIC, Shaver and Giveaway. The results of these experiments showed that the established hierarchy broke down, at least to some extent, when the algorithms were tested on TBSE rather than the single-threaded BSE. The most significant example was that the slowest of these algorithms, GDX, performed far worse on TBSE than BSE.

- Cliff's Bristol Stock Exchange (BSE) financial exchange simulator was extended to use multi-threading so that multiple traders can execute concurrently, this simulator was named TBSE. See Sections 3.1. Six popular trading algorithms were adapted to run on TBSE.
- A host of market experiments were designed to compare all six algorithms to each other, then these experiments were ran over 10,000 hours of market simulation. See Section 3.3.
- It was shown that the ratio of the number of traders from two different algorithm groups in a simulated market has an inherent affect on the outcome of their profit. See Section 4.1.
- It was demonstrated that a slow executing algorithm, GDX, which was previously argued by IBM to be the best performing public-domain trading algorithm at the time of its introduction[14], performs less favourably on TBSE than BSE due to the run-time of algorithms being allowed to properly affect their performance in TBSE. See Section 4.3.
- The TBSE source-code has been made publicly available at `https://github.com/MichaelRol/Threaded-Bristol-Stock-Exchange`, so that others can replicate and extend this work.
- I co-authored (with my supervisor) a paper submitted to a peer-reviewed international conference: The European Modelling and Simulation Symposium (EMSS). The paper was accepted for oral presentation at EMSS in September 2020, and we have been invited to submit an extended version of the paper to a special issue of an international peer-reviewed journal, submission due in November 2020. A copy of our accepted EMSS paper is included in this dissertation, as Appendix A.

Supporting Technologies

- Cliff's Bristol Stock Exchange (BSE) financial exchange simulator was extended into a multi-threaded concurrent system. Code used for ZIC, ZIP, SHVR and GVWY was adapted from Cliff's trader agents. All source code can be found here https://github.com/davecliff/BristolStockExchange.
- Daniel Snashall's GDX and AA trading algorithms were used (with minor adaptations), also found in the above repository. The paper which describes the use of these algorithms can be found in [13].
- Amazon Web Service's (AWS) Elastic Compute Cloud (EC2) was used to host the range of experiments detailed in this dissertation. Around 10,000 hours of EC2 CPU time was used. Details of EC2 can be found at https://aws.amazon.com/ec2/.

Notation and Acronyms

CDA	:	Continuous Double Auction
LOB	:	Limit Order Book
BSE	:	Bristol Stock Exchange
ZIC	:	Zero Intelligence Constrained (Trading Algorithm)
ZIP	:	Zero Intelligence Plus (Trading Algorithm)
AA	:	Adaptive Aggressive (Trading Algorithm)
GDX	:	Gjerstad Dickhaut eXtended (Trading Algorithm)
AWS	:	Amazon Web Services
EC2	:	Elastic Computer Cloud
	⋮	

Acknowledgements

I would like to take this opportunity to thank my supervisor, Prof. Dave Cliff, for all his advice and help throughout this project, without which I'm sure this project would not have been a shadow of what it has become. I'd also like to acknowledge the care and support I've received from my parents, flatmates and coursemates, without whom this project would not have been possible. Finally, a special word of thanks goes to Syritta Ltd for sponsoring the experiments used in this project.

Chapter 1

Contextual Background

This dissertation explores the world of automated trading algorithms, algorithms designed to take the place of traditional human traders in financial exchanges.

This chapter will outline the background behind key topics discussed in this dissertation. It will start by looking at financial markets themselves, then how the work of Vernon Smith founded the discipline of experimental economics and how that work has been built upon to develop and compare trading algorithms, which will lead to a brief history and explanation of major trading algorithms. The final sections of this chapter will express the importance of this work and the challenges associated with it.

1.1 Financial Markets

In 1602 the Amsterdam Stock Exchange was founded by the Dutch East India Company in order to trade stocks in their company, this established the first modern securities market in the world. Since then stock markets have been created in most of the world's major economies, and the total market capitalisation of securities (the total value of all outstanding shares in all companies traded on stock markets) around the world has increased dramatically, from $2.5 trillion in 1980 to $68.65 trillion in 2018 [19].

1.1.1 Trading

The methods behind trading assets remained very much the same for nearly four hundred years after the stock exchange was born, there would be a physical meeting of people who had the intent to either buy or sell some kind of asset or commodity. The traders would aggressively shout the price that they either wanted to buy or sell the asset for, to try and grasp the attention of opposing traders, then if two traders could agree then that trade could be executed. This continued through most of the 20th century where trading would be carried about by skilled and highly paid traders in one of many major financial exchanges around the world. The manual execution of transactions was slow, inefficient and fallible, so as computational power became more accessible in the 1980s automated exchange systems were adopted by many of the world's major markets and the work of the exchanges was moved onto a digital platform. The digital execution of transactions also opened up a new opportunity for the development of algorithms that could replace the work of human traders, and in the 1990s the first algorithmic traders were invented, algorithms that could calculate prices at which to buy or sell assets in such a way as to be more profitable than their human counterparts. More detail on algorithmic traders is given in Sections 1.3 & 2.4.

1.1.2 Traders

In this dissertation "traders" will refer to sales traders. The job of a sales trader is to execute orders on behalf of customers. Customers will request that a trader either buys or sells a number units of an asset at a given price. The trader must then attempt to execute this request by posting either a bid (an offer to buy) or an ask (an offer to sell) to the exchange. The price given by the customer is often referred to as the trader's *limit price*, the price that the trader cannot go above if buying an asset or below if selling an asset. This is where the trader has an opportunity to make a profit. If a trader has been asked to buy X units of stock ABC at price L, the trader can attempt to buy the stock at a price lower than L which will be called call P. If successful, the trade will generate a profit of $(L - P) * X$. The reverse is true if the trader receives an order to sell X units of stock ABC at price L, here the trader wants to get a sale

price that is higher than L, so that the additional money generated, above L, is received by the trader as profit equalling $(P - L) * X$.

1.1.3 Bristol Stock Exchange

The Bristol Stock Exchange[2] is a basic simulation of a financial exchange for a single financial instrument. It was originally developed by Cliff in 2012 at the University of Bristol as a teaching aid to allow students to experiment with developing and testing automated trading algorithms. Although it is a powerful tool, like many simulated exchanges it is not a true representation of a modern-day financial exchange as it ignores many of the complexities that are found in real financial exchanges. One major simplification is that BSE is a single-threaded program, meaning that the parallel and asynchronous nature of a real-world financial exchange is only very crudely approximated, and hence issues such as algorithm processing time cannot be explored. This project extends the current BSE system to allow for multi-threaded execution of traders so that all traders can perform their calculation concurrently. This will allow the response times of the algorithms to factor into their performance.

1.2 Market Experiments

Experimental economics is a field of study founded by the economist Vernon Smith, awarded the Nobel Prize in 2002. It takes an experimental approach to the study of market behaviour. In 1962 Smith published his paper "An Experimental Study Of Competitive Market Behaviour" [12] describing experiments that he ran with a group of students, each acting as either a buyer or a seller, in order to analyse the behaviour of a market. These experiments showed that markets behave predictably and that the price at which trades occur rapidly reach an equilibrium which can be predicted by controlling the demand and supply of the asset, even when the number of traders involved is very small. More detail of how these experiments were run is included in Chapter 2.

Many years later, with the invention of algorithmic traders, similar experiments have been run in order to compare the performance of traders against humans [6] and to compare different traders against each other [15]. These experiments are performed in a very similar way to the way that Smith describes in his paper, except rather than the experiments being conducted in a classroom, they are instead run on computers which simulate the behaviour of an exchange. The work in this thesis continues that tradition.

1.3 Trading Algorithms

Trading algorithms are pieces of software designed to perform the same job as human traders. By replacing human traders with algorithmic traders vast sums of money can be saved as human traders are generally very well paid. It also gives an opportunity for algorithms to be invented that can perform better than their human counterparts and generate a greater profit. Since the early 2000s there has been a rapid increase in the use of computerised trading agents to take the place of human traders and by 2016 over 80% of trading done on Foreign Exchange (Forex) markets was performed by automated trading agents[1].

The obvious benefits of using algorithms instead of human traders led academics to gain an interest in the area. The earliest published automated traders did little more than randomly generate prices, but even with just minor constraints were able to produce surprisingly human-like results. This led to a series of papers being published, each claiming to have produced a superior algorithm to those previously developed, and to the development of financial exchange simulators in which to test this algorithms against each other. However the majority of these papers only compare the "intelligence" of these algorithms, essentially giving each algorithms as much time as it needs to decide to post a bid or an offer and then at the end of a trading period of set length determine which generated the greatest profit. This kind of experiment ignores the execution time of the algorithms. For example, if two traders running different algorithms, one slow but intelligent and the other more simplistic but faster, were to compete for the same asset, could the more simplistic algorithm react more quickly to changes in the market and then "steal the deal" from the intelligent algorithm and therefore be more profitable? This is the question that this dissertation intends to explore.

One paper that has since been widely cited as an influence on the development of algorithmic trading was Gode and Sunder's 1993 paper[9] where they described two "zero intelligence" traders. They were named this way as they make no use of any information about the market and simply randomly generate

bid/offer order prices. The first is called Zero Intelligence Unconstrained (ZI-U): this simply posted random prices regardless of whether they would create a profitable trade. The second is Zero Intelligence Constrained (ZIC), the constraint being that they will never generate an order price which would result in a loss-making deal. In their paper Gode & Sunder find that ZI-U was highly inefficient and performed far worse than human traders, however ZIC's performance was surprisingly comparable to that of the human traders.

Following on from Gode and Sunder's paper, in 1997 Cliff[4] published an algorithm called Zero Intelligence Plus as part of a Hewlett Packard technical report. ZIP uses the simple Widrow-Hoff machine learning algorithm to adjust a desired profit margin based on events within the market. Like with ZIC, the bid or ask price is constrained not to make loss making trades. Cliff found that ZIP traders produce market dynamics significantly more like human traders than ZIC traders and the market price converges quickly to equilibrium.

In 1998 Gjerstad & Dickhaut[8] published a paper presenting a new model for information processing and strategy choice for agents in a double auction. In their algorithm buyers and sellers generate "belief" functions from the recent history of market activity. These belief functions estimate the probability that at at a certain price their ask or bid will be accepted with individual trades within the history acting as evidence to guide this probability. Although they did not give this algorithm a name, it has since been widely referred to as "GD".

In 2001 whilst working at IBM Tesauro & Das[15] made adjustments to the GD strategy in order to address issues of excessive volatility under certain conditions and to allow for "stingier" bidding practises to produce higher profits. This algorithm is known as modified GD (MGD). MGD was shown to outperform ZIP in a variety of tests.

The next generation of automated trader was also developed at IBM by Tesauro & Bredin, which further extended the GD system to create GDX[14]. The key difference between GDX and the algorithms that came before it is that GDX maximised long-term profits over the entire trading period, rather than in previous strategies where traders are simply designed to maximise the immediate profits with each trade. It does this by making use of a similar set of belief functions as used in the GD and MGD systems but combining it with dynamic programming. Each GDX trader includes a discount parameter γ which for values close to one maximises long term profits, and for values close to 0 maximises short term profits acting like the original GD algorithm. Teasuro & Bredin found that GDX performed far better when γ was set to a high value, outperforming both GD and ZIP.

Vytelingum's Adaptive Aggressive (AA) trader[18], published in 2006, is the most recently created algorithm studied in this project. This algorithm makes use of an aggressiveness parameter, this affects the likelihood of the trader to post a bid or not, a more aggressive trader will be more likely to post a bid to make an immediate profit, whereas a more passive trader is more likely to wait for a bigger profit margin later in the trading period. Vytelingum ran around 25,000 tests comparing AA to ZIP and GDX and found that it outperformed both of them. This was later further supported in 2011 by De Luca and Cliff[7] where they found that the performance of AA against human traders was better than that of ZIP, GD and GDX. However in 2015 Vach[16] found by varying the ratio of AA traders to one or more other traders it was in fact possible for both ZIP and GDX to outperform AA.

Two other algorithms, Giveaway and Shaver, are used in this project. Giveaway simply posts orders at its limit price, and Shaver will always try to add a penny onto the best bid, or shave a penny off the best ask that have been posted. Technical details of these traders are included in Section 2.4.

1.4 Importance

Understanding algorithmic traders and the influence they can have on markets is very important as vast amounts of assets are exchanged via automated trading systems every day. A 2016 study by Bigiotti & Navarra [1] suggested that over 80% of foreign currency exchange was performed by algorithmic traders. With algorithmic traders dominating exchange markets it is vitally important to have a thorough understanding of their operation and the affect they can cause on those markets. A memorable example of why this is so important is the 2010 Flash Crash in the United States. On the 6th of May 2010 US stock markets lost around $1trillion in value in just 36 minutes. The cause of the crash is the subject of a great amount of speculation and has not be conclusively confirmed. Although the SEC's report [5] after the event concluded that high frequency algorithmic trading was not the main cause of the crash, they may have magnified the crash by immediately reacting to the crash by sending huge numbers of orders and overloading exchanges. This highlights how a lack of understanding can have dramatic results, so

there must be a better understanding of these algorithms so that market stability can be increased. This project hopes to work towards that by providing a realistic platform for simulating financial markets.

1.5 Project Summary

This project intends to build on the simple single-threaded financial exchange simulations that have been built to support previous research. It does this by using multiple threads so that each of the traders involved in the simulation can run concurrently to each other and to the exchange itself. By running each trader in its own computational thread, the time in which each algorithm takes to execute can be taken into account when comparing these algorithm's performance, thus creating a more realistic comparison of how the algorithms in question would perform in a real-world setting. Having created a simulator which implements these techniques, multiple test have been run using a vast array of configurations to thoroughly tests the differences between how the algorithms that are explored in this project, perform on a multi-threaded simulator and a single-threaded simulator. The goal of this project is to assess whether the current hierarchy of algorithm performance, that has been established by research experiments that have been executed on single-threaded simulators, breaks down when similar experiments are run on a multi-threaded system.

1.6 Challenges

This project posed a number of challenges. The first was to understand the inner workings of the Bristol Stock Exchange and each of its component algorithms. It was also necessary to understand the processes by which each trading algorithm chose to, or not to, make an order and how it calculated its order price. This was particularly difficult as the more complex traders (such as AA and GDX) are made up of many different sub-algorithms, hundreds of lines of code and a variety of programming techniques such as the dynamic programming within the GDX algorithm. It was also essential to understand the ways in which these algorithms have previously been tested and compared to each other, as the conclusions previous researchers have come to have not always been in agreement.

Once sufficient knowledge of the platform and algorithms with which this project would be working had been acquired, the next challenge was to modify the BSE code to support each trader, and the exchange itself, to run on separate computational threads, this allows for all agents and the exchange to run concurrently. This was particularly difficult as in the original BSE system all communication and updates relating to one order, and the trade it may or may not result in, are completed and published before another trader can submit an order, however in a multi-threaded system, as each agent and the exchange are working separately, when a trade is completed involving two agents there is no guarantee that both those agents will be informed by the exchange that the trade occurred before they submit a second bid/offer for the same order. This poses the risk of a trader being involved in two trades for the same order. There were multiple communication concerns like this one that needed to be overcome in order to be certain that the exchange was matching the correct orders and then executing those trades. It was also important to ensure that traders were responding appropriately to changes in the market and were being updated regularly with this information so that it could be known that they were functioning as designed.

The final challenge of this project was to run a similar kind of testing and comparison to that described by those who have done the most thorough testing of these algorithms, Cliff & Snashall[13], and Vach[16]. This involved running hundreds of thousands of simulations, which on a single machine would take many thousands of hours of compute time. To solve this issue multiple simulations were ran simultaneously on separate Amazon Web Services EC2 cloud compute units. Once testing had been completed it was necessary to analyse and aggregate the results in order to understand which algorithms were superior under what conditions, and from this form a hierarchy of weakest to strongest trading algorithms.

1.7 Summary

The high-level objective of this dissertation is to research whether the currently established rank-order hierarchy of trading algorithm performance remains when the algorithms are tested on a multi-threaded exchange simulator, which allows these algorithms, for the first time, to be evaluated when operating in an asynchronous parallel setting, a test environment much

closer to real-world trading than the simple test-beds used in previous research. The aims of this project are to:

1. Research and survey literature on trading algorithms and how they have been compared in the past. This should reveal the current hierarchy of algorithm performance.
2. Adapt Cliff's Bristol Stock Exchange (BSE) simulator to use multi-threading to allow trading algorithms to execute concurrently.
3. Adapt existing trading algorithms designed for use on BSE to work on the new multi-threaded system.
4. Use exhaustive testing, similar to the testing described in relevant literature, to compare a variety of trading algorithms, all with different execution times, to analyse their performance on a multi-threaded exchange simulator.
5. Analyse the results and use them to decide whether the establish hierarchy breaks down when the algorithms' response times are consider, and if so determine the new hierarchy.
6. Make my code publicly available via an online open-source code repository such as Github.
7. Publish my results and analysis in one or more suitable international peer-reviewed journal or conference papers.

Chapter 2

Technical Background

2.1 Financial Markets

2.1.1 Continuous Double Auction

Auctions are one of the oldest methods by which humans have traded goods, with records of them taking place as early as 500 BC[10]. Two of the simplest and most common auctions are ascending bid (or English) auctions, and descending bid (or Dutch) auctions. In an English auction an auctioneer will announce a starting bid price, then from a group of competing buyers, someone will indicate their willingness to accept that bid, and the auctioneer will increase the bid price, then as each bid is accepted the auctioneer will continue to increase the price until there is only a single buyer left willing to pay the bid price. That buyer wins the auction and pays the highest bid price they accepted. Dutch auctions are in many ways the reverse of this. In this style of auction, the seller initially sets a high asking price for the good they are selling, if no buyer accepts this price they will slowly lower the price until the first buyer accepts their price.

A Continuous Double Auction (CDA) is a combination of both an English and a Dutch auction. They are named continuous as there is no set end to the auction, multiple sellers can keep trying to sell and multiple buyers can keep trying to buy, however, for practicality they are normally divided into fixed trading periods, often normal business hours. At any point in time during the trading period, a buyer can announce a bid, the price they are willing to pay for the asset, and a seller can announce an ask, the price they are willing to sell the asset for. These bids and asks are visible to all traders in the auction. Once a buyer announces a bid that is equal or higher than the lowest outstanding ask, or a seller posts an ask which is equal or lower than the highest outstanding bid, the trader who posted the order is said to have *crossed the spread* and the two traders are matched. The trade is completed at the price that was first submitted, the price that was crossed or matched. In many (but not all) CDAs, it is not necessary for the quantity of the buy order and the sell order to be equal, the trade will take place at the lower quantity and the remainder of the other order will remain in the auction. It is CDA-style markets that this dissertation will be be interested in.

In Vernon Smith's 1962 paper[12], which introduced the methodology of experimental economics, he showed, when human traders interact in a CDA market, the transaction prices (the prices at which trades take place) reliably converge on the market's theoretical competitive equilibrium price (explained further in Section 2.2 below), which is the most efficient price for the market. It has also been shown that using automated trading algorithms in place of human traders also causes the transaction price to converge on the equilibrium price even more efficiently[15]. The CDA has become the trading institution of choice for many modern financial markets, such as the NASDAQ and NYSE, trading a variety of different assets such as foreign exchange, equities and derivatives.

2.1.2 Limit Order Books

In a CDA there is no centralised auctioneer so the lists of bids and asks which have been posted are recorded in a Limit Order Book (LOB). The LOB displays a summary of all outstanding bids and asks, ones that have not been accepted or cancelled by the trader who posted them. One side of the LOB displays the bids in descending order, so the best (highest) bid is on top, with the quantity being bid on at each price displayed beside it, on the other side, the asks are displayed in ascending order, so the best (lowest) offer price is on top, with the quantity being sold at each price displayed beside it. The LOB is

Asks		Bids	
Quantity	Price	Price	Quantity
5	87.4	85.1	3
1	87.9	84.2	1
1	88.5	82.3	2
2	89.1	80.9	1
4	90.3		
1	92.1		

Table 2.1: An example of a Limit Order Book

the information that traders will see about a market and use to decide the price at which they want to post their bid/ask. An example of what a LOB might look like is shown in Table 2.1.

There are some key terms and definitions that it is important to understand when discussing Limit Order Books in CDA markets. The *best bid* is the highest current bid on the LOB, and the *best ask* is the lowest current ask on the LOB. The *bid-ask spread* is simply the difference between the best ask and the best bid, this is often simply be referred to as the *spread*. When looking at a LOB it is often desirable to have an estimate for the current price of the asset being exchanged, however, as the LOB displays two different "best" prices, it is necessary to calculate other values which better represent the current price. One way of doing this is simply to look at the last price at which a trade took place, although this is not always an accurate estimate as the price may have shifted significantly since the last trade took place. Another estimate is the *midprice*, this is simply the mean of the best ask and the best bid. The weakness with this estimator is it ignores the quantity being offered at each price. If the quantity of the asks is greater than the quantity of the bids, then there is an imbalance between demand and supply resulting in excess supply. This pushes prices down towards the best bid. Conversely, if there is excess demand, where there is a greater quantity on the bid side of the market, then prices will be pushed up towards the best ask. In order to take into account both the prices being quoted as well as their quantities the *microprice* uses a weighted mean and is defined as:

Where P^a = best ask price, P^b = best bid price, Q^a = best ask quantity, and Q^b = best bid quantity,

$$microprice = \frac{P^a * Q^b + P^b * Q^a}{Q^a + Q^b} \tag{2.1}$$

When an exchange publishes a Limit Order Book they will also often publish a *tape* this is a time-stamped record of all the previous transactions that have executed in the current market session, noting the time, the price that was paid and the quantity traded.

2.2 Market Economics

2.2.1 Microeconomics

Financial markets, like the ones discussed in this dissertation, are governed by microeconomic principles. Microeconomics is the study of how individuals and firms interacted in order to allocate scarce resources. Real world examples of a scarce resource could be commodities, such as oil, sugar or coffee; shares of companies; or even different types of currency. When currency is exchanged it is known as foreign exchange (Forex). Markets are where those with assets they wish to sell come together with individuals and firms who wish to purchase that asset, in order to make trades. The most common microeconomic model for competitive markets is the model of supply and demand. *Supply* is the quantity of an asset that is available to be purchased over a range of possible prices. At a particularly low price, there may only be a few sellers willing to sell at that price, whereas at a very high price nearly all sellers are likely to be willing to sell at that price. *Demand* is the amount of an asset that buyers are willing to purchase over a range of prices, at low prices, a large number of buyers will be willing to buy at that price, but at high prices there may only be a few buyers who would be willing to pay so much. These concepts can be represented by plotting price against quantity and drawing curves representing both the supply and the demand, shown in Figure 2.1. Supply and demand curves do not need to be linear as they are in Figure 2.1, however this dissertation deals exclusively with linear supply and demand curves. The point at which the demand and supply curves cross is known as the *competitive equilibrium*, the supply and

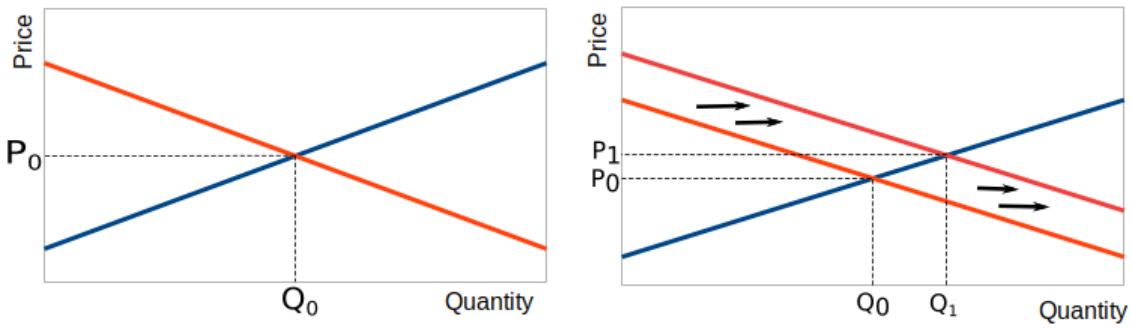


Figure 2.1: Example Supply and Demand curves. Supply shown in blue, demand in red. The right hand graph shows a positive shift in demand.

demand model states that assuming there are no external factors this is the price at which the market will settle and trades will be made. Figure 2.1 shows values P_0 as the equilibrium price and Q_0 as the equilibrium quantity.

If there is a change in the demand or supply of an asset, the curves can move, this can be caused by a range of factors. When there is a change in either the supply or demand curve, this is known as a *market shift*, If, for example, there is a problem with the supply chain that produces an asset, the supply of the asset will decrease meaning that the quantity available on the supply curve will decrease, moving the supply curve to the right. This will cause the competitive equilibrium to increase in price, but at a lower quantity. This is known as a negative shift in supply. On the other hand, if for some reason the supply of the asset increases, the supply curve will move positively along the quantity axis, resulting in the equilibrium price decreasing, and the quantity increasing. This is a positive shift in supply. Demand curves can be affected in much the same way. If there is an increase in demand for an asset, for example, an unexpectedly positive financial report from a company, causing the demand for shares in that company to increase, then the quantity of that asset being demanded by the market will increase. This will increase the equilibrium price and quantity, an example of this is shown on the right hand side of Figure 2.1, with P_1 and Q_1 representing the new equilibrium price and quantity respectively. This is called a positive shift in demand. If the demand decreases for an asset, the quantity being demand by the market will decrease, causing the equilibrium price and quantity to decrease, known as a negative shift in demand.

2.2.2 Performance Measurement

In previous work a range of metrics have been used to measure the performance of markets and of traders, whose performance can then be compared to each others. The simplest way to measuring the performance of traders is to measure the amount of profit each trader makes over the course of the trading period. If there are multiple traders running different algorithms, you simply average the profit from all traders that use the same algorithm and observe which algorithm generated the greatest profit per trader.

To understand the other performance metrics used, it is necessary to understand the theoretical equilibrium price often written as P_0. This value can be calculated as the intersection point of the underlying supply and demand curves in the market. In Smith-style experiments, like those often used to compare algorithm performance, these supply and demand curves are drawn from the limit prices given to the traders, so the theoretical equilibrium price can be easily calculated even before an experiment has began. Of course in real markets it is impossible to know the limit prices that traders have been given, so such a calculation is not possible.

In Smith's 1962 paper[12], he introduced a coefficient of convergence, which has become known as *Smith's* α. This is a measure of how volatile a market is as a whole, rather than a measure of an individual trader's performance, and is define as the root mean square deviation of the transaction prices from P_0, it is then expressed as a percentage of P_0. So if $P_0 = 0$, then all transactions occurred at P_0.

Allocative Efficiency (AE), defined in [9], is a method of measure how well a whole market extracts profits through trade, and is calculated by dividing the total profits earned by all traders in the market, by the *Theoretical Equilibrium Profit* (TEP) which is the total profit generated by the market if all trades where to take place at P_0, this is the maximum profit that can be generated. *Profit Dispersion* (PD) is a similar calculation used for measuring a single trader's ability to extract a profit from the market. It is defined as the profit made by the trader, divided by the profit that trader would have made if every trader

```
buyers_spec = [('ZIC', 5), ('ZIP', 5),
               ('GDX', 5), ('AA', 5)]
sellers_spec = buyers_spec
traders_spec = {'sellers':sellers_spec, 'buyers':buyers_spec}
```

Listing 2.1: An example *trader_spec* with 10 ZIC, ZIP, GDX and AA traders, equally split between buyers and sellers.

was executed at P_0. AE has a maximum values of 1, which is where all trades execute at P_0, whereas PD can have a value greater than 1, as a better preforming trader can capitalise on an under-performing trader.

2.3 Bristol Stock Exchange

The Bristol Stock Exchange[2] (BSE) is an example of a simple single-threaded financial exchange simulator, written in Python 2.7, on which a single asset is exchanged via a LOB, as described in Section 2.1.2. In order to maintain its simplicity and ease of use, it includes a number of simplifications compared to a real financial exchange. These can be summarised as:

- A trader can only have a single order on the LOB at any one time, if that trader submits a new order to the exchange it replaces any order already on the LOB.
- Due to BSE being single threaded, only one trader can act at a time. This means that if a trader submits an order which alters the LOB, the updated LOB is distributed and seen by all other traders before any further action is taken.
- Traders may only submit orders with a quantity of 1.
- Only a single asset is traded on BSE, whereas a real exchange would have multiple LOBs for multiple different assets.

BSE consists of three main components; the exchange itself, which is responsible for matching orders to complete trades and updating and managing the LOB; a system for generating and distributing customer orders, based on a number of parameters that can be altered when running each experiment; and a loop for processing a market session. BSE also contains implementations of common trading algorithms, more detail on these can be found in Section 2.4.

When initiating a market session there are a number of input variables that the user can use to define the conditions of the market. The most important of these are the *trader_spec* and the *order_sched*.

2.3.1 Trader Specification

The *trader_spec* is a dictionary containing two arrays, one that defines the *buyers_spec* and one that defines the *sellers_spec*. Each of these specify how many traders on each side of the auction there are, and which algorithms they run. It is common for tests to be run using the same number of traders running each algorithm on both the *buyers_spec* and the *sellers_spec* as all the popular trading algorithms used in academic literature act symmetrically as buyers or sellers.

2.3.2 Order Schedule

The *order_sched* consists of a *supply_schedule* and a *demand_schedule* as well as some variables which define when orders get passed to traders. The *supply_schedule* defines the supply of the asset at any given time, and the *demand_schedule* does the same for the demand. Each schedule is an array which holds a number of dictionaries which define the supply or demand of the asset between a range of times, this allows the user to have one dictionary in the array if they want a fixed schedule for the entire duration of the market session, or multiple dictionaries if they want to implement market shifts throughout the session. Each dictionary has four keys, two which define the start and end times, then an array which holds the range of values between which limit prices should be generated, it is also possible to include in this array an *schedule_offsetfn*, which is a function that offsets the limit prices over time. The final

dictionary key is for the *stepmode* and can take one of three values which tells BSE how to generate the limit prices within the given range. The three possible values are:

- 'fixed' - prices have constant difference between them.
- 'jittered' - prices are generated in a similar way to 'fixed' but with a small amount of random noise.
- 'random' - prices are drawn at random from a uniform distribution.

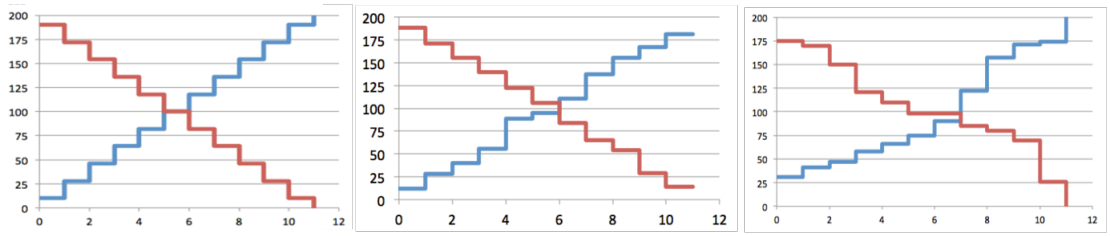


Figure 2.2: Supply and Demand schedules generated by BSE with *stepmode*s from left to right: fixed, jittered and random. Taken from Cliff's BSE Guide[2]. Horizontal axis is quantity, vertical axis is price.

As well as the schedules for the supply and demand the *order_sched* has two final keys which specify the timing by which the customer orders are distributed to traders, *interval* and *timemode*. The *interval* is how regularly new customer orders are generated. The *timemode* can take four values:

- 'periodic' - all customer orders will be distributed to traders at the same time every *interval* seconds.
- 'drip-fixed' - each customer order is distributed at a regular interval over the course of the *interval* i.e. if the *interval* is p and there are n traders, there will be p/n seconds between the arrival of each order.
- 'drip-jitter' - Similar to 'drip-fixed' but with an additional delay of between 0 and p/n seconds, determined by a uniform distribution.
- 'drip-poisson' - customer order arrivals are modelled as a Poisson process.

The *customer_orders()* function takes the order schedule and trader specification as input and is responsible for the distribution of the customer orders to each of the traders.

2.3.3 Market Session

Each market session consists of a loop which runs for the duration of the session. On each iteration of the loop it first runs the *customer_orders()* function which checks if new orders need distributing, and then does so if needed. Then a trader is chosen at random and its *getorder()* function is called: this function determines whether the trader wishes to post an order and if so, at what price. If an order is returned it is then processed by the exchange via the *process_order()* function. If the order crosses the spread and results in a trade, both of the traders involved call their *bookkeep()* function which updates their internal records with their profit. Finally each trader calls their *respond()* function to update its internal variables according to the new information on the LOB. Once this has been completed for every trader the loop starts again and a new random trader is polled for an order. Once the market session has finished two files are generated, one *transactions.csv* records the time and price at which each trade was executed, and a *balances.csv* file which records the profits of the different kinds of traders.

2.3.4 Exchange

The main functions of the exchange are to match orders to execute trades and maintain, update and publish the LOB. The core of the exchange is the *process_order()* function, when an order is submitted to the exchange, this function first adds the order to the LOB overwriting any preexisting orders from the same trader. The next step is to check whether the new order crosses the spread by comparing its value to the best bid or ask on the LOB, depending on whether the new order is a bid or an ask. If the

spread has been crossed the best ask and bid, which are the orders involved in the trade, are removed from LOB and a record of the transaction is returned.

It is also necessary for the exchange to regularly publish the publicly available information from the LOB. This is done via the *publish_lob()* function. This function returns a dictionary which contains the best and worst bids and asks, the number of bids and asks on the LOB and a full anonymised (details of which traders posted which offer are removed) copy of the LOB. This dictionary is then passed to each trader before they run their algorithm to submit an order.

2.3.5 Timing

BSE operates using a simulated clock with discreet time intervals. Each time the main market session loops executes, the simulated clock progresses by 1/N seconds, where N is the total number of traders operating in the market. This means that regardless of how long each trader takes to execute its *getorder()* or *respond()* function, the clock will always progress by the same amount. It also means that over the full course of a trading period, each trader will be polled for an order approximately once per simulated and will update their internal variables with *respond()* N times per simulated second.

2.4 Trading Algorithms

Although a highly detailed understanding and analysis of the trading algorithms used in this project is not necessary to understand the experiments and results of the project, it is important to have a grasp on the functionality of each algorithm, so that it can be understood how they may perform differently on an asynchronous system, compared to a synchronous system, such as BSE.

2.4.1 Giveaway

Giveaway is the simplest trading agent used in this project. Giveaway's *getorder()* function simply always "gives away" its deal by submitting an order at its limit price, assuming it currently has a customer order. Although this may appear to mean that it is impossible for a Giveaway trader to make a profit, this is not actually the case. By simply considering two traders, one buyer and one seller, where the buyer as a limit price of 120 and the seller has a limit price of 100, it can be seen that if the buyer submits their order first, followed by the seller submitting their order, the seller will cross the spread and the trade will be executed at the price of 120. This would result in the selling agent to make a profit of 20. However, if they were to submit their orders in the opposite sequence the trade would again be executed at the price of the first order submitted, which would be 100, therefore resulting in a profit of 20 for the buyer. Listing 2.2 shows the *getorder()* function for Giveaway.

```
def getorder(self, time, countdown, lob):
    if len(self.orders) < 1:
        order = None
    else:
        quoteprice = self.orders[0].price
        order = Order(self.tid,
                self.orders[0].otype,
                quoteprice,
                self.orders[0].qty,
                time, lob['QID'])
        self.lastquote=order
    return order
```

Listing 2.2: The *getorder()* function of the Giveaway trading algorithm.

2.4.2 Zero Intelligence - Constrained

ZIC is another very simply designed algorithmic trader, first described by Gode & Sunder (1993)[9]. When its *getorder()* function is called, if it currently has a customer order, it will generate a random order price, but constrained to only generate an order price which will not result in a loss for the trader,

therefore for an bid, the order price cannot be higher than the customer limit price, and for an ask the order price cannot be lower than the limit price. Listing 2.3 shows the *getorder()* function for ZIC.

```
def getorder(self, time, countdown, lob):
    if len(self.orders) < 1:
        order = None
    else:
        minprice = lob['bids']['worst']
        maxprice = lob['asks']['worst']
        qid = lob['QID']
        limit = self.orders[0].price
        otype = self.orders[0].otype
        if otype == 'Bid':
            quoteprice = random.randint(minprice, limit)
        else:
            quoteprice = random.randint(limit, maxprice)
        order = Order(self.tid, otype, quoteprice,
                      self.orders[0].qty, time, qid)
        self.lastquote = order
    return order
```

Listing 2.3: The *getorder()* function of the ZIC trading algorithm.

2.4.3 Shaver

Shaver is an algorithm which always tries to have the best bid on the LOB at any time, it does this by shaving a single point off the current best bid or ask currently on the LOB, however it as with the ZIC trader, will only do so if the resulting order will not result in a loss making trade. Listing 2.4 shows the *getorder()* function for Shaver.

2.4.4 Zero Intelligence Plus

The ZIP algorithm was first introduced in Cliff (1997)[4] and was one of the earliest algorithmic traders to make use of an adaptive algorithm, namely the Widrow-Hoff rule. This allows the trader to adapt the profit margin it is aiming for based on the conditions of the market, viewed through the LOB. A general form the Widrow-Hoff rule has been included in Equation 2.2

$$A(t+1) = A(t) + \Delta(t) \tag{2.2}$$

Where A is the *actual output* and Δ_t change in output, which itself is described in 2.3.

$$\Delta(t) = \beta(D(t) - A(t)) \tag{2.3}$$

Here, $D(t)$ is the desired output, and β represents the *learning rate coefficient* which controls the rate at which the *actual output* approaches the *desired output.*

This has been implemented into the ZIP algorithm. The ZIP algorithm functions by maintaining a variable which defines its desired profit margin, noted as *margin* which is used to select its order price p. This value is then adjusted based on the most recent activity on the market. If the last order posted is accepted, the trader will adjust its *margin* so that its p value moves towards the price at which the trade was executed. If an order posted but not accepted, then if the trader's current p is more ambitious (i.e. would generate a greater profit) than the price that was not accepted, it will reduce its *margin* value. This can be separately summarised for buyers and sellers by the pseudocode in Listing 2.5 taken from [4].

The *getorder()* function for ZIP has not been included as a listing, as much of its content is the same as in Giveaway and ZIC's *getorder()* functions, the most important difference being that for a trader i, with limit price λ_i, and desired profit margin μ_i, the order price p_i at time t is given by the equation:

$$p_i(t) = \lambda_i(1 + \mu_i(t)) \tag{2.4}$$

```
def getorder(self, time, countdown, lob):
    if len(self.orders) < 1:
        order = None
    else:
        limitprice = self.orders[0].price
        otype = self.orders[0].otype
        if otype == 'Bid':
            if lob['bids']['n'] > 0:
                quoteprice = lob['bids']['best'] + 1
                if quoteprice > limitprice :
                    quoteprice = limitprice
            else:
                quoteprice = lob['bids']['worst']
        else:
            if lob['asks']['n'] > 0:
                quoteprice = lob['asks']['best'] - 1
                if quoteprice < limitprice:
                    quoteprice = limitprice
            else:
                quoteprice = lob['asks']['worst']
        order = Order(self.tid, otype, quoteprice,
                      self.orders[0].qty, time, lob['QID'])
        self.lastquote = order
    return order
```

Listing 2.4: The *getorder()* function of the Shaver trading algorithm.

The original definition for the ZIP algorithm included another index j which represented the unit that trader i was operating on, but as both BSE and the extension described in this project only allow for a trader to trade a single unit at once, this has been omitted for simplicity.

The core of the ZIP algorithm is the calculation of the profit margin $\mu_i(t)$. For buyers, this variable is constrained by $\mu_i(t) \in [-1, 0]; \forall t$, meaning that $\mu_i(t)$ can generate order prices between 0 and λ_i, conversely for sellers it is constrained by $\mu_i(t) \in [0, \infty]; \forall t$, meaning $\mu_i(t)$ can be any value greater than λ_i. The profit margin $\mu_i(t)$ is calculated from Formula 2.5.

$$\mu_i(t+1) = (p_i(t) + \Delta_i(t))/\lambda_i - 1 \tag{2.5}$$

The variable $\Delta_i(t)$ comes from the Widrow-Hoff rule, which is adapted for ZIP to produce Formula 2.6.

$$\Delta_i(t) = \beta_i(\tau_i(t) - p_i(t)) \tag{2.6}$$

Here a new variable, the *target price* $\tau_i(t)$, is used in place of $D(a)$ the *desired output*. It would be easy to simply set $\tau_i(t)$ to the last *order price* $q(t)$ posted to the LOB, however this would cause an issue when the previous *order price* is similar to the traders current p_i value, as it would cause only minor or zero changes to be made to the new *order price*. This is an issue as in a competitive market traders should always be trying to test the limits of the market in order to maximise their profit. To model this $\tau_i(t)$ is produced via two random values, $\mathcal{R}_i$ and $\mathcal{A}_i$. $\mathcal{R}_i$ alters the *target price* τ_i relative to q and $\mathcal{A}_i$ is a small absolute alteration in the *order price*. $\mathcal{R}_i$ can either greater or smaller than 1, and $\mathcal{A}_i$ can be greater or smaller than 0, depending on whether the trader i is trying to raise or lower its *order price* p_i. This is shown in Formula 2.7

$$\tau_i(t) = \mathcal{R}_i(t) \cdot q(t) + \mathcal{A}_i(t) \tag{2.7}$$

This sets a *target price* $\tau_i(t)$ that then in turn sets $\Delta_i(t)$ and $\mu_i(t+1)$ causing the order price $p_i(t)$ to move towards $\tau_i(t)$.

```
For SELLERS:
    - if (the last shout was accepted at price q)
    - then
        1. any seller s_i for which p_i ≤ q should raise its profit margin
        2. if (the last shout was a bid)
            then
            1. any active seller s_i for which p_i ≥ q should lower its margin
    - else
        1. if (the last shout was an offer)
        then
            1. any active seller s_i for which p_i ≥ q should lower its margin
For BUYERS:
    - if (the last shout was accepted at price q)
    - then
        1. any buyer b_i for which p_i ≥ q should raise its profit margin
        2. if (the last shout was an offer)
            then
            1. any active buyer b_i for which p_i ≤ q should lower its margin
    - else
        1. if (the last shout was a bid)
        then
            1. any active buyer b_i for which p_i ≤ q should lower its margin
```

Listing 2.5: Pseudocode for the ZIP trading algorithm, as found in [4]

2.4.5 GD eXtended (GDX)

The GDX algorithm[14], introduced in Tesauro & Bredin (2002), builds on the previous work of Gjerstad & Dickhaut (1998)[8] who produced a trading algorithm that has become known simply as GD. GD was further modified by Tesauro & Das in their 2001 paper[15] which introduced the MGD algorithm.

Gjerstad-Dickhaut (GD)

GD was the first algorithm to make use of a "belief function", this is defined in Tesauro & Bredin's work as any function which takes a history of market activity H_M leading to the last M trades and uses this information to estimate the probability that a price p will be accepted.

For a buyer, their belief function is defined as:

$$f_b(p) = \frac{ABL(p) + AL(p)}{ABL(p) + AL(p) + UBG(p)}, \tag{2.8}$$

where,

ABL(p)	:	The number of Accepted Bids less than p
AL(p)	:	The total number of Asks Less than p
UBG(p)	:	The number of Unaccepted Bids Greater than p

For a seller, their belief function is defined as:

$$f_s(p) = \frac{AAG(p) + BG(p)}{AAG(p) + BG(p) + UAL(p)}, \tag{2.9}$$

where,

AAG(p)	:	The number of Accepted Asks Great than p
BG(p)	:	The total number of Bids Greater than p
UAL(p)	:	The number of Unaccepted Asks Less than p

The goal of the GD algorithm is then to find prices which will generate the greatest profit but are also likely to be accepted. This is done by maximising their expected profit, which is calculated by:

For buyers:

$$p* = \arg\max_p f_b(p)(l - p) \tag{2.10}$$

For sellers:

$$p* = \arg\max_p f_s(p)(p - l) \tag{2.11}$$

These values are calculated each time a trader makes an order to the exchange.

Modified GD (MGD)

MGD[15] modifies the GD algorithm in a number of ways, in order to address shortfalls and flaws that Tesauro & Das found with the algorithm. In GD when H_M contains rejected bids or asks, the algorithm will unrealistically expect any price to be accepted and therefore post very high asks or very low bids. They will then slowly lower or raising their orders until trades begin to take place, often in rapid succession, which again may cause H_M to no longer contain any rejections, once again causing GD to post unrealistic orders. To solve this MGD buyers store the highest successful trade price in the current and previous market session p_h, then sets $f(p) = 0$ for all $p \geq p_h$, and MGD sellers store the lowest successful trade price p_l, and set $f(p) = 0$ for all $p \leq p_l$. Tests using homogeneous populations of MGD traders were found to have much lower price volatility than similar tests using GD traders.

It was also found that the GD algorithm would make reasonable bids, that are expected to be accepted reasonably quickly. This was advantageous in purely homogeneous populations of GD traders, resulting in increased efficiency, however it exposed weakness when place in heterogeneous tests against other trading algorithms. To improve on this, MGD traders would improve on the current best ask or bid by a smaller margin, resulting in making MGD traders more reluctant to reduce their expected profit margin.

MGD also allowed traders who hold multiple tradeable assets to post bids for the least valuable unit, however as in this dissertation each trader is only allowed a single tradeable unit at a time, this is not relevant.

GDX

GDX[14] further expands on the GD algorithm by making use of Dynamic Programming. (H, T) is used to represent the current state of the market, where H is the event history and T is the trading time remaining. From T the trader calculates the number of future opportunities to make a bid/ask it will have before the end of the trading period, this is N and is calculated as $N = T/K$ where the trader expects to be able to post an order every K seconds. (H, T) is also used to calculate the belief function $f(p, \pm\overrightarrow{q}, t)$, which estimates the probability that a bundle of assets $\overrightarrow{q}$ ($+\overrightarrow{q}$ for buys and $-\overrightarrow{q}$ for sellers) will be successfully traded at price p with t time remaining in the trading period. In the experiments described in this dissertation, only one unit can be traded at a time $\overrightarrow{q}$ is simply +1 for buyers and -1 for sellers, so will be omitted for simplicity from the rest of this description. This simplification also means that the belief functions used throughout this project are identical to the functions described in the GD paper[8].

The next stage in the algorithm is to calculate $V(x, n)$, a table of predicted values, where x is the internal state of the trader and n is the number of bidding opportunities remaining. As the experiments used in this dissertation only allow a trader to have one bid or ask on the LOB, this is simply the assets being traded which are represented by M. The original GDX algorithm stored more information in this variable, but this is not necessary for this project. This is where the Dynamic Programming aspect of the algorithm is used. GDX starts by setting the terminal state $V(m, 0) = 0$, for all $m \in M$ and $V(0, n) = 0$ for all $n \in N$. From here $V(m, n)$ is calculated from $V(m, n-1)$ using Algorithm 2.1.

Algorithm 2.1: Expected value computation for GDX[14].

```
for n = 1 to N do
    for m = 1 to M do
        V(m,n) = max_p(f(p,t_n))[s(p) + γV(m−1,n−1)] + (1 − f(p,t_n))γV(m,n−1)
    end
end
```

γ is the discount parameter and $s(p)$ is the profit generated if the asset in question is traded at price p, for a buyer $s(p) = L - p$ and for a seller $s(p) = p - L$ where L is the trader's limit price. From the

experiments conducted in [14], it was found that as γ approaches 1, GDX's performance improves, but falls off at 0.99. Tesauro & Bredin set $\gamma = 0.9$ for their experiments, this is the same value for γ that is used throughout this project.

Finally, once the expected values table has been calculated the trader must choose the price it wished to post to the exchange. This calculation is show in Equation 2.12

$$p*(T) = \arg\max_{p}(f(p,T)[s(p) + \gamma V(M-1, N-1)] + (1 - f(p,T))\gamma V(M, N-1)) \tag{2.12}$$

Both the table of expected values and the optimal price are calculated each time GDX's *getorder()* function is called.

2.4.6 Adaptive-Aggressive (AA)

In Vytelingum's 2006 PhD thesis[18], a new type of algorithmic trader was described. This algorithm introduced the concept of aggressiveness. A trader's aggressiveness determines how it submits offer prices in relation to its estimate of the competitive equilibrium price p^*. Aggressiveness is held as a variable $r \in [-1, 1]$. An *aggressive* trader, where $r < 0$, will submit orders that are better than its estimate of p^*, which therefore will generate less profit, but are more likely to be traded. A trader is described as *active*, $r = 0$ if it submits offers at p^*. *Passive* traders, $r > 0$, instead post offers that would generate a greater profit than trading at p^*, although this results in the offers being less likely to be traded on the market. The "adaptive" part of the algorithm's name comes from the learning mechanism it uses to adapt the trader's aggressiveness according to the current market conditions.

Estimating Competitive Equilibrium p^*

The competitive equilibrium price of the market is impossible for a trader to know, therefore it is necessary for each AA trader to estimate the equilibrium, and then use this value throughout its computation. The equilibrium is estimated using a moving average, which calculates the average transaction price, from the latest transaction, T, to the N-th transaction before that. AA also uses a term known as *recency*, denoted by ρ, which gives greater weight to the most recent transactions, as it is assumed that the transaction prices will be converging towards the equilibrium, a high ρ value results in convergence patterns being emphasised. The calculation of p^* is shown in Equation 2.13

$$p^* = \frac{\sum_{i=T-N}^{T}(w_i \cdot p_i)}{N} \quad where, \sum_{i=T-N+1}^{T} w_i = 1, w_{i-1} = \rho w_i \tag{2.13}$$

Aggressiveness Model

The aggressiveness model is the part of the AA algorithm which generates a target price τ given the traders current level of aggressiveness, r. For this traders are divided into two categories, intra-marginal and extra-marginal. The difference between these two types of traders is best summarised in Vytelingum, Cliff & Jennings (2008) [17], "A buyer(seller) is intra-marginal if its limit price is higher (lower) than the competitive equilibrium price. In contrast, the extra-marginal buyer's (seller's) limit price is lower (higher) than the competitive equilibrium price." Extra-marginal traders are not expected to transact, and if the market is perfectly efficient, never will. The only way these traders can complete trades is to exploit orders that stray away from the equilibrium price p^*. This results in AA requiring a different approach for the different trader types. Equations 2.14 & 2.15 adapted from [17], describe how τ is calculated for intra-marginal traders, and Equations 2.16 & 2.17 show to equivalent calculation for extra-marginal traders.

For an intra-marginal buyer,

$$\tau = \begin{cases} p^*(1 - \frac{e^{-r\theta}-1}{e^{\theta}-1}), & \text{if } r \in (-1, 0) \\ p^* + (\ell_i - p^*)(\frac{e^{r\theta}-1}{e^{\theta}-1}), & \text{if } r \in (1, 0) \end{cases} \tag{2.14}$$

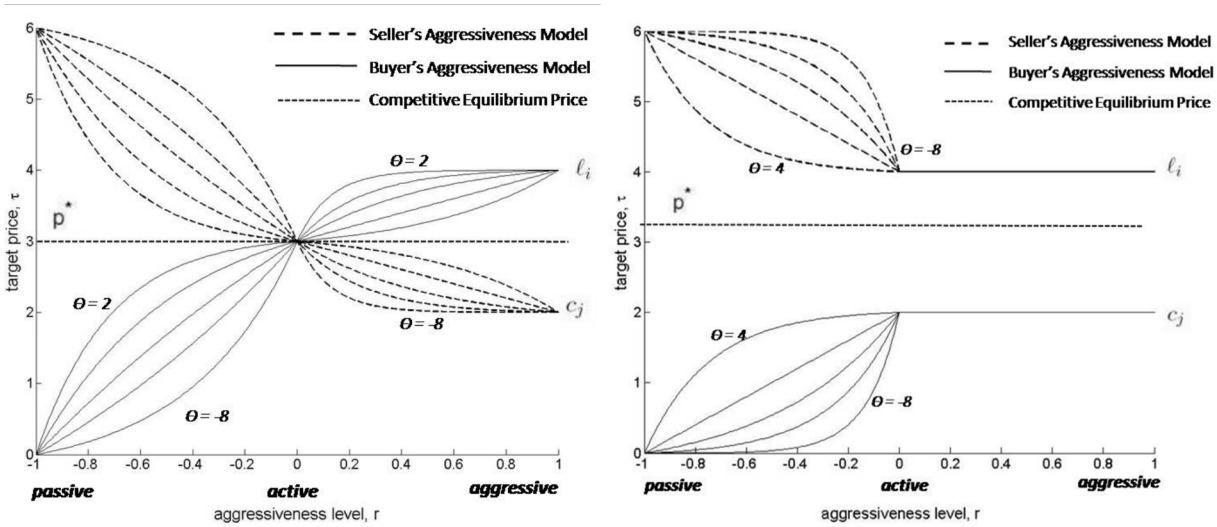


Figure 2.3: Aggressiveness of traders depending on θ, left graph is for inter-marginal traders and right graph is for extra-marginal traders.[17]

For an intra-marginal seller,

$$\tau = \begin{cases} p^* + (MAX - p^*)(\frac{e^{-r\theta}-1}{e^{\underline{\theta}}-1}), & \text{if } r \in (-1, 0) \\ c_j + (p^* - c_j)(\frac{e^{r\theta}-1}{e^{\theta}-1}), & \text{if } r \in (1, 0) \end{cases} \tag{2.15}$$

For an extra-marginal buyer,

$$\tau = \begin{cases} \ell_i(1 - \frac{e^{-r\theta}-1}{e^{\theta}-1}), & \text{if } r \in (-1, 0) \\ \ell_i, & \text{if } r \in (1, 0) \end{cases} \tag{2.16}$$

For an extra-marginal seller,

$$\tau = \begin{cases} c_j + (MAX - c_j)(\frac{e^{-r\theta}-1}{e^{\theta}-1}), & \text{if } r \in (-1, 0) \\ c_j, & \text{if } r \in (1, 0) \end{cases} \tag{2.17}$$

Where $\underline{\theta}$ is calculated such that the function is continuous as r = 0, that is there is no jump in the first derivative of τ. ℓ_i denotes a buyer i's limit price and c_j denotes a seller j's limit price. More about how θ and r is included in the Adaptive Layer.

Extra-marginal trader's cannot be aggressive as their limit price is already less profitable than p^*, hence when $r > 0$, τ is equal to their limit price. Figure 2.3 shows how the target price τ of a trader varies depending on its aggressiveness r and its long-term learning θ.

Adaptive Layer

The adaptive layer of the AA algorithm adjusts r via a short-term learning mechanism, and the same for θ via a long-term learning mechanism.

Short-Term Learning

This mechanism updates the aggressiveness r of the trader each time there is a new order placed onto the market, or when a trade occurs. This is done using the same Widrow-Hoff algorithm[11] as used in the ZIP trading algorithm. Here, r is adjusted towards $\delta(t)$, the *desired aggressiveness*, for a buyer $\delta(t)$ is the minimum of its limit price and a price slightly higher than the best outstanding bid, for a seller $\delta(t)$ is the maximum of its limit price a price slightly lower than the best outstanding ask. The implementation of the Widrow-Hoff algorithm that is use is shown in Equation 2.18

$$\begin{aligned} r(t+1) &= r(t) + \beta_1(\delta(t) - r(t)) \\ \delta(t) &= (1 \pm \lambda_r) r_{shout} \pm \lambda_a \end{aligned} \tag{2.18}$$

Here, r_{shout} is the value of r which would result in a bid or ask of the same price as the current best bid or ask, respectively. λ_r and λ_a are the relative and absolute change in r_{shout} and β_1 is the learning rate. The trader follows the rules in Listing 2.6 to determine whether to increase its aggressiveness (positive λ_r and λ_a) to increase the likelihood that it will trade, or decrease its aggressiveness (negative λ_r and λ_a) to increase its profits.

```
For BUYERS:
    - if (transaction occurs at price q) then
        if (τ ≥ q) then buyer must be less aggressive
        else buyers must be more aggressive
    - else if (bid, b, submitted)
        if (τ ≤ b) then buyer must be more aggressive
For SELLERS:
    - if (transaction occurs at price q) then
        if (τ ≤ q) then seller must be less aggressive
        else buyers must be more aggressive
    - else if (ask, a, submitted)
        if (τ ≥ a) then seller must be more aggressive
```

Listing 2.6: AA Short Term Learning Rules[17]

Long-Term Learning

Figure 2.3 shows how different values of θ results in different aggressiveness curves, this is done to better fit the market conditions and increase the traders efficiency. Which values for θ is best depends on the market volatility. Market volatility is measured by estimating *Smith's* α (see 2.20) which can only be estimated as the trader only has an estimate for p^*. Equation 2.19 describes the mechanism for adjusting θ.

$$\theta(t+1) = \theta(t) + \beta_2(\theta^*(\alpha) - \theta(t)) \tag{2.19}$$

$$\alpha = \frac{\sqrt{\frac{1}{N}\sum_{i=T-N+1}^{T}(p_i - p^*)^2}}{p^*} \tag{2.20}$$

β_2 is once again the learning rate. $\theta^*(\alpha)$ is a function that determines the optimum values for θ and is determined by Equation 2.21.

$$\theta^*(\alpha) = (\theta_{max} - \theta_{min})(1 - (\alpha - \alpha_{min})/(\alpha_{max} - \alpha_{min}))e^{\gamma((\alpha-\alpha_{min})/(\alpha_{max}-\alpha_{min})-1)} + \theta_{min} \tag{2.21}$$

Where $[\theta_{min}, \theta_{max}]$ is the range over which θ is updated, α_{max} is the maximum α that occurs in the market, and α_{min} is the minimum α. γ determines the shape of the function and originally was set to 2 in Vytelignum (2006)[18].

Bidding Layer

The bidding layer of the AA algorithm uses the value of τ given by r and θ to determine if it should submit a bid or ask, and if so, at what price. The rules to determine this are shown in 2.7.

$$bid_i = \begin{cases} o_{bid} + (\min\{\ell_i, o^+_{ask}\} - o_{bid})/\eta & \text{if first round} \\ o_{bid} + (\tau - o_{bid})/\eta, & \text{otherwise} \end{cases} \tag{2.22}$$

$$ask_j = \begin{cases} o_{ask} - (o_{ask} - \max\{c_j, o^-_{bid}\})/\eta & \text{if first round} \\ o_{ask} - (o_{ask} - \tau)/\eta, & \text{otherwise} \end{cases} \tag{2.23}$$

where,

$$o^{+}_{ask} = (1 + \lambda_r)o_{ask} + \lambda_a$$
$$o^{-}_{bid} = (1 - \lambda_r)o_{bid} - \lambda_a$$
$$\eta \in [1, \infty)$$

```
For BUYERS:
    - if (ℓ_i ≤ o_bid) then submit no bid
    - else
        if (first trading round) submit bid given by Equation 2.22
        else
            if (o_ask ≤ τ ) accept o_ask
            else submit bid given by Equation 2.22
For SELLERS:
    - if (c_j ≥ o_ask) then submit no ask
    - else
        if (first trading round) submit ask given by Equation 2.23
        else
            if (o_bid ≥ τ) accept o_bid
            else submit ask given by Equation 2.23
```

Listing 2.7: AA Bidding Rules[17]

Chapter 3

Project Execution

3.1 Threaded BSE (TBSE)

3.1.1 Motivation

Much of the functionality of TBSE was directly borrowed from the original Bristol Stock Exchange simulator. The key changes affect the way each market session operates. In BSE, each market session is executed on a single thread in which a core loop executes repeatedly until the session end-time is passed. Within this loop a single trader in the market's population of traders is selected at random, and that trader is polled to see if it wants to issue an order: this is achieved by calling that trader's *getorder()* function. If the trader does issue an order, that order is then processed by the exchange and all traders in the population are notified of any resultant change in the market-data published by the exchange. Once an order has been processed, each trader in the market is given an opportunity to respond to the new market conditions, by updating their internal variables based on the new information. This is done via a function called *respond()* that is particular to each specific trading algorithm. Some of the most simple algorithms investigated in this dissertation (Giveaway, ZIC, Shaver) do not have *respond()* functions, whereas the more complex algorithms (ZIP, GDX, AA) encode most of their detail within this function. Once all traders have called their *respond()* functions, the main loop restarts and another random trader is polled for an order. The simulated discreet time mechanism within BSE means that it takes a notional time of 1/N seconds for a trader to be polled for an order, and for all traders to respond to it, where N is the total number of traders in the market. This means that on average a trader will be polled for an order once per simulated second, and will respond to market changes via *respond()* N times per simulated second. But if Trader A's *respond()* takes 1ms of real-time to execute, and trader B's *respond()* takes 10ms of real-time to compute an answer, BSE's record of simulated time is the same in both cases: the simulated clock progresses in increments of 1/N seconds, regardless of the real execution time needed for the different *respond()* functions. The problem with this style of simulation is that each trader is allotted as much time as it needs to execute its *getorder()* and *respond()* functions, and it is guaranteed that nothing else in the market will change while it executes either of these functions. This means that the execution time of each trader has no impact on its performance: instead all that counts is its ability to generate an order price that will be accepted by another trader, whilst attempting to generate the greatest profit. This differs from a real financial exchange where all traders operating on the market will be operating asynchronously. In an asynchronous market, a trader may look at the exchange's market data, i.e. the currently available information on that market, and use this to calculate what it considers its best order to send to the exchange. However, if this trader uses a complex, slow running algorithm to do so, it may find that by the time it has completed its calculation, another simpler and faster trader has already succeeded in executing a trade, which as a result changes the position of the market which may render the order that the complex trader has posted less profitable than expected. This situation, which happens all the time in real-world financial markets, cannot be modelled in BSE.

3.1.2 TBSE Operation

In order to address the inadequacies of single threaded simulations, an asynchronous multi-threaded exchange simulator, the Threaded BSE (TBSE), was created. The original BSE was written in the Python programming language so it was logical to continue to use Python when developing TBSE. In TBSE each trader executes on its own computational thread, as does the exchange. There is also the

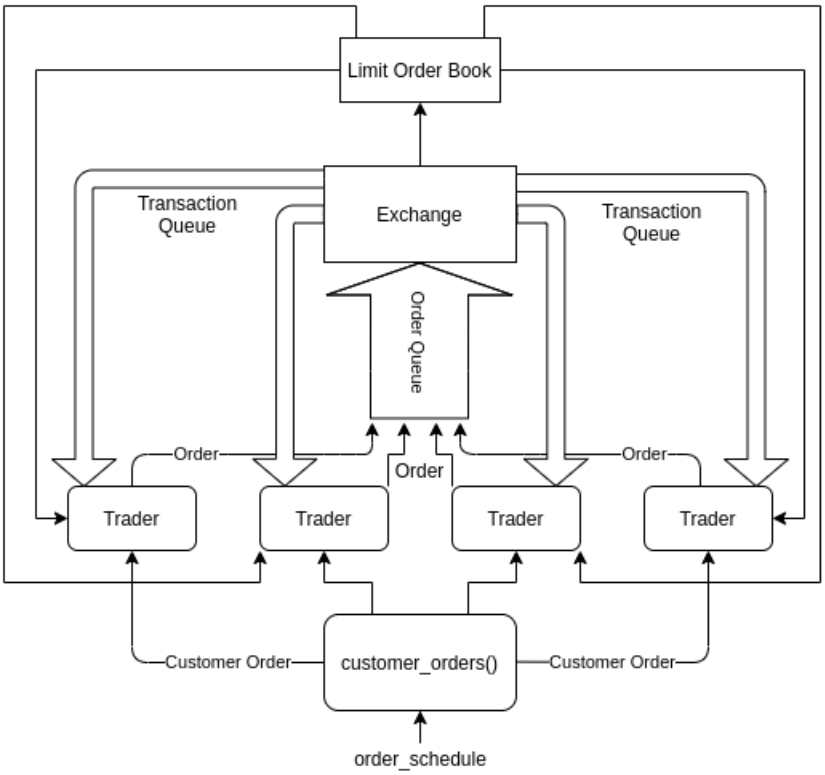


Figure 3.1: System architecture diagram for TBSE with four traders in the market.

main thread of the Python code which continues to run during the market session and is responsible for the distribution of customer orders to each of the traders. Figure 3.1 shows how these components interact with each other.

Traders

Each trader thread consists of a loop which executes continuously until the end of the market session. Within this loop the trader first receives information of trades which have been successfully executed at the exchange. It then updates its own internal record-keeping if any of these trades involved itself via the *bookkeep()* function, this tracks the profit that the trader has generated, the total of number of trades that the trader has successfully executed and finishes by deleting the customer order that has now been fulfilled. Each trader then executes its *respond()* function to update its internal variables based on the new market data that has been produced as a result of the trades which have been executed. Once it is updated with the latest market data it then executes its *getorder()* function which determines whether it should post a new order to the exchange, and if so at what price. This order is then placed on the *order_queue* to be sent to the exchange. A simplification included in BSE which is carried over into TBSE is that if a trader places a new order onto the exchange but already has an order on the exchange, then the new order replaces the old one, therefore each trader can have at most one order on the exchange at a time. A simplified version of a trader's main function is included in Algorithm 3.1 as pseudocode.

The *getorder()* and *respond()* functions are specific to each type of trader, details of which can be found in Section 2.4. Details of the queuing mechanism can be found at Section 3.1.4.

Exchange

The exchange also operates on the *order_queue*, reading orders from the queue and then processing them by either adding them to the its list of available orders on the LOB or executing a trade if the new order can be matched with any of its current list of available orders. If a trade can be executed it then places the details of the resultant transaction onto a queue for each trader to read before submitting its next order. In Figure 3.1 this queue is displayed as the "Transaction Queue". This does not replace the *tape* mentioned in Section 2.1.2, but is simply a method for communication with traders whose orders have been matched. A simplified pseudocode for the operation of the exchange is included as Algorithm 3.2.

Algorithm 3.1: Simplified operation of a Trader process in TBSE.

```
while time < end_time do
    if len(trader.queue) != 0 then
        trade = trader.queue.get()
        if trader ∈ trade.traders then
            trader.bookkeep(trade)
        end
        trader.respond(LOB, trade)
    end
    trader.respond(LOB, None)
    order = trader.getorder(LOB)
    if order != None then
        exchange.order_queue.put(order)
    end
end
```

Algorithm 3.2: Simplified operation of the Exchange process in TBSE.

```
while time < end_time do
    if len(order_queue) != 0 then
        order = exchange.order_queue.get()
        trade = exchange.process_order(order)
        if trade != None then
            for trader ∈ exchange.Traders do
                trader.queue.put(trade)
            end
        end
    end
end
```

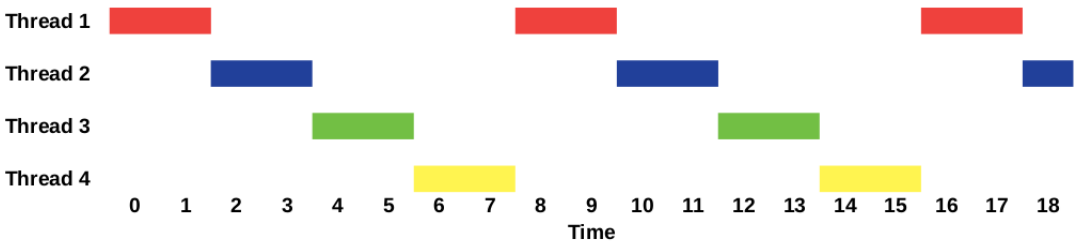


Figure 3.2: Basic diagram showing how multiple threads run concurrently.

The *process_order()* function operates much the same as in BSE and details of it can be found in Section 2.3.4.

3.1.3 Customer Orders

Customer orders are distributed on a separate thread to the traders and the exchange. This thread runs the *customer_orders()* function in a loop, repeating until the end of the session. *customer_orders()* reads in the *order_schedule*, which describes the supply and demand curves of the market, and how limit prices are issued to traders, details of order schedules can be found at Section 2.3.2. If the order schedule has a *interval* less than the length of the market session, multiple customer orders will be issued to traders over the course of the session. When a new customer order is generated for a trader, if the trader has not successfully executed its first customer order, the first customer order is deemed not to be tradeable (most likely because the limit price differs too greatly from the competitive equilibrium price). If this occurs it is likely that the trader has an order on the exchange which relates to the old customer order, therefore the trader must send a cancellation order to the exchange so that this order is removed. Once this has been done, the trader is free to begin acting on its new customer order. This differs from BSE's approach to cancelling orders, where the *customer_orders()* function is called on every iteration of the core loop, before a trader is polled for an order, and cancellation orders are completed immediately. This means that if new customer orders are distributed to multiple traders at the same time, for example, when the *order_schedule*'s *timemode* is periodic, all cancellation orders will appear to execute simultaneously.

3.1.4 Python Features

Multi-threaded Programming

In Python, as with other languages that offer such a feature, multi-threading is where a processor can operate multiple threads of execution concurrently. This differs from multi-processing, which Python also supports, where different processes execute simultaneously. Both are forms of parallelism, but whereas in multi-processor programming multiple processes are executing at the same time on different physical processors, in multi-threading only one thread can execute at any one time. The execution of each thread is divided into short segments and the processor regularly switches between each thread for a short period of time, giving the appearance that all threads are executing at the same time. A diagram showing a simplified representation of how the threads share CPU time is shown in Figure 3.2. It is not guaranteed that each thread will be given the exact same amount of execution time during each cycle, but over the course of executing an entire program, which may involve millions of cycles, the execution time given to each thread should tend towards the same value. In TBSE, this means that each trader is given the same amount of processing time, so a faster algorithm can complete its execution and start processing a second order while a slower algorithm is still calculating its first order price. In Python, the execution of threads is synchronised by a Global Interpreter Lock (GIL) which ensures that only one thread is executing at a time. Multi-threading was chosen over multi-processing as most general-purpose CPUs have no more than 8 processing cores, but for the experiments in this project many more than than 8 traders will be operating simultaneously.

Queues

The queues used within TBSE are synchronous first in, first out (FIFO) queues, meaning that the trader who places an order on the exchange queue first will have their order processed first. This is a good

simulation of real exchange behaviour as it is commonplace for exchanges to prioritise orders by their time of arrival.

3.2 Verification

There are two parts to this project which require some form of verification. Firstly, the exchange portion of TBSE should be verified to show that it correctly matches orders and executes trades at the correct prices, as in BSE. Once TBSE has been shown to correctly run an exchange based on incoming orders, it is necessary to verify that the trading algorithms being used still operate in much the same way as on a single threaded simulation, such as BSE.

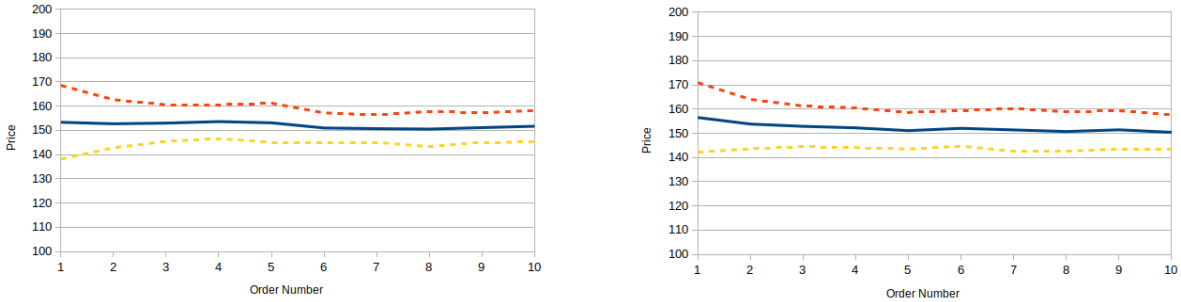
3.2.1 TBSE

In order to verify that TBSE was correctly processing and matching orders to execute trades, the original BSE code was used to generate a list of orders and the trades that those orders generated. The same list of orders was then fed into TBSE to see if the same trades were generated as a result. For these tests a setup was chosen where each trader was given a single customer order to either buy or sell an asset at the start of the market session. No new customer orders were issued throughout the market session because this requires traders to cancel orders they have submitted to the exchange for an old customer orders. TBSE and BSE handle this process differently, with BSE executing cancel orders immediately whereas TBSE requires cancel orders to be submitted in the same way as a buy or sell order, which is a more realistic simulation of the cancellation process on a financial exchange (see 3.1.3 for details). The difference between these two approaches can cause the exchange to produce different results, so has been excluded from these tests. As BSE is known to be a well functioning exchange simulator, it can be said that if TBSE processes the same set of orders to produce the same set of trades as BSE, under the conditions described above, TBSE must also be functioning as an accurate simulation of an exchange. This process was repeated over multiple sets of trading data, using different configurations of BSE to produce the orders, in order to thoroughly test that for all sets of input, both BSE and TBSE produce the same output. It was found that TBSE did accurately recreate the exchange modelled by BSE.

As it was not possible to verify the how cancellation orders operate in TBSE by direct comparison BSE, an additional test was conducted on TBSE, that verified that when a new customer order was generated for a trader, if that trader had not managed to fulfil the old customer order, a maximum of one new buy or sell order is submitted to the exchange before a cancellation order. The reason for allowing one further order after the new customer order is generated is, because the customer order is generated asynchronously to the trader's operation, the trader may be in the process of generating an order when the new customer order is received. The test then verified that the order following the cancellation order relates to the new customer order, this was achieved by tagging the buy/ask orders submitted to the exchange with a unique ID relating to the customer order.

3.2.2 Trading Algorithms

In previous publications implementations of trading algorithms have been verified by comparing the performance of the implementation to the performance described in the paper in which the algorithm was originally described. This is not an appropriate method of verification for these algorithms within the context of this project, as the project expects the performance of these traders to be altered by being run on TBSE. This creates the problem of how to verify that these algorithms are still true implementations of how they were originally described by their developers. To do this, minimal changes were made to the algorithm's implementations which have already been verified to function correctly on BSE, this is discussed in [13]. Instead of comparing the performance of the different algorithms competing against each other, tests were ran where the market was homogeneously populated with traders of a single type and each algorithm's ability to find the competitive equilibrium price on both BSE and TBSE was compared. If an homogeneous market of traders find the equilibrium price quickly, that is sufficient to conclude the algorithms are functioning correctly on TBSE. It should be noted that Snashall & Cliff were unable to perfectly recreate the results for AA and GDX first published by Vytelingum and Tesauro & Bredin respectively, however the difference in their results was found to be statistically insignificant. In this project, the only changes that were made were to ensure than the correct customer orders were being acted upon, and that no order was accidentally fulfilled twice due to the asynchronous nature of TBSE. A bug found in Snashall & Cliff's GDX implementation was also fixed, discussed in the subsection 'GDX'.



(a) BSE (b) TBSE

Figure 3.3: Transaction prices of ZIP algorithms in a homogeneous market. The blue line represents the mean transaction price and the dotted lines above and below represent the mean plus one standard deviation and minus one standard deviation, respectively.

Simple Algorithms

The three most simple algorithms looked at in this dissertation, namely Giveaway, ZIC and Shaver do not require any formal verification, as they do not use any information about the market in their calculation (other than Shaver requiring to know the best current bid or ask). Giveaway and ZIC are so simple, that they would act identically regardless of the market they were in, or the simulated exchange they were acting on, only being guided by their customer's limit price. Shaver is also very simple, but does require knowledge of the best ask or bid on the market, this should be provided by the simulated exchange given that it is operating correctly, so it can be said the verification of Shaver follows from the verification of TBSE.

ZIP

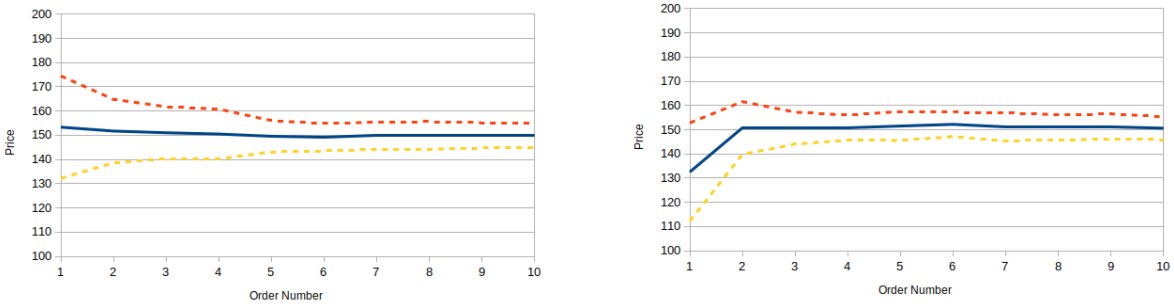
In order to verify that the ZIP algorithm operated correctly on TBSE, homogeneous tests were ran to observe how the transaction prices for traders between ZIP traders approached the theoretical equilibrium price. A symmetric order schedule with theoretical equilibrium price P_0 of 150 was used. During each market session 10 customer orders were issued to traders at regular intervals. Figure 3.3 shows the mean transaction prices averaged over 50 trials. The dashed lines represents the mean plus and minus one standard deviation.

On both BSE and TBSE the transaction prices very swiftly approach P_0 and the standard deviation from P_0 reduces over successive customer orders. There is a slightly higher deviation on TBSE, which is likely caused by traders in TBSE having less complete information about the activity on the market whilst it is performing its calculations. The difference is however small enough for it to be reasonable to state that the ZIP algorithm does indeed function correctly on TBSE.

GDX

It should be noted that the version of GDX used in this project is very slightly altered from the original version used in Snashall & Cliff (2019)[13], which is described in Section 2.4.5. The original version was found to have a bug which resulted in GDX traders not submitting a single order when placed into a homogeneous market. This bug was as a result of GDX traders not placing an order on their first turn and instead using their first turn to populate their array of expected values. This is performed on the first execution of their *respond()* function, however BSE only calls the trader's *respond()* function after an order has been received onto the market so in a homogeneous market all the GDX traders are waiting for an order to be placed onto the market before they start submitting orders. This was easily fixed by having GDX traders populate their expected value arrays on the first call of *getorder()* rather than *respond()*.

As with the ZIP algorithm, once again 50 homogeneous market sessions were ran, with 10 customers orders for each trader and supply and demand curves resulting in a P_0 values of 150. The results from these tests can be found in Figure 3.4. Here, the GDX traders on BSE almost immediately determine P_0, with the standard deviation starting high at 21.1, but then rapidly reducing to less than 5 by the end of the session. The TBSE results shows a very similar pattern for the standard deviation, but the mean



(a) BSE (b) TBSE

Figure 3.4: Mean transaction prices of GDX algorithms in a homogeneous market.

transaction price of the first set of customer orders is 17.5 away from P_0, however it rapidly corrects this error by the next set of customer orders. The cause of this, and the difference in result between BSE and TBSE, is that on BSE only the first few traders polled for an order have little or no information. Once a few orders have been placed, and every trader has been able to respond to them, the last traders in the market have a great deal of information with which to form accurate estimates of P_0. On TBSE however, all traders will simultaneously start calculating their first order, without any knowledge of the other traders or information on the LOB, therefore making far less accurate estimates. By the time the second set of customer orders is issued however, all traders will have obtained enough information the market to make accurate predictions of P_0. It is believed that these results show GDX to operate correctly, if not optimally, on TBSE.

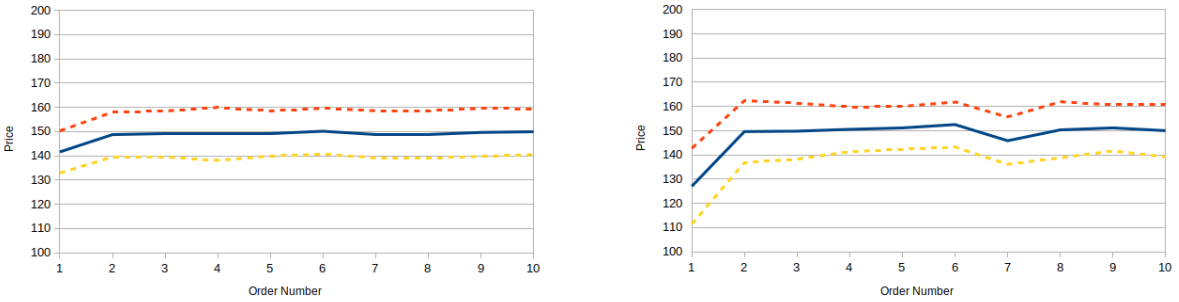
AA

The AA algorithm was tested under the same conditions as described above. Once again, on TBSE the mean transaction price relating to the first customer orders issued was much further from P_0 than on BSE, this can once again be explained by the lack of information that TBSE traders have at the start of a market session. As with GDX, the AA traders swiftly find the equilibrium price. At the start of the market session, on both BSE and TBSE, the standard deviation is far less than with GDX, however unlike the GDX it does not reduce quickly over time. The explanation for this is likely due to the details of the AA algorithm, where traders adapt to be more passive or aggressive. More passive traders, who post more profitable order prices, which therefore are less likely to be accepted, may be being matched with more aggressive traders on the other side of the market, who post conservative orders hoping to guarantee a trade, resulting in greater variation around the mean.

The AA algorithm used in this experiment was taken from Snashall & Cliff (2019)[13], in which it is acknowledge is the implementation is not a perfect recreation from the original Vytelingum (2006)[18] paper, due to details of certain equations missing from the original paper. It is stated however that the implementation produces results which are very close to those in Vytelingum's paper, and the difference between them is insignificant. As this algorithm, run on both BSE and TBSE, produces similar results, other than relating to the first set of customer orders (as explained above), once again this shows with confidence that the TBSE implementation work as expected.

3.3 Experimental Design

The key focus of this project is to identify whether there is significant differences in how a variety of popular trading algorithms perform when running on a single-threaded exchange simulator or a multi-threaded exchange simulator. It was therefore logical to run identical experiments on BSE and TBSE and analyse how their results differ. Both BSE and TBSE share the same system for specifying the market conditions for experiments, which are controlled via the *trader_spec* and the *order_schedule*, details of which can be found in the next two subsections.



(a) BSE (b) TBSE

Figure 3.5: Mean transaction prices of AA algorithms in a homogeneous market.

3.3.1 Trader Specification

In previous work (such as the original MGD[15] and GDX[14] papers) the two most popular methods of comparing two algorithms against each other have been; balanced group tests, where each algorithm is assigned the same number of buyers and sellers, and 1-in-many tests where all traders in the market run the same algorithm except for one which runs the opposing algorithm. Balanced group tests are considered the fairest way of comparing two algorithms and 1-in-many are thought to be a good way of testing how robust a homogeneous population of a single algorithm is to a single trader defecting to another strategy. Simply using these two methods to compare a pair of algorithms has been criticised due to neither scenario being realistic.

In Snashall & Cliff (2019)[13] it was observed that the performance of an algorithm can vary greatly depending on the ratio of different traders in the market. In their experiments they ran four algorithms against each other, with 16 buyers and 16 sellers. In order to test these algorithms exhaustively, they tested every single possible permutation of trader ratios and then averaged their results for all ratios. As this project intended to compare six different trading algorithms the number of different experiments that would need to be ran would be far too great, requiring a vast amount of CPU time. As the goal of this research is not to exhaustively compare these algorithms, but instead see how the performance of the algorithms changes when ran on TBSE instead of BSE, it was decided that it would be sufficient to run pair-wise comparisons, simply testing two algorithms against each other at a time.

Each experiment consists of 20 buyers and 20 sellers, a total of 40 traders was chosen as this was found to be near the limit of how many threads could operate simultaneously without the system becoming prone to failure. It was decided that the buyer and seller schedules should be symmetric, so the ratio A:B of two algorithms was the same on the buyer side of the market, and on the seller side. Although it was deemed too costly to do a full comparison off all six algorithms in all possible configurations, it was still desirable to design a set of experiments which takes into account the fact that the ratio of the two different algorithms involved in the tests has a great effect on their performance. Therefore, on one side of the market, the ratio of Algorithm A to Algorithm B would be N:(20-N) with N ranging from 1 to 19, with the same being true on the other side of the market. In the currently published literature on automated trading algorithms, there is no other examples of this kind of experiment where full analysis of how two traders perform against each other across a full sweep of ratios. Snashall & Cliff (2019)[13] will have performed many of these experiments during their exhaustive four-way comparisons of AA, ASAD (not discussed in this project), GDX and ZIP and AA, ASAD, GDX and ZIC however as their results were averaged across all ratios, there is no analysis of how the performance of two algorithms changes as the ratio of A to B changes.

In summary, each pair of the six algorithms (15 pairs) will be tested on 19 different ratios of traders from 1:19 to 19:1, totalling 285 different experiments.

3.3.2 Order Schedules

In previous work exploring trading agents, such as Tesauro & Das (2001)[15], it has been common practice to draw limit prices from a fixed uniform random distribution. When Vytelingum [18] first tested his AA algorithm, he chose four predetermined sets of supply and demand curves on which to run his experiments, this method was then replicated by Snashall & Cliff in their 2019 paper[13]. In real-world markets, the

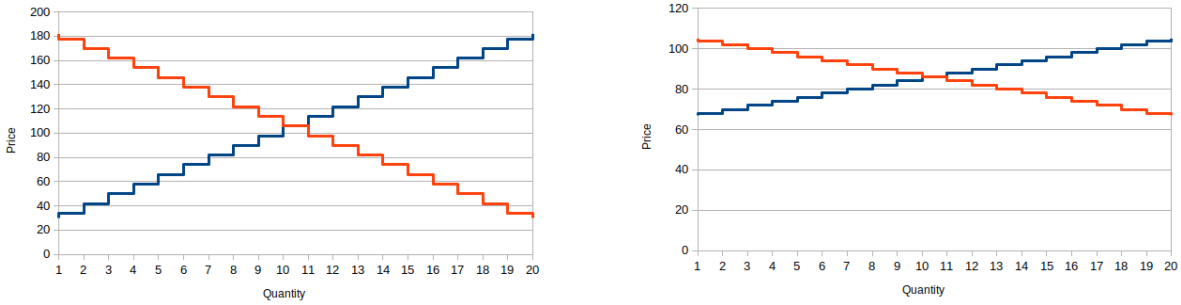


Figure 3.6: Example supply and demand curves.

true underlying supply and demand curves can never be known, therefore, this method was not repeated in case certain schedules give an unknown advantage to one of the algorithms. It was determined that the fairest way to test the algorithms would be to randomly generate multiple schedules and then average the results across them.

Each of the 285 experiments consisted of 10 randomly generated supply and demand curves on which 100 individual market sessions were conducted. The supply and demand curves are generated by producing a random maximum value between 100 and 200 and a random minimum value value between 1 and 100, then limit prices are evenly distributed between the maximum and minimum values (*stepmode = 'fixed'*). A simplification included in the order schedules for the experiments ran in this project is that the maximum and minimum values are the same for both the demand and the supply schedules, resulting is symmetric supply and demand curves. This was done to allow for more accurate analysis of the results, details of the reasoning behind this can be found in Section 4.1. Two example supply and demand curves are included in Figure 3.6.

The *timemode* variable within the order schedule was set to periodic, meaning new customer orders are distributed to traders at the same time. The *interval* for the order schedule was set to 30. For both the experiments ran on TBSE and BSE the time given is 600, meaning that 19 customer orders are given to each trader throughout a single market session (due to the way the *periodic* timemode is designed the first customer orders are distributed at time 30). Full detail of the order schedule for the experiments ran in this project can be found in Listing 3.1.

```
range_max = random.randint(100,200)
range_min = random.randint(1, 100)
rangeS = (range_min, range_max, offset_fn)
supply_schedule = [ {'from':0, 'to':600, 'ranges':[rangeS],
                     'stepmode':'fixed'}]

rangeD = (range_min, range_max, offset_fn)
demand_schedule = [ {'from':0, 'to':600, 'ranges':[rangeD],
                     'stepmode':'fixed'}]

order_schedule = {'sup':supply_schedule, 'dem':demand_schedule,
                  'interval':30, 'timemode':'periodic'}
```

Listing 3.1: Python code for the *order_schedule* in both TBSE and BSE tests

The first set of experiments performed used static markets, this is where the supply and demand schedules are fixed throughout the experiment. This means that the *offset_fn* function, seen in the tuples *rangeS* and *rangeD*, was not included, and the schedules remain the same throughout the duration of the experiment.

For the second set of experiments that were performed, an offset function was used that adds a varying offset to the competitive equilibrium price over time. It is a sinusoidal wave function that increases in amplitude and frequency over time. This function was taken from the original release of BSE, its python code is included in Listing 3.2 and a graph showing how the offset changes over time is included as Figure 3.7. This experiment was designed to introduce the traders to a more complex market dynamic and force them to adapt to a moving equilibrium price.

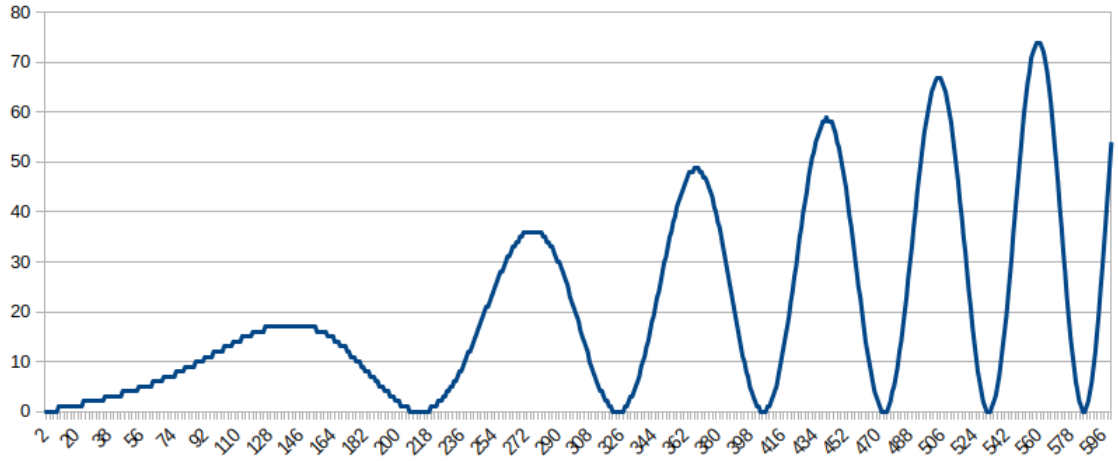


Figure 3.7: Offset function used for second set of experiments.

```
def schedule_offsetfn(t):
    pi2 = math.pi * 2
    c = math.pi * 3000
    wavelength = t / c
    gradient = 100 * t / (c / pi2)
    amplitude = 100 * t / (c / pi2)
    offset = gradient + amplitude * math.sin(wavelength * t)
    return int(round(offset, 0))
```

Listing 3.2: Python code for the offset function.

3.3.3 Measuring Performance

Section 2.2.2 discusses the various ways in which the performance of trading algorithms has been compared in previous research. *Profit Dispersion* (PD), which is defined as a trader's profit divided by the profit the trader would have made if every trade was executed at the competitive equilibrium price P_0, has commonly been thought to be the best way to measure a trader's performance because it not only considers the profit of the trader, but also how well the trader capitalised on the available surplus in the market. In the experiments for this project however, it was decided to simply use the profit that each trader generated. This decision was made because PD requires the calculation of P_0. This is a simple calculation in experiments of the style conducted by Vernon Smith in his 1962 paper[12] where P_0 is known in advance, however if P_0 is variable over time, it would require that P_0 be recalculate after every event on the market, requiring knowledge of every limit price currently being used by traders. This poses a technical problem, in that it would require significant additional calculation and that an individual trader does not have access to the limit prices of other traders. Additionally, PD ignores the number of customer orders that a trader manages to complete; if a trader only completes one trade out of many customer orders, but generates a good profit on that single trade, it would have a greater PD than a trader that completes multiple customer orders but at a lower profit margin, even though it may have generated far more profit in total. In a real world scenario it is normally only the profit that is generated from an algorithm that is important to traders.

3.3.4 Amazon Web Services

As discussed in the previous sections each style of experiment requires 285 different combinations of traders and ratios of those traders. Each of these combinations is tested on 1000 market sessions, resulting in a total of 285,000 market sessions per experiment, per simulated exchange. So with the two styles of experiments being run on both TBSE and BSE, well over 1 million market sessions are required. If each experiment took a single second to run, it would take a single machine over 280 hours to run all these experiments, and it should be noted that the run time for BSE experiments with large numbers of traders with long execution times can be upwards of a minute, meaning it is not unrealistic to think the

total computation time require could be more than 10,000 hours (over 1 year). Clearly, this is far too great a quantity of computation to be executed on a single machine, therefore it was decided that the experiments should be run on Amazon Web Services (AWS).

AWS's Elastic Compute Cloud (EC2) allows virtual computers to be rented on demand, supplying an operating system on which software can be installed and executed. As the execution of TBSE and BSE experiments do not require vast quantities of RAM or specialised hardware, simple 't2.micro' instances, which contain 1 vCPU and 1GB of RAM, were sufficient to run the experiments. By creating dozens of instances, all running different tests, the experiments could all be completed in just a couple of weeks.

Chapter 4

Critical Evaluation

4.1 Equal-Availability Ratio (EAR)

This project describes experiments where two types of trading algorithms are compared head to head, by running tests involving different ratios of traders running Algorithm A against traders running Algorithm B. For each ratio, the algorithm which produces the greatest profit per trader is then declared the winner of the test. In previous work, these tests have been repeated many times and the algorithm which wins the most of these tests has been declared the winner. At first glance this seems like a logical approach, however, if you consider an ideal scenario where both algorithms A and B are running the same perfect algorithm that always results in trades executing at the competitive equilibrium price P_0, it becomes interesting to ask which group of traders, A or B, would win at each ratio. At 1:1 it is correct to assume that each group would win an equal amount of tests, but what happens in the extreme situation when there are N traders in total, and A has only 1 trader and B has $N-1$ traders? For simplicity only one side of the market will be consider here, the buyers and their demand curve, and N will be set to equal 5, so there is one trader of type A and four of type B. Consider a demand curve where $P_0 = 150$ and there are five limit prices, one for each trader, spread equally between 50 and 250, i.e. 50, 100, 150, 200, 250. It will be assumed that the supply curve in this test is symmetric to the demand, hence $P_0 = 150$. These limit prices are randomly assigned to the traders, so if 1000 tests are ran in an ideal situation, Trader A should get each limit price 200 times, it will be assumed that this is the case. Consider each case in turn, starting from the highest limit price. If Trader A has limit price of 250, and its trade executes at P_0, it will generate a profit of 100. As there is only one trader of type A, the average profit is also 100. In this case the four traders of type B will be assigned the other four limit prices, 200, 150, 100, and 50. The first, with limit price 200 will generate a profit of 50 when it trades at P_0, the trader with limit price 150, equal to P_0 will trade but not gain a profit, and the other two traders have limit prices lower than P_0, so will not be able to trade. So the total profit made by group B is 50, making their average profit $50/4 = 12.5$. As the average profit of Trader A is greater, Trader A will win all 200 tests where it is assigned a limit price of 250. Next, on 200 occasions Trader A will get limit price 200, and therefore will produce a profit of 50 each time. The other four traders will be assigned the remaining limit prices, but only the trader assigned the limit price of 250 will gain a profit, which will be 100. Therefore when Trader A has limit price of 200, the group B will have an average profit of 25. Once again Trader A will win all 200 of these experiments. When assigned the remaining three limit prices, 150, 100 and 50, Trader A will not generate any profit, but group B will always get assigned the two profit generating limit prices, 250 and 200, producing an average profit of 37.5 and winning the remaining 600 tests. So in summary, when group A has a single trader and group B has the remaining four traders, group A will win 400 times and group B will win 600 times, in this ideal scenario. This means that when testing different algorithms in this same way, for Algorithm B to be declared the winner over Algorithm A, it is not sufficient for Algorithm B to win half the tests, in fact it would need to win more than 60% of the tests. In this dissertation the *Equal-Availability Ratio* (EAR) is defined as the difference in wins between group A and group B divided by the total number of tests, when ran under the conditions described above. So for the above example the EAR would be define as:

$$EAR = \frac{400 - 600}{1000} = -0.2 \tag{4.1}$$

The EAR changes depending on the number of traders in the market, the ratio of A:B and the supply and demand curves of the market. Figure 4.1 shows the EAR (expressed a decimal) for a range of N

values across their full spread of ratios, given the supply and demand curves described above.

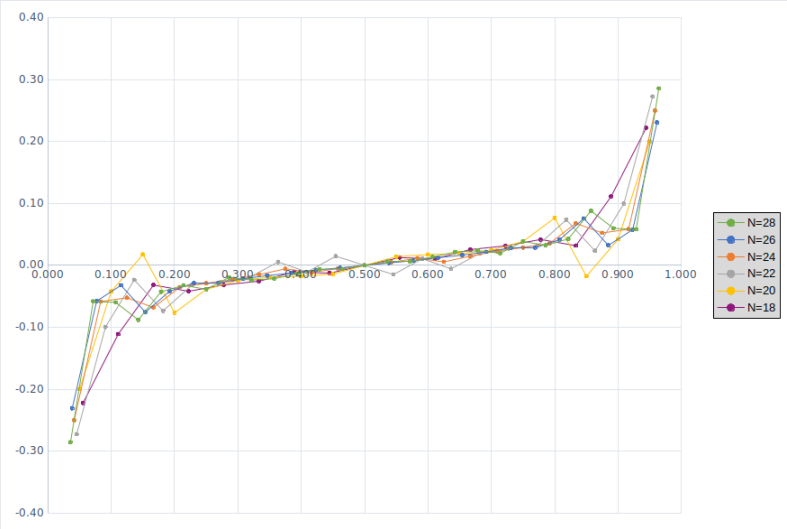


Figure 4.1: Graph showing the EAR at all ratios for different values of N.

As Figure 4.1 shows, for most ratios of traders the EAR is less than ±0.1, which shows that if perfect algorithms were possible and in use, the ratio of group A to group B normally has only a small affect on the outcome. The graph shows that in the majority of cases, the trader in the majority has an advantage, i.e. by default would win more than half the tests. The interesting part of Figure 4.1 is when there is a single member of either group A or group B on each side of the market. For the example values of N given, the EAR is greater than or equal to ±0.2 in all cases. If the EAR is 0.2, means that when a single trader of group A is compared to many traders of group B, over 1000 tests, it should be expected that group B will win 200 more tests than group A. As a result of this, algorithms that have been said to operate poorly in 1-in-Many style experiments, may have been somewhat misjudged, as it should be expected, simply by the nature of the distribution of limit prices, that the single trader performs worse than its many opponents. The noise towards either end of the lines would appear to be a result of quantization effects and also because of how many fewer trails are calculated at the ends of the lines than in the middle, due to the number of permutations of the two algorithms are far less when the groups are uneven.

In the experiments that were performed for this project, described in Section 3.3, there are always 20 traders on both the supply and demand curves, therefore it is sensible to compare the results produced by these experiments to the EAR difference graph for N=20. However, in Figure 4.1 all the lines drawn, including the one for N=20, are for the specific demand and supply curves where there are 20 traders on each curve, equally spread between 50 and 250, resulting in a P_0 of 150. As the shape of the supply and demand curves affects the resulting shape of the EAR difference graph, it would not be appropriate to compare the results of this project to the N=20 line from Figure 4.1. The optimum solution would be to record the details of each order schedule used for each experiment and produce an average EAR difference graph for each combination of trading algorithms. Unfortunately, when the experiments were ran, these details were not recorded. The solution to this problem was to produce 100 random supply and demand curves, in the same way as used in the experiments (see Listing 3.1), and then calculate the mean EAR difference at each ratio across all 100 curves, then use the resulting graph, seen in Figure 4.2, to compare to the results of the experiments. As the 10 supply and demand curves used in the actual experiments are produced from the same random distribution as the set of 100 curves used to produce Figure 4.2, it is safe to assume that the averages of both resulting sets of EAR differences are likely to be very similar. Even if there is a variation, in reality, the difference between different EAR curves for the same N is very small in comparison to the difference between the performance of actual algorithms when compared to each other, so a minor error in the EAR is unlikely to ever cause a significant change in the results.

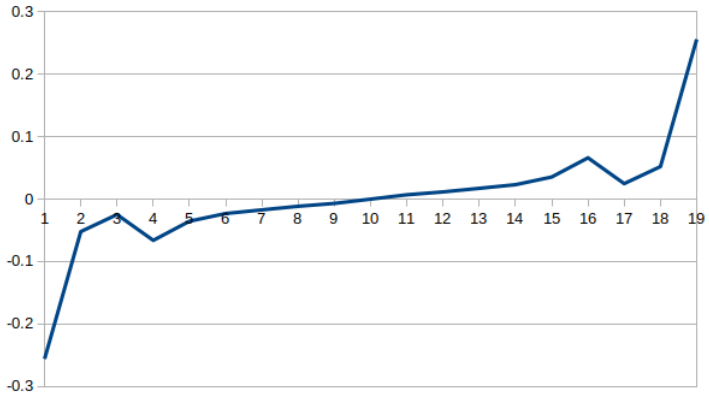


Figure 4.2: Graph showing the EAR for different ratios of 20 traders averaged across 100 supply and demand curves.

4.2 Algorithm Run-times

One of the major differences between BSE and TBSE is that when traders are run on TBSE their run-time has the potential to affect their performance, therefore Table 4.1 shows the average run-times for each algorithm's *getorder()* and *respond()* functions on both BSE and TBSE. These were recorded by replicating the experiments described in Section 3.3 but with only 100 runs per ratio. As Giveaway, Shaver and ZIC do not make use of a *respond()* functions, these are not included. The algorithms all execute *getorder()* faster on TBSE than BSE, but all by approximately the same magnitude, therefore the order of slowest to fastest does not change and is: GDX, AA, ZIC, ZIP, Shaver & Giveaway, with GDX being by far the slowest algorithms, taking almost 100 times the amount of CPU time than all other algorithms. The results for *respond()* show greater variation, with all algorithms executing *respond()* within 3μs of each other on BSE, but AA taking twice as long as the other traders on TBSE. How this affects the performance of these algorithms will be explored in Section 4.3.

	BSE		TBSE	
	getorder()	*respond()*	*getorder()*	*respond()*
Giveaway	30μs	N/A	12μs	N/A
Shaver	31μs	N/A	12μs	N/A
ZIC	50μs	N/A	23μs	N/A
ZIP	48μs	89μs	20μs	23μs
AA	65μs	90μs	32μs	51μs
GDX	5484μs	92μs	3403μs	22μs

Table 4.1: Average run-times for the main components of trading algorithms on BSE and TBSE.

4.3 Results

The following results were generated from the experiments described in Section 3.3. Each experiment compares two of the algorithms discussed in this project. In each market session there are 40 traders, 20 are buyers and 20 are sellers. The two sides of the market are split between the two algorithms, with each side having the same number of traders running each algorithm, and every possible ratio of the two algorithms is tested with 1000 market sessions per ratio. The following sections summarises the findings from these experiments.

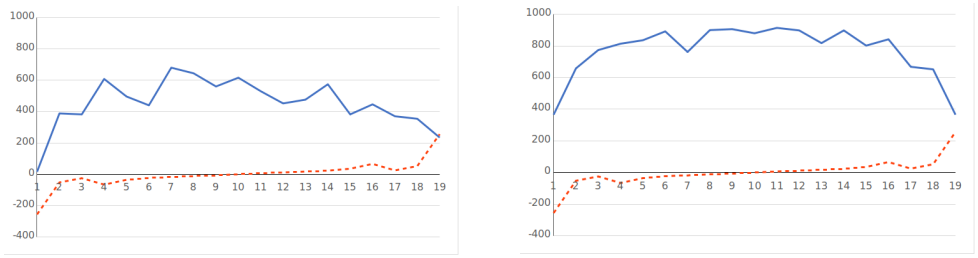
4.3.1 No Offset

This first set of results were produced from experiments where there was no offset function used to gradually alter the competitive equilibrium price throughout each market session. Table 4.2 shows the total number of tests that were won by each algorithm on BSE and TBSE, across all ratios. When considering this complete set of data, it is unnecessary to consider the EAR, as although Algorithm

		BSE		TBSE	
Algorithm A	Algorithm B	A Wins	B Wins	A Wins	B Wins
AA	ZIC	**13824**	5176	**16822**	2178
AA	ZIP	**13546**	5454	**12441**	6559
GDX	ZIC	**13683**	5317	**10674**	8326
GDX	ZIP	**15684**	3316	*8628*	***10372***
ZIP	ZIC	**9930**	9070	**12452**	6548
AA	GDX	**11395**	7605	**9599**	9401
SHVR	ZIC	**13186**	5814	**14725**	4275
SHVR	ZIP	**14750**	4250	**13774**	5226
SHVR	AA	8697	**10303**	8363	**10637**
SHVR	GDX	**11338**	7662	*3927*	***15073***
GVWY	ZIC	**10028**	8972	**14198**	4802
GVWY	ZIP	**10861**	8139	**13005**	5995
GVWY	AA	7402	**11598**	6704	**12296**
GVWY	GDX	6665	**12335**	***11292***	*7708*
GVWY	SHVR	7606	**11394**	7817	**11183**

Table 4.2: Results of all pairwise combinations of the algorithms used on this project, for both BSE and TBSE tests. Italic results are those where the overall winner is different on TBSE than on BSE.

A has a disadvantage when it is in the minority, it has an identical advantage for the ratios in which it is in the majority. The graphs below show a more detail analysis of each pair of algorithms, with the difference in the number of wins between Algorithm A and Algorithm B being plotted against the number of traders on each side of the market that were running Algorithm A. Algorithm A will be the first algorithm named, i.e. AA vs ZIC means Algorithm A is AA and Algorithm B is ZIC.

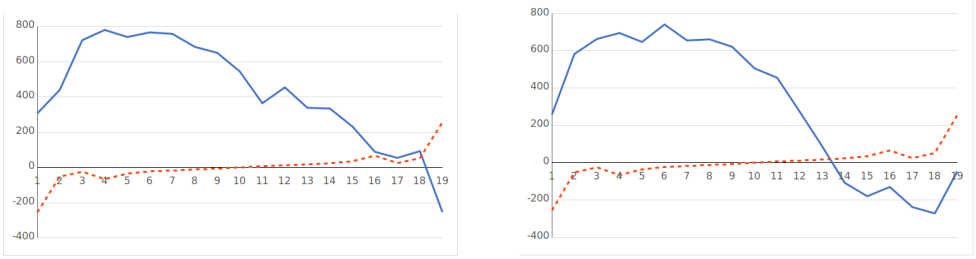


(a) BSE (b) TBSE

Figure 4.3: Difference in number of wins in 1000 tests between AA and ZIC, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

Figure 4.3 shows the first result, AA vs ZIC, which is a fairly conclusive victory for AA in both the BSE and TBSE tests. Although the overall winner does not change when the tests were run on the different simulators, the margin by which AA wins does increase on TBSE. The two graphs in Figure 4.3 share a similar shape, but with a greater difference in wins on TBSE. This is somewhat surprising, as ZIC is the faster of the two algorithms it might be expected that it may perform better in the time-dependant environment of TBSE. The asynchronous nature of TBSE also means that each trader has a temporarily limited view of the market when calculating its order price, as ZIC uses no information about the market to calculate its order price, this should be not be a disadvantage, whereas AA does use the information from the LOB, so would be at a disadvantage on TBSE compared to BSE. The reason AA performs better on TBSE than on BSE may in fact be due to the rate at which orders are posted. On BSE each trader gets to submit approximately one order every cycle. If a ZIC trader, by its random nature, submits a particularly "foolish" order (one that is very close to or equal to its limit price), then the next trader on the opposing side of the market has the opportunity to capitalise on this. If the next trader is an AA trader, as this is an intelligent trader that will recognise the ZIC trader's mistake, it will almost certainly place an order to execute a trade at that price, whereas if a ZIC trader is next, there is simply a random chance that it will execute a trade. This means that an AA trader may have to wait for numerous cycles before a ZIC trader posts a foolish order, by which time many traders may have already completed their

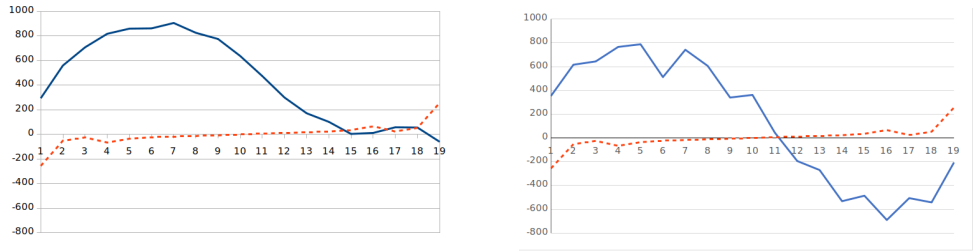
customer orders, or the AA trader may have simply traded at a less profitable price. On TBSE, all the ZIC traders will be posting orders as a far greater rate than on BSE, and therefore they will post less profitable orders at a greater rate, that an AA trader can more quickly take advantage of.



(a) BSE (b) TBSE

Figure 4.4: Difference in number of wins in 1000 tests between AA and ZIP, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

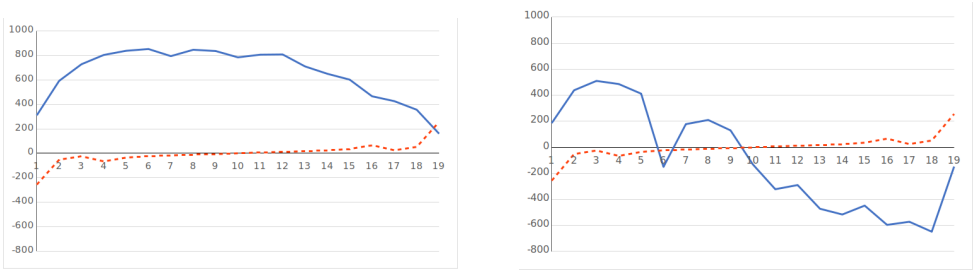
The results for the experiments comparing AA and ZIP are shown in Figure 4.4. Looking at Table 4.2 you can see the total number of wins for AA decreased by 1105 when ran on TBSE rather than BSE. By comparing the two graphs in Figure 4.4 it would appear that this decrease is a result of ZIP performing better when it is in the minority on TBSE, crossing the EAR line to win more than AA at 6 ZIP traders or less. It is possible that this increase in performance by ZIP on TBSE is a result of its slightly faster execution time which allows it to react faster to changes in the market. This pattern is similar in shape to the graph for BSE, and therefore the difference is not great enough to draw definite conclusions. An interesting observation is the upturn of the line at 19 AA traders, which closely follows the gradient of the EAR graph, this shows that is likely the EAR does have an affect on the algorithm's performance.



(a) BSE (b) TBSE

Figure 4.5: Difference in number of wins in 1000 tests between GDX and ZIC, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

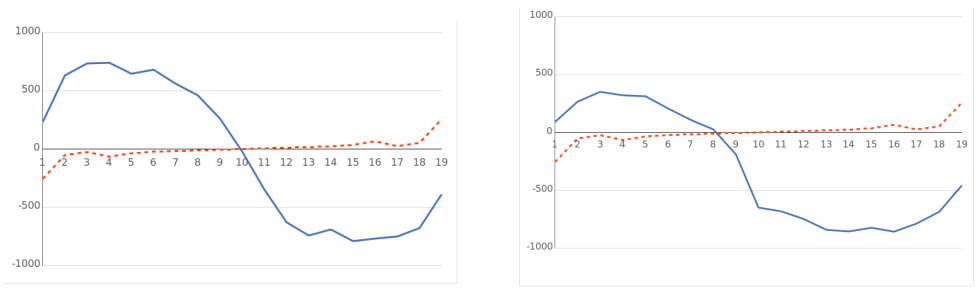
GDX vs ZIC, shown in Figure 4.5, presents a more interesting result. Here GDX can be seen to dominate ZIC on the BSE simulation, at least as far as a ratio of 14:6, but only achieving a far more narrow victory on TBSE. This would support the hypothesis that faster algorithms have an advantage on TBSE, as ZIC may be altering the market so quickly that by the time the slow GDX algorithm has completed its dynamic programming calculation, the market has changed so significantly that its order price is no longer optimum. Closer inspection of the two graphs shows that on BSE the performance of GDX peaks at 7 traders, and then gradually drops of, with ZIC winning experiments where it has 15, 16 or 19 traders, but only by a very slight margin. On TBSE, ZIC wins all tests where is is in the minority. On both ends of the TBSE graph the gradient can be seen to align with the EAR graph, giving more evidence that the dynamics caused by the ratio of traders have a significant impact on the shape of these graphs, at their extreme points.



(a) BSE (b) TBSE

Figure 4.6: Difference in number of wins in 1000 tests between GDX and ZIP, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

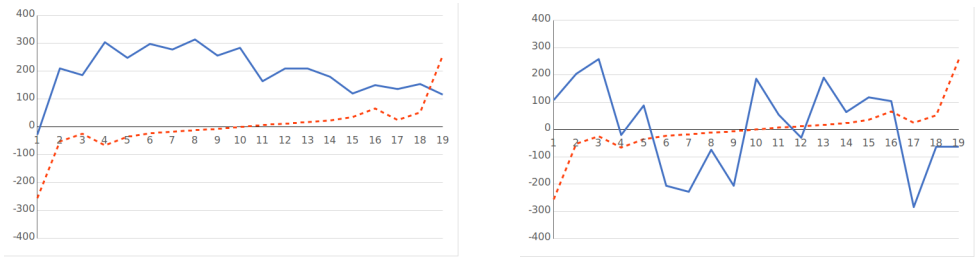
Figure 4.6 shows the results from GDX vs ZIP, this is the first time where the overall winner of the experiments changes depending on the simulation on which they are being tested. On BSE GDX is once again the dominant strategy, beating ZIP on all ratios except for 19:1 where, due to the EAR, ZIP wins a narrow victory. On TBSE, however, there is a significant difference with ZIP winning all tests where it is in the minority and, unlike in the GDX vs ZIC tests, doing so by enough of a margin to win the overall majority. The cause of ZIP's performance improvement is likely to be for the same reason as ZIC's in Figure 4.5: ZIP is significantly faster than GDX meaning it can better react to the market, but also performs better than ZIC did because it is also a more intelligent trader than ZIC. The point where there are 6 AA traders and 14 ZIP traders, shows a sharp spike downwards resulting in a win for ZIP. This anomaly may be the result of an unique interaction between these two traders specifically at this ratio, however this was not possible to confirm from the data recorded. Alternatively, it may be that a majority of the 10 different supply and demand schedules used at this ratio particularly favoured ZIP giving it an advantage, this could be shown by repeating the experiment, however this was not possible with the resources of this project. Section 5.3 discussed further work that could be undertaken to better understand how different supply and demand curves affect different algorithms.



(a) BSE (b) TBSE

Figure 4.7: Difference in number of wins in 1000 tests between ZIC and ZIP, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

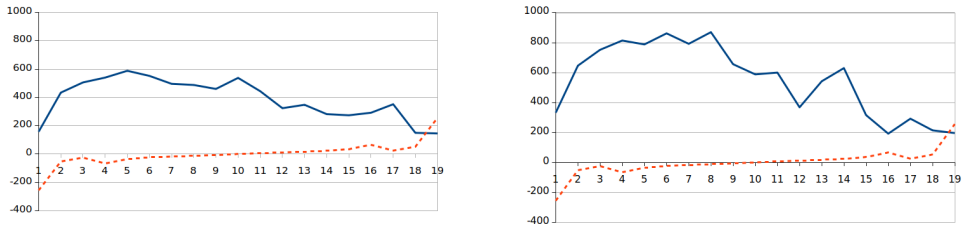
The results for experiments running ZIC vs ZIP are shown in Figure 4.7. Table 4.2 shows that on BSE ZIP wins overall but only by a small margin. It is surprising that the results are so close when ZIP is known to be the far more intelligent algorithm. Both algorithms win when they are in the minority, with near equal wins and losses when the ratio is 1:1. On TBSE a significant shift in favour of ZIP can be seen. Here it is also seen, on both BSE and TBSE, that in the extreme minority both algorithms performance deteriorates, following the path of the EAR curve. Overall a decisive win for ZIP instinctively feels correct, however when compared to the BSE results it is more unexpected as ZIC is faster and is not negatively effected by a lack of information, whereas ZIP is. Further research may be needed to analyse the exact market conditions which lead to either one of these algorithms winning.



(a) BSE (b) TBSE

Figure 4.8: Difference in number of wins in 1000 tests between AA and GDX, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

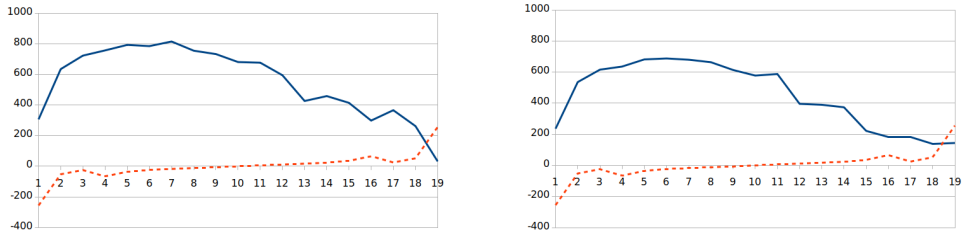
AA vs GDX, shown in Figure 4.8, produces a very strange result. On BSE AA consistently wins across all ratios, except for 19 AA to 1 GDX, this is not a surprising results as there exists previous work which would support this result, such as De Luca & Cliff (2011)[7]. On TBSE there is no clear pattern to the results. Overall, Table 4.2 shows that AA remains the winner, however by a very slight margin of 198 wins. Although there is no clear answer as to why the results vary so widely at different ratios, it is interesting to note that the average number of trades completed by each algorithm decrease as the number of GDX traders increase. This is a phenomenon replicated by all tests involving GDX. These two algorithms are by far the most complex, so it is possible that either the algorithms themselves, or their implementations within this project have an unexpected interaction on TBSE which results in a slowing down of trading behaviour. Further study of the GDX and AA algorithms and their implementations may be needed in order to fully understand these results.



(a) BSE (b) TBSE

Figure 4.9: Difference in number of wins in 1000 tests between Shaver and ZIC, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

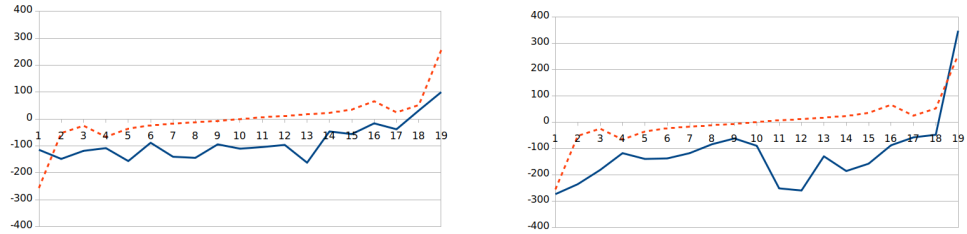
Figure 4.9 shows the Shaver vs ZIC results, with Shaver consistently performing better than ZIC. It is logical that Shaver should beat ZIC, as ZIC simply places random orders that would not result in a loss, whereas Shaver acts to always have the best bid or ask on the market. The result of this is that Shaver on average made 6.05 trades per trader, whereas ZIC only made 2.85, giving Shaver a lot more opportunity to make profit. Both curves in Figure 4.9 are almost identical, this makes sense as both algorithms are very fast, and neither requires historical knowledge of the market, so there is no reason why their performance should dramatically change depending on the two environments that they are tested on.



(a) BSE (b) TBSE

Figure 4.10: Difference in number of wins in 1000 tests between Shaver and ZIP, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

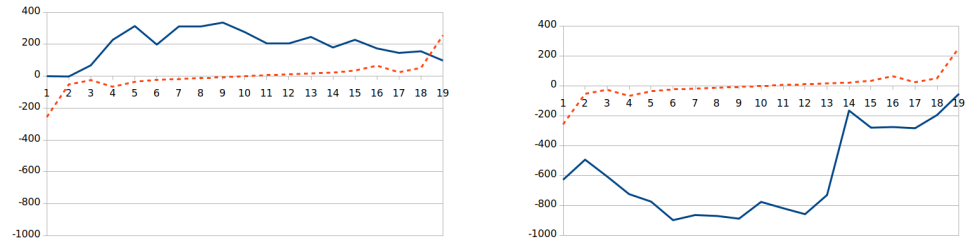
Shaver vs ZIP shows similar results to Shaver vs ZIC. In Figure 4.10 it is once again obvious that Shaver dominates at nearly every ratio, with the exception of 19 Shaver to 1 ZIP. This once again would appear to be due to Shaver's ability to execute a great number of trades, averaging 6.72 per trader whereas ZIP only manages 4.96.



(a) BSE (b) TBSE

Figure 4.11: Difference in number of wins in 1000 tests between Shaver and AA, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

In Figure 4.11 Shaver is finally toppled by AA on both BSE and TBSE. The difference in total wins by AA on BSE and TBSE is once again very slight, only 334 more on TBSE. This time AA manages to execute 0.57 more traders per trader than Shaver, and also gains 0.50 more profit per trade, meaning AA traders have more opportunities to make a profit, and makes more profit when they do trade. Once again there is little change in the graphs between BSE and TBSE, it is interesting to note that all experiments involving AA but not GDX (Figures 4.3, 4.4, 4.11, 4.15) and all experiments involving Shaver but not GDX (Figures 4.9, 4.10, 4.11, 4.17) show very little difference between the experiments ran on BSE and those ran on TBSE. This suggests that GDX, with its very slow run-time, may be the algorithm which is most significantly affected by the asynchronous environment of TBSE.

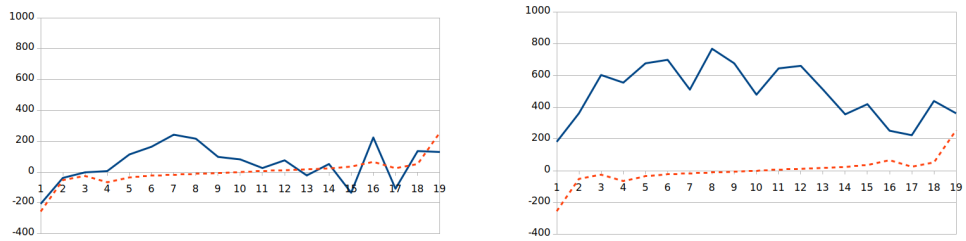


(a) BSE (b) TBSE

Figure 4.12: Difference in number of wins in 1000 tests between Shaver and GDX, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

The Shaver vs GDX tests are another example where a large difference between how the algorithms perform on BSE and TBSE can be seen. Figure 4.12 shows how Shaver is clearly the dominant strategy

when ran on BSE, only loosing when 19 out of 20 traders are running Shaver. This however completely reverses when ran on TBSE, where GDX wins at all ratios, and at some by over 800 wins. This is a very surprising result. As GDX is the slowest algorithm tested and Shaver is the second fastest, it could be expected that Shaver would continue to dominate on TBSE as it did on BSE. By looking at the results from the tests in greater detail, it would appear that Shaver wins on BSE simply because it executes the most trades. Shaver averaged 9.78 traders per trader and GDX only 7.12, when the average profit per trader is divide by this, to calculate the average profit per trade, it is clear that GDX traders make much more profitable trades, with an average profit of 29.36, whereas Shaver only makes a profit of 23.92. This suggests that by simply always having the best bid or ask on the market Shaver traders can ensure they make the most trades, even if they are not as profitable as the GDX trades. On TBSE it is GDX that makes more trades and greater profit per trade, which allows it to win all tests. The reason Shaver makes less trades against GDX on TBSE than it does on BSE is likely to be a result of GDX's long run-time. When the GDX trader's algorithms are calculating, which takes significantly longer than any other algorithm in this project, it is as if the market is a homogeneous Shaver market. In Cliff's BSE guide[2], he shows there is a pause before homogeneous Shaver traders start trading, as they each shave one of the best buy/ask one at a time before one trader crosses the spread and trades start occurring rapidly. Therefore, in these tests if a GDX algorithm completes its calculations before the Shaver traders have began trading, it may be able to jump in and steal the deal. On the TBSE graph of Figure 4.12 there is a sharp decrease in the amount by which GDX wins when there are 14 or more Shaver traders, and therefore 6 or less GDX traders. This could be because when there are more Shaver traders in the market there will be more traders shaving single units off the best buy/ask which means they can converge more rapidly. This may mean that they more consistently manage to converge before a GDX trader can complete its calculations and steal the deal.

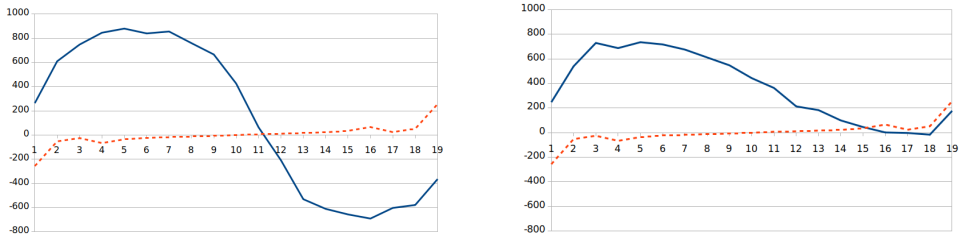


(a) BSE

(b) TBSE

Figure 4.13: Difference in number of wins in 1000 tests between Giveaway and ZIC, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

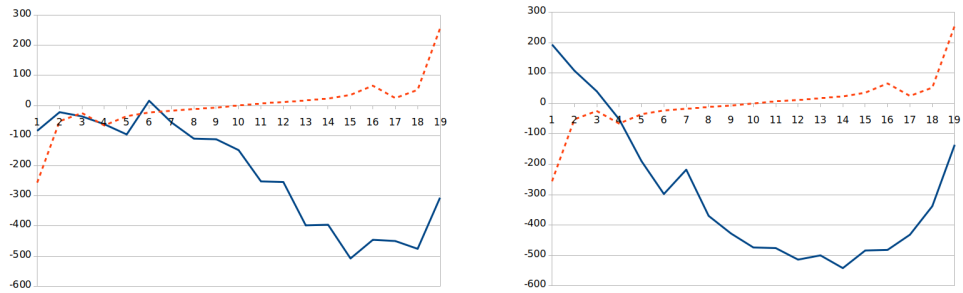
Finally, the five Giveaway tests are analysed, starting with Giveaway vs ZIC, shown in Figure 4.13. As explained in Section 2.4 the Giveaway algorithm never intends to make a profit, as it always posts orders at its customer order's limit price. It can however "accidentally" make a profit, if it crosses the spread so that the trade actually executes at a price that produces a profit. This is what allows Giveaway to win, both on BSE and TBSE. At first glance, this is a surprising result, as it seems counterintuitive that accidental profit could amount to enough to produce a win. As with some of the other more surprising results seen above, the key factor would appear to be that Giveaway simply makes so many more trades. On BSE Giveaway consistently makes twice as many trades as ZIC and on TBSE Giveaway makes over three times as many trades on average.



(a) BSE (b) TBSE

Figure 4.14: Difference in number of wins in 1000 tests between Giveaway and ZIP, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

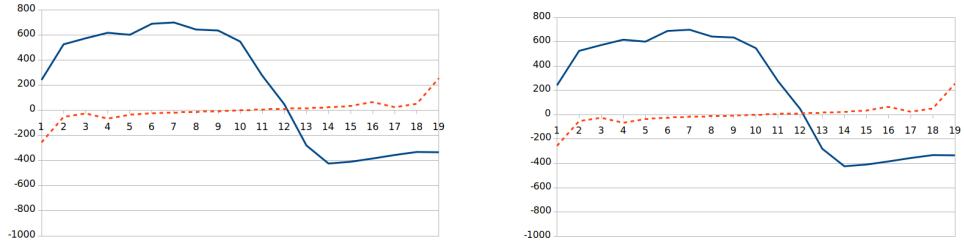
Giveaway vs ZIP once again shows that Giveaway can still produce a respectable profit, which can be enough to win. Figure 4.14 shows that on BSE Giveaway wins all tests where it is in the minority, and continues to win until there are more than 11 Giveaway traders on each side of the market, at which point ZIP becomes the superior strategy. On TBSE ZIP does not perform as well as on BSE, only winning when it as 4 or less traders on each side of the market and by a very small margin. This would be explained by the market changing during its calculation causing it to post less profitable bids and asks than if it is on BSE where the market cannot change during its calculation. Again, Giveaway manages to win by successfully completing more trades than the ZIP traders.



(a) BSE (b) TBSE

Figure 4.15: Difference in number of wins in 1000 tests between Giveaway and AA, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

Figure 4.15, shows the results of the Giveaway vs AA tests. Here AA is clearly the dominant strategy on both BSE and TBSE, winning slightly more heavily on TBSE. Although, as with the other Giveaway tests, Giveaway still manages to produce slightly more trades per trader, its accidentally produced profit is no match for AA which produces a far greater profit, clearly showing it to be the better strategy at all but a few ratios.



(a) BSE (b) TBSE

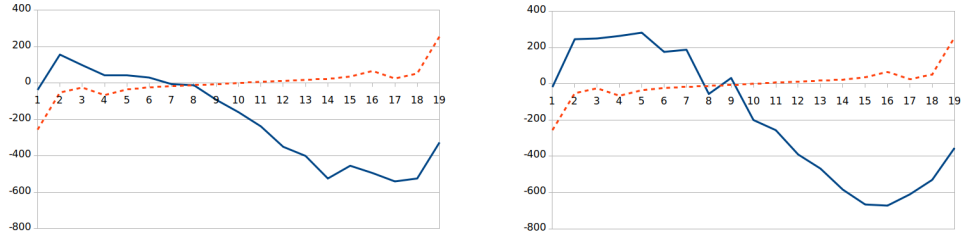
Figure 4.16: Difference in number of wins in 1000 tests between Giveaway and GDX, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

The experiments for Giveaway vs GDX, shown in Figure 4.16, show that on BSE, GDX, as expected,

Algorithms Beaten		Total Wins	
AA	5	AA	60666
Shaver	4	Shaver	59365
GDX	3	GDX	56969
Giveaway	2	Giveaway	42562
ZIP	1	ZIC	34349
ZIC	0	ZIP	31089

Table 4.3: Two hierarchies for ranking algorithms as a result of BSE no offset experiments.

is the dominant strategy. On TBSE however the results are reversed and it is Giveaway which performs significantly better. On both BSE and TBSE Giveaway consistently has the higher average number of trades, but on BSE this is far outweighed by the far superior profit that GDX can generate with each trade. On TBSE this is not the case, this may be a result of how damaging GDX's slow execution time is on its performance, it is nearly 300 times slower than Giveaway. This may mean that by the time its execution of *getorder()* is complete, the market has shifted enough to render the order no longer viable, which may be explain the low number of trades GDX manages to execute on TBSE.



(a) BSE (b) TBSE

Figure 4.17: Difference in number of wins in 1000 tests between Giveaway and Shaver, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

The final result from this set of experiments is Giveaway vs Shaver, shown in Figure 4.17. Here there is very little difference between how the algorithms perform on BSE or TBSE. This is an expected result, as both algorithms execute very quickly, being the fastest and second fastest algorithms in these experiments (see Table 4.1), it is not surprising that TBSE has little affect on their performance. It is interesting to note, that on both ends of both graphs you can see how the number of wins follows the gradient of the EAR graph, once again showing that in these extreme cases the ratio of algorithms clearly has a significant impact in their performance.

Summary

I decided on using two different methods for determining hierarchies of performance, which produced two different hierarchies. The first was the number of other algorithms that each algorithm beat head to head, e.g. AA beat all five other algorithms on BSE so gets a score of 5, ZIC lost all experiments head to head so gets a score of 0. The algorithms are then ranked in order of that number, deciding ties by the winner of the experiment between the two tied algorithms. The second method was to total the absolute number of wins across all tests for each trader and rank them by that number. For BSE these two hierarchies are shown in Table 4.3

Much of these hierarchies is supported by previous work, ZIP beating ZIC was reported in Cliff (1997)[4], GDX being superior to ZIP was discussed in Tesauro & Bredin (2002)[14], although it is interesting that ZIP actually outperforms GDX on total wins across all algorithms. AA dominating both GDX and ZIP was shown in De Luca & Cliff (2011)[7], although work such as Snashall & Cliff (2019)[13] has suggested that in tests where there are more than two algorithms being compared AA does not always dominate, it is fair to say that the results produced by BSE are supported by preexisting research. Shaver and Giveaway have not been studied in great detail before, so their performance is being reported for the first time. Giveaway being higher on these hierarchies than ZIC is not a particularly unsurprising result. Giveaway always posts its limit price, which means that they always have the best chance of making a

Algorithms Beaten		Total Wins	
AA	5	AA	61795
Shaver	3	Giveaway	56382
Giveaway	3	Shaver	51972
ZIP	2	GDX	51484
GDX	2	ZIP	40604
ZIC	0	ZIC	26129

Table 4.4: Two hierarchies for ranking algorithms as a result of TBSE no offset experiments.

trade, which leads Giveaway traders to make far more trades than any other trader type. ZIC produces random order prices, which means sometimes they will post order prices that will make a small profit, sometimes order prices that will produce a large profit, and sometime they will make order prices which will never be traded. It would appear that the amount of times which ZIC traders post orders that won't be accepted prevents them for completing enough trades to beat Giveaway. The fact that Giveaway does beat ZIP is a very surprising result however, as ZIP is known to be an algorithm which can efficiently find the competitive equilibrium price. The reason for Giveaway beating ZIP is likely to be the same as why Giveaway beats ZIC, it simply executes so many trades that even if they don't generate a large profit individually, collectively they are superior. Shaver's impressive performance is in someways surprising due to its simplicity, but makes sense when you consider the importance of successfully completing customer orders. By always attempting to beat the best bid or ask on the market, they are primed to always be at the top of the LOB when a trader on the opposite side of the market crosses the spread. The fact that completing as many customer orders as possible is almost as important as how much profit is made from each one, is a very interesting result.

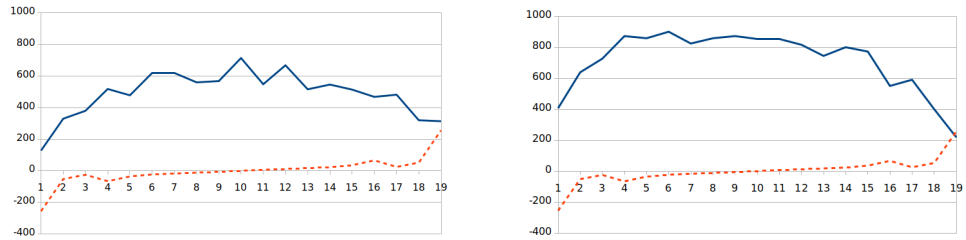
The same hierarchies for the tests ran on TBSE are shown in Table 4.4. Here the biggest difference from the BSE hierarchies is that GDX falls down to 5th position on the Algorithms Beaten list and 4th position on the Total Wins list. This is an expected change, as all other algorithms execute their *getorder()* function in between 12 and 32μs whereas GDX takes on average 3403μs. Clearly this is a major disadvantage when these traders are operating on the asynchronous TBSE exchange. It is interesting that although AA is the second slowest executing trader on TBSE, its positions in the hierarchies remains unchanged. This suggests that the small difference in execution time between all of the fastest algorithms is not sufficient to overcome the advantage of having a more intelligent algorithm which most accurately predicts the competitive equilibrium price.

4.3.2 Variable Offset

In real financial markets, the competitive equilibrium price is not static, and can vary significantly over time as the supply and demand for the asset changes. To better represent this within these experiments, an offset was added to the equilibrium price, determined by the order schedule. This is done by adding a fixed quantity to customer order prices, based on the time at which the order is issued. Details of this offset can be found in Section 3.3.2. The idea of these experiments is compare how algorithms can follow a changing equilibrium price. An algorithm which can react to changes in the market more quickly can better take advantage of algorithms that do not do this well, and therefore increase their profitability. The complete set of results showing the total number of wins for each algorithm pair on both BSE and TBSE is found in Table 4.5. Each pair is then discussed in turn below.

		BSE		TBSE	
Algorithm A	Algorithm B	A Wins	B Wins	A Wins	B Wins
AA	ZIC	**14146**	4854	**16276**	2724
AA	ZIP	**15115**	3885	*9231*	***9769***
GDX	ZIC	**11039**	7961	*8650*	***10350***
GDX	ZIP	**12986**	6014	*5007*	***13993***
ZIP	ZIC	8753	**10247**	***17025***	*1975*
AA	GDX	**5420**	3580	*8697*	***10303***
SHVR	ZIC	**13419**	5581	**14287**	4713
SHVR	ZIP	**15875**	3125	*8717*	***10283***
SHVR	AA	8718	**10282**	7221	**11779**
SHVR	GDX	**15142**	3858	*2625*	***16375***
GVWY	ZIC	**10438**	8562	**13001**	5999
GVWY	ZIP	**11858**	7142	**9835**	9165
GVWY	AA	7292	**11708**	8773	**10227**
GVWY	GDX	**10971**	8029	**13538**	5462
GVWY	SHVR	7661	**11339**	8430	**10570**

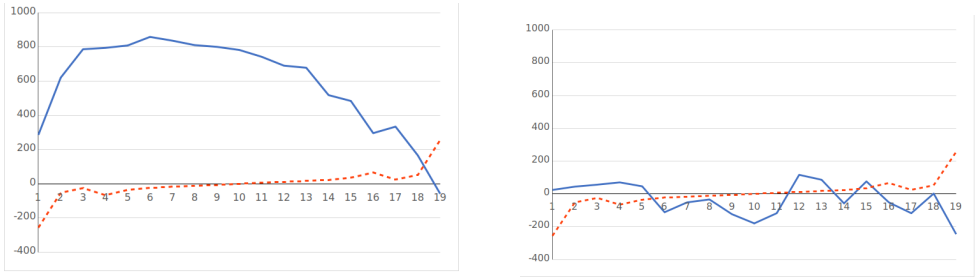
Table 4.5: Results of all pairwise combinations of the algorithms used on this project, for both BSE and TBSE tests using the offset function shown in Figure 3.7. Italic results are those where the overall winner is different on TBSE than on BSE.



(a) BSE (b) TBSE

Figure 4.18: Difference in number of wins in 1000 tests with variable offset between AA and ZIC, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

Figure 4.18 shows the first set of results, AA vs ZIC. These graphs are very similar to those seen in the corresponding tests without an offset, Figure 4.3. The total number of wins for each trader are also very similar, suggesting that this test is unaffected by the offset used, and that AA continues to dominate ZIC at every ratio. This is due to AA, as its name "Adaptive-Aggressive" suggests, being able to adapt its order prices to match the changing equilibrium whereas ZIC will always post random order prices.

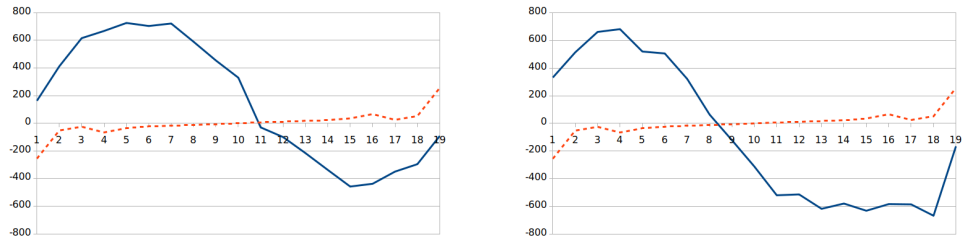


(a) BSE (b) TBSE

Figure 4.19: Difference in number of wins in 1000 tests with variable offset between AA and ZIP, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

AA vs ZIP, in contrast to AA vs ZIC, does show significant change from the tests without offset. Figure 4.19 shows that on BSE the result is quite similar to that of the no offset test, shown in Figure

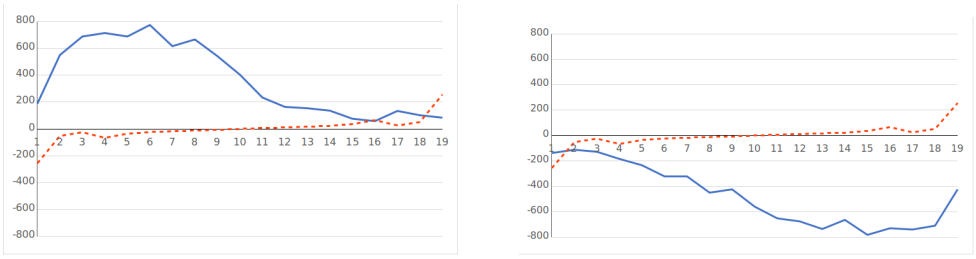
4.4, with AA even extending its lead over ZIP. This is a sign that when both algorithms always have an uninterrupted view of the market activity, as is the case on BSE, not only does AA continue to be more profitable than ZIP, but that it follows the change of the equilibrium price better as well. In the TBSE test there is a substantial change in the number of wins won by both algorithms at each ratio, with ZIP being declared the overall winner. This may indicate that neither algorithm managed to track the moving equilibrium price well when their view of the market is temporarily limited when calculating their order price, as both use market information to adapt their strategy.



(a) BSE (b) TBSE

Figure 4.20: Difference in number of wins in 1000 tests with variable offset between GDX and ZIC, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

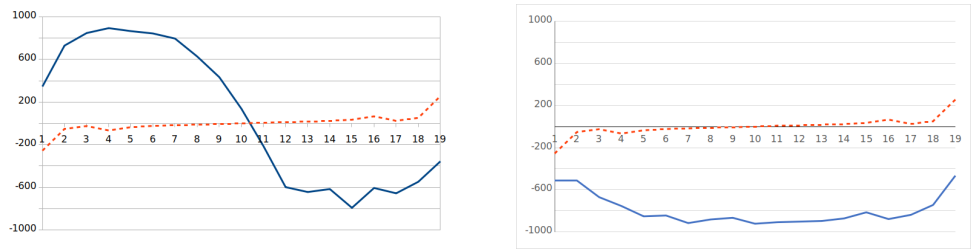
In the GDX vs ZIC tests, shown in Figure 4.20, there is further changes from the no offset tests (Figure 4.5). When the tests were run on BSE the resulting graph is shifted downwards due to ZIC winning a greater number of matches when in the minority. As ZIC does not attempted to track the equilibrium price, its increase in number of wins is more likely to be a result in a decrease in performance by GDX, however GDX still manages to win the majority of matches overall. On TBSE ZIC takes the overall lead, although it may be surprising for ZIC to win any test, as in the tests without an offset it was judged to be the poorest performing algorithm on both BSE and TBSE, it makes sense that GDX would struggle on TBSE when the equilibrium price is not fixed. From the first set of experiments it is already known that GDX struggles in an asynchronous exchange simulator, but when the equilibrium price is moving, it struggles even more because as the equilibrium price moves, this causes greater variation in the order prices produced by other algorithms, as they attempt to track it. As it is already known that GDX's slow execution time will result on information being missed by GDX during its calculations it causes GDX to post less profitable orders, hence giving ZIC the opportunity to make a greater profit, by simply posting random orders.



(a) BSE (b) TBSE

Figure 4.21: Difference in number of wins in 1000 tests with variable offset between GDX and ZIP, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

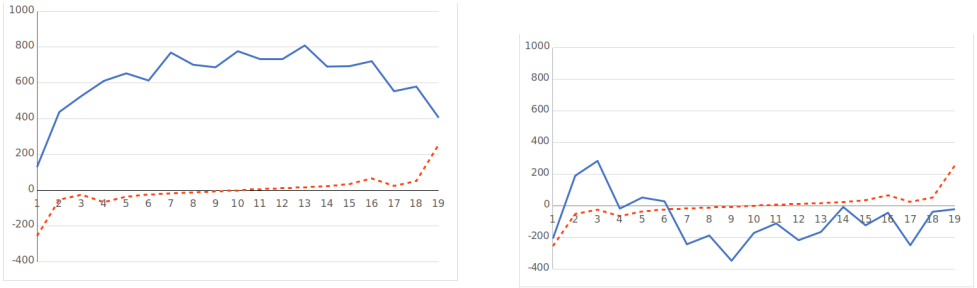
Figure 4.21 shows the results for GDX vs ZIP. On BSE, GDX once again beats ZIP at every ratio but one, although interestingly it does not do so as convincingly as in Figure 4.6, maybe indicating that GDX may again have found following the changing equilibrium price challenging. In the tests without an offset ZIP won on TBSE, and this result is furthered when an offset is introduced, with ZIP winning all but one ratio. This provides further evidence along with Figure 4.20 that GDX cannot keep up with the changing equilibrium price on TBSE, and for that it is outperformed by faster-responding algorithms.



(a) BSE (b) TBSE

Figure 4.22: Difference in number of wins in 1000 tests with variable offset between ZIC and ZIP, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

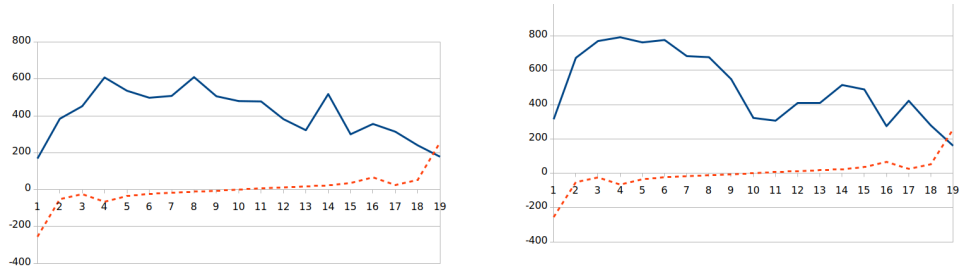
For tests run on BSE with an offset, ZIC vs ZIP produces a similarly shaped graph (Figure 4.22) to the test without an offset (Figure 4.7), however the slight majority that ZIP has without an offset, is now reversed and ZIC wins the most tests when there is an offset. This may be a result of ZIP not being able to extract a profit as efficiently when the equilibrium price is moving. On TBSE a dramatic change in the outcome is seen, on the tests with an offset ZIP wins by the largest margin seen in this project. This is a highly unexpected result, as on BSE ZIP performs worse with an offset than when an offset is used, whereas on TBSE the reverse is true. As it is the case that in both cases the ZIC vs ZIP experiments have produced unexpected results, it is an interesting avenue for future work to do further analysis to work out what produces wins for either algorithms. Rather than using a set of 10 random order schedules, there may be benefit to experimenting with fixed order schedules to see which schedules benefit which algorithms more, as it may be the case that by chance the majority of the 10 order schedules used naturally favoured the ZIP algorithm.



(a) BSE (b) TBSE

Figure 4.23: Difference in number of wins in 1000 tests with variable offset between AA and GDX, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

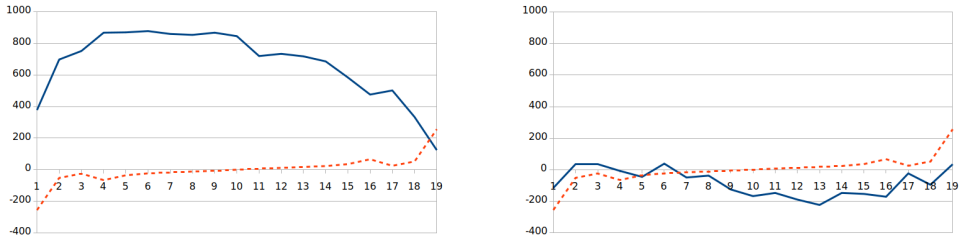
In the tests running AA vs GDX, shown in Figure 4.23, it is seen that on BSE, AA increases in dominance over GDX, which shows that when the algorithms are given enough time to execute without any other changes happening to the market in that time, AA both performs better and follows a varying equilibrium price better. On TBSE, a jagged shape is seen, similar to the one that was seen in Figure 4.8, this indicates that the shape may not be completely random, but in fact be a result of different ratios having an advantage for a certain trader. In these tests however, GDX comes out as the better performing strategy. This is surprising due to how dominant AA has been throughout these experiments and may suggest that there is a unique interaction that occurs between these algorithms which cause the behaviour seen in Figures 4.8 and 4.23.



(a) BSE (b) TBSE

Figure 4.24: Difference in number of wins in 1000 tests with variable offset between Shaver and ZIC, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

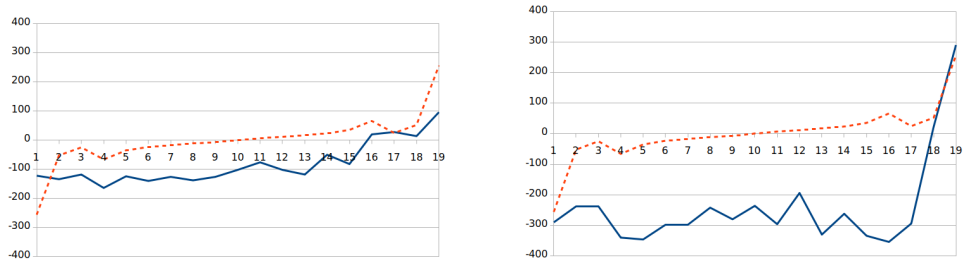
The results shown in Figure 4.24 are for Shaver vs ZIC results with an offset altering the equilibrium price, here very little difference from Figure 4.9 is seen, with Shaver remaining the superior strategy on both BSE and TBSE. This makes sense as neither algorithm attempts to track the equilibrium price and just acts either randomly, in ZIC's case, or based on other traders orders, in Shaver's case, so there is little reason for a moving equilibrium price to have an affect.



(a) BSE (b) TBSE

Figure 4.25: Difference in number of wins in 1000 tests with variable offset between Shaver and ZIP, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

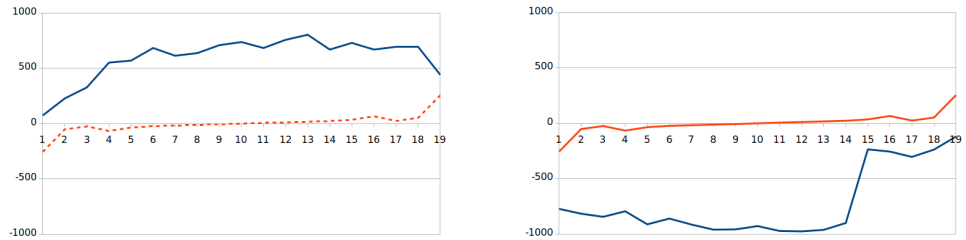
On the BSE graph of Figure 4.25 an almost identically shaped graph to the one on the corresponding section of Figure 4.10 is seen, this makes sense as Shaver acts by following the orders of other traders, simply beating them by a single unit of currency, so Shaver only follows the moving equilibrium price as well as its opposing traders do. On the TBSE graphs however there is a large difference, with ZIP coming out on top in tests using a variable offset, as opposed to Shaver winning when there was no offset. This could be as a result of Shaver being at a disadvantage due to having to wait until the ZIP traders have altered their order prices towards the new equilibrium price as it moves, before Shaver's orders begin to reflect the new equilibrium price. This is however contradicted by both Figures 4.26 and 4.27, where their graphs are both similar to their no offset counterparts. This result would appear to be an outlier as all other Shaver tests that use a moving equilibrium price act in a similar manner to the tests which use a fixed equilibrium price.



(a) BSE (b) TBSE

Figure 4.26: Difference in number of wins in 1000 tests with variable offset between Shaver and AA, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

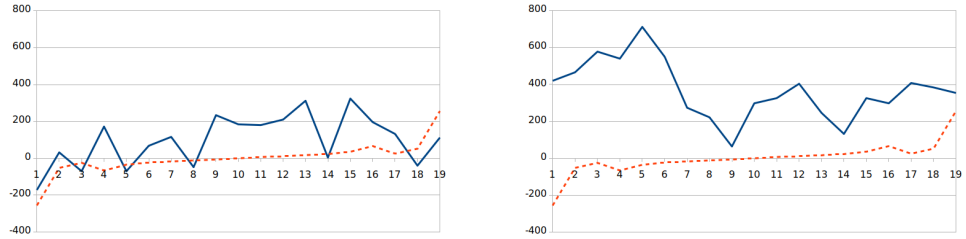
Both Shaver vs AA tests in Figure 4.26 are very similar to the previous tests in Figure 4.11, with the difference on BSE being only 21 runs out of 19,000. This again supports the idea that as all Shaver's order prices are simply based on the order prices of other traders it does not really matter how well the other algorithm follows the movement of the equilibrium price, as Shaver will still always place its orders at prices just better than the best bid or ask on the market, and therefore will continue to win and lose by the same proportions.



(a) BSE (b) TBSE

Figure 4.27: Difference in number of wins in 1000 tests with variable offset between Shaver and GDX, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

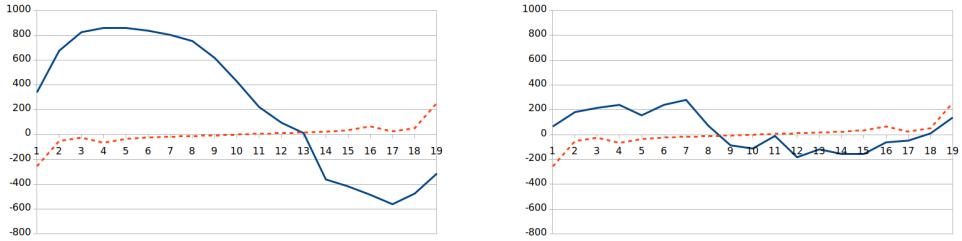
For Shaver vs GDX in offset tests, Figure 4.27 again produces similar results to the no offset tests, Figure 4.12, but with Shaver significantly increasing its winning margin on BSE significantly, this may be a result of GDX again struggling to follow the movement of the equilibrium price and posting orders that produce less profitable trades, that opposing Shaver traders take advantage of. The similarities of this TBSE tests and the one performed without an offset are likely to be for the same reasons described in Section 4.3.1.



(a) BSE (b) TBSE

Figure 4.28: Difference in number of wins in 1000 tests with variable offset between Giveaway and ZIC, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

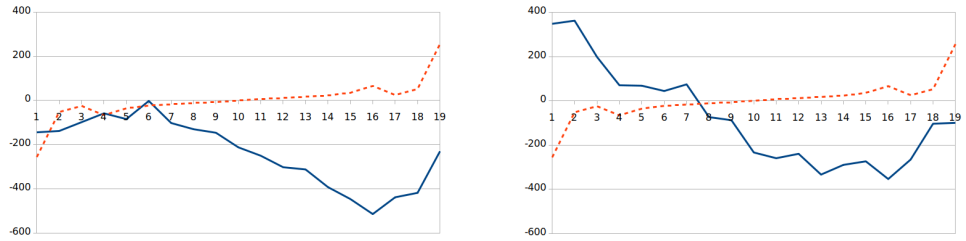
Giveaway and ZIC are the two least intelligent algorithms tested during this project which is likely to be why the results from the experiments comparing them, Figure 4.28 for the offset results and Figure 4.13 for the results without offset, are so irregular. Although they are irregular, it would appear that overall results of both sets of results are quite similar, with Giveaway winning in all four cases. It again makes sense that for traders that do not attempt to predict the equilibrium price, varying it would make little difference in either of their performances.



(a) BSE (b) TBSE

Figure 4.29: Difference in number of wins in 1000 tests with variable offset between Giveaway and ZIP, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

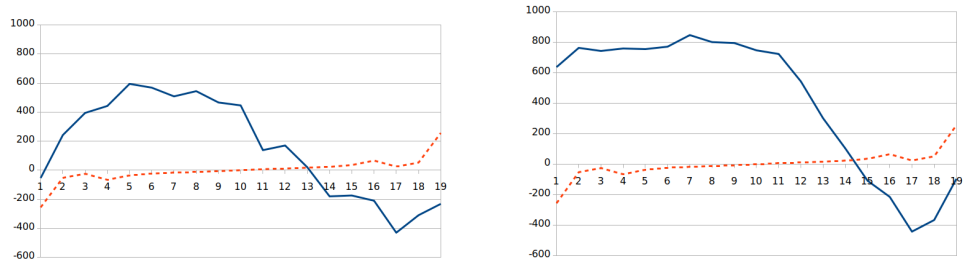
Figure 4.29 shows the results using an offset for Giveaway vs ZIP, here a very similar result to the one in Figure 4.14 is seen, indicating that Giveaway's ability to make the most trades under any circumstances allows it continually outperform ZIP. On TBSE Giveaway's performance is significantly lower than when there was no offset. There is no clear explanation for this, as Giveaway's performance should not change regardless of how the equilibrium price changes, which would indicate the difference between results is due to an increase in ZIP's performance, but it would appear strange for ZIP to improve when the equilibrium price moves. In cases where ZIP is against another algorithm that attempts to post orders close to an estimated equilibrium price, it would be possible for these conditions to improve ZIP's relative performance if ZIP could better follow the movement of the equilibrium price. That does not apply here however as in neither test does Giveaway do anything other than post order at its limit price, which is directly linked to the equilibrium price as a result of the distribution of orders via the order schedule. This is a case which could be inspected more deeply if further work was carried out in this area.



(a) BSE (b) TBSE

Figure 4.30: Difference in number of wins in 1000 tests with variable offset between Giveaway and AA, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

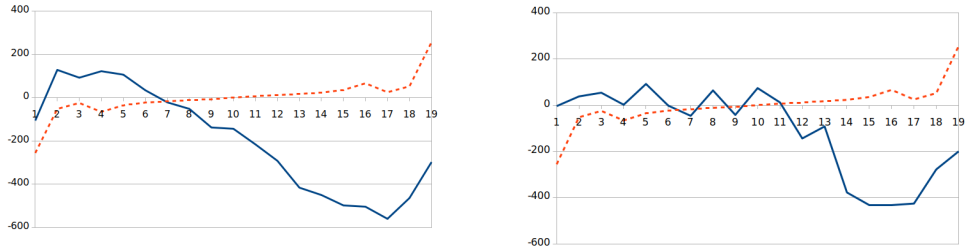
Giveaway vs AA, shown in Figure 4.30, once again produces similar results to those that did not use an offset (Figure 4.15). The BSE results are almost identical, indicating that when AA traders can respond to all changes that happen in the market, they continue to perform well and significantly outperform Giveaway. On TBSE, again AA continues to consistently outperform Giveaway, however on the tests that made use of the variable offset AA's winning margin was not as great. As Giveaway should be unaffected by the moving equilibrium price this is a result of AA not being able to predict the equilibrium price as well as it can when it is static, therefore missing out on potential profit.



(a) BSE (b) TBSE

Figure 4.31: Difference in number of wins in 1000 tests with variable offset between Giveaway and GDX, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

Similar shapes are once again found when comparing Figure 4.31, the Giveaway vs GDX graph for testing using an offset, and Figure 4.16, the graphs from tests without the offset. Here it is found that for both simulators GDX performs worse when the offset is applied, this backs up the other GDX experiments which has suggested that GDX does not cope with this style of equilibrium change. On BSE this even results in Giveaway getting the overall majority of victories over GDX.



(a) BSE (b) TBSE

Figure 4.32: Difference in number of wins in 1000 tests with variable offset between Giveaway and Shaver, across all ratios of 20 traders. Red dotted line is the Equal-Availability Ratio (EAR). See Section 4.1 for details.

The final experiment ran for this project was Giveaway vs Shaver, shown in Figure 4.32. As expect, these results are not dissimilar to those found in Figure 4.17, as neither algorithm is intelligent, and therefore does not attempt to estimate the equilibrium price. The result of this is that on both BSE and TBSE Shaver remains the dominant strategy.

Summary

Again, the same methods for determining the order of the hierarchies were used. Table 4.6 shows the BSE hierarchies and Table 4.7 shows the TBSE hierarchies.

Algorithms Beaten		Total Wins	
AA	5	Shaver	64493
Shaver	4	AA	62105
Giveaway	3	GDX	58391
GDX	2	Giveaway	48220
ZIC	1	ZIC	37216
ZIP	0	ZIP	28919

Table 4.6: Two hierarchies for ranking algorithms as a result of BSE offset experiments.

On BSE there are a number of things to note. ZIP falls to the bottom of both lists, being beaten by all other algorithms including ZIC, this is maybe the most surprisings result found in these experiments

as it directly contradicts Cliff (1997)[4]. Shaver also takes the number one spot from AA on the Total Wins list, but as Shaver was the second best performing algorithm on the no offset tests, this is not overly surprising. Cliff (2019)[3] also suggested that Shaver could perform better than AA under some conditions, so this is not an entirely new result. Other than those changes the orders do not change too much from the no offset tests, AA is still performing better than GDX, which is still performing better than ZIP.

Algorithms Beaten		Total Wins	
ZIP	4	AA	56210
AA	3	Giveaway	55717
Giveaway	3	ZIP	49547
GDX	2	GDX	45797
Shaver	2	Shaver	43420
ZIC	1	ZIC	25761

Table 4.7: Two hierarchies for ranking algorithms as a result of TBSE offset experiments.

When the same offset experiments are conducted on TBSE, ZIC falls back to last place, and ZIP unexpectedly rises to the number one position on the Algorithms Beaten list by beating all algorithms except for Giveaway, this suggests that ZIP better handles a moving equilibrium price than ZIC when they both have temporarily limited views of the market when calculating order prices. AA still wins the most tests in total. Shaver falls to 5th place in both lists, only managing to beat ZIC and Giveaway on TBSE offset tests. It is interesting that the only two algorithms it manages to beat, are the two other unintelligent algorithms that do not attempt to track the equilibrium price, although on the no offset tests the only additional algorithms it beat was ZIP, the algorithm which does particularly well under TBSE offset conditions. GDX continues to struggle more on TBSE than it does on BSE, this is an expected result due to GDX's exceptionally long execution time. Giveaway gains two places on the Total Wins list on TBSE, which it also did on the no offset hierarchies. This is not because Giveaway starts posting better order prices, as it simply cannot do this, but is more likely caused by the more intelligent algorithms struggling to make as much profit on TBSE due to having a paused view of the market when calculating order prices, and therefore Giveaway's tactic which allows it to make an exceptionally high number of trades, even if they are at low profit, gives it an overall advantage on algorithms such as ZIP and GDX.

To finish the analysis of this result, two final hierarchies were created, one for each simulator. These were created by ranking each trader by how they are situated in each of the four hierarchies of each simulator, for example AA has the highest average position on all BSE hierarchies and TBSE hierarchies, therefore it gains the top place on both of the collated hierarchies. These hierarchies are shown in Table 4.8.

BSE	TBSE
AA	AA
Shaver	Giveaway
GDX	ZIP
Giveaway	Shaver
ZIC	GDX
ZIP	ZIC

Table 4.8: Overall hierarchies for BSE and TBSE created by aggregating each algorithms positions on the other hierarchies.

The first thing to note is that regardless of the simulator used AA remains the most dominant algorithm tested throughout this project. This is not an unexpected result, as both Vytelingum (2006)[18] and De Luca & Cliff (2011)[7] both support that AA dominates all other algorithms under simple conditions, such as the ones used in these experiments. Later research, such as Vach (2015)[16] and Snashall & Cliff (2019)[13], has suggested under more rigorous testing AA does not in fact dominate and can be beaten by both GDX and ZIP in certain circumstances, i.e. AA's dominance is not total.

Shaver's high performance in BSE is backed up by Cliff (2019)[3], which suggests that it is Shaver's parasitic nature of simply waiting for another trader to post an order and then beating that order by a single unit of currency, that allows it to generate such a high profit. The evidence collected in this

project further supports that, showing that by always having the best order on the LOB, Shaver can increase its likelihood of completing its customer order, and by completing more customer orders it can generate more profit than almost any other algorithm. On TBSE its performance diminishes, the reason for this is likely to be that, as TBSE allows traders to constantly submit orders, it is much harder for a Shaver trader to ensure that it has the highest bid/ask on the LOB. This is made even harder when the equilibrium price is moving as when the equilibrium price is fixed, over time intelligent traders will get a better idea of what the equilibrium price is, and therefore their order prices are likely to tend towards some value, when this is the case it is easy to simply place an order 1 unit better than that price and steal the deal. When the equilibrium price is moving however, they may place an order 1 unit better than the current best bid/ask, only to find that is immediately beaten by another trader as that trader has recalculated the equilibrium price to be moving and decided a more competitive price will improve its chances of trading.

The drop in performance by GDX on TBSE was an expected result, its has by far the longest execution time for its *getorder()* function, as shown in Table 4.1. When calculating its optimum order price, many other traders can post orders multiple times during the execution of that calculation, and that can alter the market in the meantime, making the result of the GDX trader's calculation outdated and potentially damaging to its profit margin.

As Giveaway is an entirely unintelligent trader it is surprising to see how well it performs on TBSE. The reason for this high performance is two-fold. Firstly, by always posting its limit price, it is always posting the most aggressive (i.e. likely to be accepted) price possible, this results in Giveaway consistently making more trades than the other algorithms, even if these are relatively low-profit trades, by executing enough of them Giveaway can generate more profit overall. The second reason, is less about the performance of Giveaway and more about the performance of the other algorithms. On TBSE Giveaway rises two places higher than it is on the BSE hierarchy, on TBSE it now beats Shaver and GDX, and in the paragraphs above it has already been discussed how the asynchronous TBSE simulator negatively impacts these traders in ways that would not affect Giveaway, hence making them perform more poorly relative to Giveaway.

The under-performance of ZIP on BSE contradicts previous research, particularly Cliff (1997)[4]. This may be as a result of the specific market conditions designed for these experiments putting ZIP at a particular disadvantage. By exploring some of the more exhaustive testing carried out for Snashall & Cliff (2019)[13], it is quite possible that it would be found that ZIP performs much better. On TBSE ZIP performs better, placing higher than ZIC, Shaver and GDX in the TBSE hierarchy. ZIP is also the only algorithm to beat AA in any test in this project, namely the TBSE offset test. This shows that ZIP performs well on TBSE, which is likely due to its fast execution time and also other algorithm's performance being hampered by TBSE.

ZIC performs poorly when ran on both simulators, this is not surprising. Unlike Shaver and Giveaway, the other two unintelligent algorithms in this project, ZIC's algorithm does not guarantee that it will be able to make more trades than its opposing algorithm, as its order prices are completely random, which means it is possible that a large percentage of the time it produces order prices which will not be accepted, thus making it very hard for ZIC to reliably make a good profit.

4.4 Relevance to Real Markets

TBSE was designed to be a more accurate simulation of a financial exchange market than BSE. It did this by using multi-threading to simulate traders acting asynchronously which allowed the execution times of algorithms to affect how they performed. It also meant that trades could occur and new orders be posted on the market, whilst a trader was calculating its order price, and this means that the trader would not see these changes and not be able to use them in its calculation. These are all things that happen on real financial exchange, so TBSE is indeed a better model for a financial exchange than BSE. It is not however a perfect model for a financial exchange nor were the experiments that were used here perfectly representative of the type of markets that occur in real life. Below are detailed some of the ways in which these current experiments fall short.

4.4.1 TBSE

Although TBSE is multi-threaded, there is only a single thread running the exchange. When there are forty traders on the market, there could be up to forty orders being posted to the exchange simultaneously, which would cause the exchange to be overwhelmed and create a backlog of orders which would result in

the LOB being out of date, almost constantly. The way this problem was fixed on TBSE was to introduce a sleep-wake pattern to the traders. This pattern means that they sleep for 0.1 seconds between orders, this slows them down enough that the exchange can keep up with the traders, but as all traders are given the same amount of sleep time allows the testing to remain fair.

An aspect of real financial markets which cannot be simulated by TSBE is *latency*. In the real world, traders do not execute on the same computer as the exchange, they are on separate machines that are connected to a network. When a real trader posts an order it takes time for that order to travel along the wires that connect the trader to the exchange and it takes time for the exchange to send market information back to the trader, this time delay is called latency. In high frequency trading this latency can have a great affect on trader's ability to make a profit so trading firms spend vast amounts of money to try and reduce this latency, however it will always be present to some degree.

For these experiments a fixed number of forty traders were used. This number was chosen as it was close to, but not beyond, the limit of how many traders the exchange can handle before timing problems begin to appear. In reality there would be a far greater number of traders acting on the same market. The number of traders active on different exchanges can also vary massively, so being fixed at forty is a simplification. Ways in which this could be simulated in the future are included in Section 5.3.

4.4.2 Experiments

As well as TBSE having limitations that prevent it from being a truly accurate model for a financial exchange there were also simplifications within the experiments that were run, one of which has already been mentioned, being the number of traders in the market being fixed at forty. Another simplification is that, although this project tested six different strategies which is on the upper end of how many strategies have been tested together in previous research, in reality there are probably many more strategies than this acting in real-world markets, many of which may never be known about, as traders will want to keep their profitable strategies a secret.

These tests were also run in fixed market scenarios which do not replicate real world markets perfectly. The first set of experiments that used no offset is a good example of this, in reality the equilibrium price would never remain static and would constantly be moving unpredictably with changes in the supply and demand for the asset being traded. The tests that used a variable offset better modelled this, however this still is not perfect as equilibrium prices can move in a vast variety of ways. Again how this could be improved on is discussed in Section 5.3.

Chapter 5

Conclusion

5.1 Summary

The aim of this project was to develop a multi-threaded financial exchange simulator TBSE, based on Cliff's Bristol Stock Exchange[2] and then use this simulator to compare well-known leading public-domain trading algorithms to see if the currently accepted hierarchy of algorithm performance is maintained or altered when introduced to a multi-threaded exchange environment, where the run-time of algorithms can affect their performance, an aspect of these algorithms which has never been tested before. The research hypothesis was that slower executing algorithms would perform less well on TBSE than on BSE, due to the market changing whilst they are calculating their order price, something that cannot happen on BSE as each trader is given as much time as they need to complete their transaction, with nothing else occurring on the market in this time, essentially pausing the activity of the market for as long as it takes the trader to compute a response.

TBSE was successfully developed and shown to accurately model a financial exchange, described in Section 2.3. Experiments were then designed to test six different trading algorithms, of varying complexity, namely: AA, GDX, ZIP, ZIC, Giveaway and Shaver, details of these algorithms were given in Section 2.4. These experiments involved comparing these algorithms two at a time, and by varying the amount of traders that were running each algorithm, so that it could be observed how the ratio of the two algorithms affected the performance of each algorithm. Twenty traders were used on either side of the market (i.e. 20 buyers and 20 sellers), and the ratio of A:B was varied from 1:19 through to 19:1, with 1000 experiments being conducted at each ratio, broken down into 10 sets of randomly-generated symmetric order schedules tested 100 times each. This resulted in 19,000 tests being run per pair of algorithms, of which there were 15 pairs, resulting in a total of 285,000 tests being run per experiment, per simulator (tests were run on both BSE and TBSE for comparison). Two styles of experiments were used, firstly a simple experiment, where the competitive equilibrium price was static throughout the tests, and secondly a test where a variable offset, shown in Figure 3.7, was applied to alter the competitive equilibrium price throughout each test. This test was designed to see how well the algorithms managed to track a moving competitive equilibrium price and extract profit from this style of market. As a result 1,140,000 tests have been recorded and analysed in this project.

The results of these experiments confirmed that the slowest of these algorithms, GDX, did indeed perform significantly worse on TBSE than BSE, this supported the research hypothesis. Other results however showed that the research hypothesis was not entirely correct, with the second slowest algorithm, AA, still maintaining its dominance over the other five algorithms. It should be noted however that GDX is over 100 times slower than all other algorithms, and AA is less than three times slower than the fastest executing algorithms (Giveaway and Shaver), discussion of algorithm run-times was given in Section 4.2. This indicates that although the research hypothesis may be correct, it may only apply to large differences in execution time, and algorithms with similar execution times are less drastically affected by operating on an asynchronous exchange simulator. The results also revealed that some of the most simple algorithms, Giveaway and Shaver, could perform very well by simply ensuring that they completed as many of their customer orders as possible, even if the profit received from those trades was not as high as algorithms such as AA, ZIP or GDX could acquire. More details about the results were given in Section 4.3.

Whilst considering the affects of varying the ratios between algorithms, the concept of Equal-Availability Ratio (EAR) was developed. EAR is a calculation of the ratio of experiments that would be won by two

trading groups (A and B) running the same "perfect" algorithm that always results in trades being executed at the competitive equilibrium price, the most efficient price for the market, when the ratio of A:B is varied. This resulted in discovering that when A or B is in the extreme position of only having a single trader on either side of the market, with the rest of the traders being from the opposing group, that trader is at a natural disadvantage simply due to the way limit prices are distributed in order to form the supply and demand curves needed for a functioning market. This called into question whether previously recorded results of "1-in-Many" experiments, where there is a single trader of one type in a market otherwise entirely populated by traders running a different algorithm, were in fact true representations of each algorithm's performance.

5.2 Project Status

In Section 1.7 six goals for this project were stated. Below, a summary of the status of each of these goals is listed.

1. Research revealed varied results of which algorithms are considered best. Early work looking at the AA algorithm, such as Vytelingum (2006)[18] and De Luca & Cliff (2011)[7] suggested that AA was the dominant strategy, this was then disputed in Vach (2015)[16] and Snashall & Cliff (2019)[13]. Results from experiments on BSE were used to determine the hierarchy which were back up by the work of Vytelingum and De Luca & Cliff, the more recent papers used more thorough methods of comparison, not repeated in this project so it was declared that AA the best strategy on BSE for the purposes of this project.

2. Cliff's Bristol Stock Exchange (BSE) simulator was successfully extended to use multi-threading, which allowed for trading algorithms to execute concurrently, with their execution times impacting performance. This simulator was named the Threaded Bristol Stock Exchange (TBSE).

3. Six trading algorithms were adapted to run on TBSE, and a bug present in the version of GDX that was used in Snashall & Cliff (2019)[13] was fixed. Some doubts over the implementation of GDX that were first discussed in Snashall & Cliff (2019)[13] paper remain.

4. Extensive testing of the six algorithms, on both BSE and TBSE was carried out. The exhaustive kind of testing that was performed for Snashall & Cliff (2019)[13] would have taken too much CPU time on AWS to be financially viable, so instead it was decided to draw inspiration from their testing but develop unique tests for this project. Instead of testing all six algorithms together at every permutation of those algorithms, (as Snashall & Cliff did with four algorithm) every permutation of two algorithms were tested. It would have been desirable to perform the more exhaustive style of testing, and this is discussed in Section 5.3.

5. The results from the testing mentioned above were successfully analysed in Section 4.3. Notable changes in the established BSE hierarchy were found and so a new hierarchy was established, which can be found in Table 4.8.

6. My code has been made publicly available in the following GitHub repository: `https://github.com/MichaelRol/Threaded-Bristol-Stock-Exchange`.

7. A paper summarising early results from this project was submitted to, and accepted for oral presentation at, a peer-reviewed international conference (EMSS2020, Athens, October 2020). A copy of that paper is in Appendix A.

5.3 Further Work

Throughout this project opportunities for further work arose that were not possible to be conducted within the limits of time and resources allotted to this project. Therefore in this section these ideas are introduced and discussed.

5.3.1 Extensive Algorithm Testing

Three or More Algorithm Experiments

The exhaustive testing described in Snashall & Cliff (2019)[13] was determined not to be within the limits of this project, for six algorithms the combinatorics of dividing them across 20 or more traders would result in so many tests that it would not be feasibly possible to test them all within time and expense deemed reasonable for this project. However if possible, this style of testing would be desirable, as the work of Snashall & Cliff (2019)[13] showed that AA was not dominant in all cases, which suggests that further variation of the combination of algorithms in each market may provide further insight into which algorithms truly are *best*. Section 4.4 discussed how in real markets there would be far greater than two algorithms competing, therefore by increasing the number of algorithms acting in the market at one time the experiments would better represent real financial markets.

Market Variation

Another detail of the experiments performed for Snashall & Cliff (2019)[13] that could be explored would be to run more experiments with greater variation in offsets and market shocks. This project elected to use an offset that varied over time to represent the constantly changing nature of a real financial market, but in reality the competitive equilibrium price does not move in a predefined way. Therefore, it would be of value to explore different offset types, such as the ones used in Snashall & Cliff (2019)[13], like square waves, saw-tooth offsets and sinusoidal. It would be interesting to also use a constrained random offset, so that the equilibrium price is entirely unpredictable. Another feature of real financial markets that was not explored in this project is market shock, where there is a sudden sharp increase or decrease in the equilibrium price, therefore once again it would be interesting to run tests comparing if algorithms respond differently to market shock on BSE verses TBSE.

A simplification of these experiments was using symmetric order schedules, where the demand and supply curves had the same maximum and minimum value, and as a result their theoretical equilibrium price was the midpoint between these values. This was done as it allowed for simple calculation of the EAR. The maximum and minimum values for these were randomly generated in order to prevent a fixed set of order schedules giving certain algorithms an advantage. Further research could repeat the experiments of this project but have the demand and supply curves generated both randomly and separately to increase the variation in market dynamics. In order to continue comparing the results to the EAR, the EAR would need to be calculated repeatedly for every set of supply and demand curves, thus increasing the computational requirements for these experiments. However, by doing this it would also give the opportunity for different performance measurements to be considered as well, as the equilibrium price of the market will already need to be calculated for the EAR. Details of the different kind of performance measurements that can be used when comparing algorithmic traders can be found in Section 2.2.2.

Although selecting a range of random supply and demand curves prevents certain algorithms from gaining an advantage based on their ability to act on a specific order schedule, the potential for certain algorithms to have an advantage is worth researching in itself. Therefore it could be of value to generate a large set of supply and demand curves and then run identical market experiments on all of them, in order to identify patterns in these supply and demand curves that advantage certain algorithms.

5.3.2 High-Performance Computing

Out of all the research that could be done in this area, I believe the most valuable would to be to create a High-Performance Computing (HPC) implementation of a financial exchange. HPC is the practice of using a large cluster of computers to distribute vast quantities of computation over. HPC is already used for many simulation tasks, such as weather forecasting and even drug interaction simulation. By using HPC the magnitude of the number of traders acting upon a simulated exchange could increase dramatically.

One of the major drawback of the current implementation of TBSE is that there is fixed limit on the number of threads running simultaneously. This limits the number of traders that can be on the market at the same time, and forces TBSE to introduce a sleep-wake cycle to prevent the exchange from becoming overwhelmed. If this was to be to implemented on an HPC system, each trader could run on its own computational node and the exchange could be designed to work itself across multiple nodes to allow it to keep up with the demand from traders. Another drawback discussed in Section 4.4 was that latency could not be explored. If physical knowledge of the HPC system being used was available it would

be possible to vary the distance between trader nodes and the exchange nodes (which would ideally be located together in a local cluster), in order to model the affects of latency.

This concept could allow for hundreds or potentially thousands of traders to all interact on the same simulated exchange. This could then be used to recreate known patterns of market activity from real financial exchanges and from that it may be possible to draw conclusions about the make up of the type of algorithms that are used on that financial exchange.

Bibliography

[1] Navarra A. Bigiotti A. Optimizing automated trading systems. *Digital Science. DSIC18 2018. Advances in Intelligent Systems and Computing,*, 850:254–261, 10 2018. https://doi.org/10.1007/978-3-030-02351-5_30.

[2] Dave Cliff. Bristol stock exchange, 2012. https://github.com/davecliff/BristolStockExchange.

[3] Dave Cliff. Exhaustive testing of trader-agents in realistically dynamic continuous double auction markets: Aa does not dominate. *Proceedings of the 11th International Conference on Autonomous Agents and Artificial Intelligence (ICAART 2019)*, 2:224–236, March 2019. https://doi.org/10.5220/0007382802240236.

[4] Dave Cliff and Janet Bruten. Zero is not enough: On the lower limit of agent intelligence for continuous double auction markets. 1997. https://www.researchgate.net/publication/2826931_Zero_is_Not_Enough_On_The_Lower_Limit_of_Agent_Intelligence_for_Continuous_Double_Auction_Markets.

[5] U.S. Securities & Exchange Commission & U.S. Commodity Futures Trading Commission. Findings regarding the market events of May 6, 2010, 2010. https://www.sec.gov/news/studies/2010/marketevents-report.pdf.

[6] Rajarshi Das, James E. Hanson, Jeffrey O. Kephart, and Gerald Tesauro. Agent-human interactions in the continuous double auction. *Proceedings of the 17th International Joint Conference on Artificial Intelligence - Volume 2*, pages 1169–1176, 2001. https://dl.acm.org/doi/10.5555/1642194.1642251.

[7] Marco De Luca and Dave Cliff. Human-agent auction interactions: Adaptive-aggressive agents dominate. *IJCAI International Joint Conference on Artificial Intelligence*, pages 178–185, 01 2011. https://doi.org/10.5591/978-1-57735-516-8/IJCAI11-041.

[8] Steven Gjerstad and John Dickhaut. Price formation in double auctions. *Games and Economic Behaviour*, 22(1):1 – 29, 1998. https://doi.org/10.1006/game.1997.0576.

[9] Dhananjay K. Gode and Shyam Sunder. Allocative efficiency of markets with zero-intelligence traders: Market as a partial substitute for individual rationality. *Journal of Political Economy*, 101(1):119–137, 1993. https://www.doi.org/10.1086/261868.

[10] Vijay Krishna. *Auction Theory*. Elsevier, 1st edition, 2002.

[11] Rob Schapire. Theoretical machine learning, 2013. https://www.cs.princeton.edu/courses/archive/spring13/cos511/scribe_notes/0411.pdf.

[12] Vernon Smith. An experimental study of competitive market behavior. *Journal of Political Economy*, 70:322–322, 02 1962. https://doi.org/10.1086/258609.

[13] Daniel Snashall and Dave Cliff. Adaptive-aggressive traders don't dominate. *Agents and Artificial Intelligence*, pages 246–269, 10 2019. https://arxiv.org/abs/1910.09947.

[14] Gerald Tesauro and Jonathan L. Bredin. Strategic sequential bidding in auctions using dynamic programming. *Proceedings of the First International Joint Conference on Autonomous Agents and Multiagent Systems: Part 2*, pages 591–598, 2002. https://doi.org/10.1145/544862.544885.

[15] Gerald Tesauro and Rajarshi Das. High-performance bidding agents for the continuous double auction. *Proceedings of the 3rd ACM Conference on Electronic Commerce*, pages 206–209, 2001. https://doi.org/10.1145/501158.501183.

[16] Daniel Vach. Comparison of double auction bidding strategies for automated trading agents. 2015. Masters Thesis, Charles University, Prague. https://is.cuni.cz/webapps/zzp/detail/152184/?lang=en.

[17] P. Vytelingum, D. Cliff, and N.R. Jennings. Strategic bidding in continuous double auctions. *Artificial Intelligence*, 172(14):1700 – 1729, 2008. https://doi.org/10.1016/j.artint.2008.06.001.

[18] Perukrishnen Vytelingum. *The Structure and Behaviour of the Continuous Double Auction*. PhD thesis, University of Southampton, December 2006. https://eprints.soton.ac.uk/263234/.

[19] The World Bank Group (WBG). Market capitalization of listed domestic companies (current US$), 2018. https://data.worldbank.org/indicator/CM.MKT.LCAP.CD?end=2018&start=1975.

Appendix A

Paper to be presented at the 32nd European Modelling & Simulation Symposium (EMSS-2020)

32nd European Modeling & Simulation Symposium
17th International Multidisciplinary Modeling & Simulation Multiconference

2724-0029 © 2020 The Authors.
doi: xx.xxxx/xxxx

Which Trading Agent is Best? Using a Threaded Parallel Simulation of a Financial Market Changes the Pecking-Order

Michael Rollins and Dave Cliff*

Department of Computer Science, The University of Bristol, Woodland Road, Bristol, BS8 1UB, UK

*Corresponding author. Email address: csdtc@bristol.ac.uk

Abstract

This paper presents novel results, generated from a new simulation model of a contemporary financial market, that cast serious doubt on the previously widely accepted view of the relative performance of various well-known public-domain automated-trading algorithms. Put simply, we show here that if you use a more realistic market simulator, then trading algorithms previously thought to be the best-performing are shown to be not as good as people think they are, and some algorithms previously thought to be poor performers can be seen to do surprisingly well. Automated trading is now entirely commonplace in most of the world's major financial markets: adaptive algorithmic trading systems operate largely autonomously, interacting with other traders (either other automated systems, or humans) via an electronic exchange platform. Various public-domain trading algorithms have been proposed over the past 25 years in a kind of arms-race, where each new trading algorithm was compared to the previous best, thereby establishing a "pecking order", i.e. a partially-ordered dominance hierarchy from best to worst of the various trading algorithms. Many of these algorithms were developed, tested, and evaluated using simple minimal simulations of financial markets that only very weakly approximated the fact that real markets involve many different trading systems operating asynchronously and in parallel. In this paper we use BSE, a long-established public-domain market simulator, to run a set of experiments generating benchmark results from several well-known trading algorithms. BSE incorporates a very simple time-sliced approach to simulating parallelism, which has obvious known weaknesses. We then alter and extend BSE to make it *threaded*, so that different trader algorithms operate asynchronously and in parallel: we call this simulator *Threaded-BSE* (TBSE). We then re-run the trader experiments on TBSE and compare the TBSE results to our earlier benchmark results from BSE. Our comparison shows that the dominance hierarchy in our more realistic experiments is different from the one given by the original simple simulator. We conclude that simulated parallelism matters a lot, and that earlier results from simple simulations comparing different trader algorithms are no longer to be entirely trusted.

Keywords: Financial Markets; Market Simulators; Simulation Methods; Trading Agents; Experimental Economics.

1. Introduction

This paper presents new evidence that key results previously published in the literature on automated trading systems may be artefacts of the simplistic modelling and simulation methods used to generate them. In particular, we have taken a long-established public-domain simulator of a contemporary electronic financial market, called BSE, and used it to generate a set of benchmark results from four very widely-used public-domain automated-trading algorithms, known as ZIC, ZIP, GDX, and AA (we explain these in more detail, later). These initial BSE benchmark results confirm the previously-published dominance-hierarchy, or "pecking order", where AA beats GDX, GDX beats ZIP, and ZIP beats ZIC, which we write in abbreviated form as AA>GDX>ZIP>ZIC. The BSE simulator is deliberately simple in its modelling of time, and uses only a single thread, a single computer process, which it divides among the various simulated automated traders in such a way that the response-time, the time for an automated trader to compute an answer, is not simulated and hence very complicated trading systems that would take a lot of "thinking time" to compute a response to some change in the market are treated as if they take exactly the same time as the very fastest of trading algorithms. This is clearly unrealistic, yet BSE's approach to simulating time is not uncommon.

We then constructed a modified version of the BSE simulator, which is *threaded*, i.e. uses multiple concurrently running computer processes ("threads") to model the parallel and

asynchronous nature of real financial markets, where multiple traders operate independently, in parallel and asynchronously: we call this simulator Threaded-BSE (TBSE). We then re-run our experiments on TBSE and find, as we show here for the first time, that the dominance hierarchy in TBSE is very different from that found in BSE: from our current results it seems that when operating in parallel the dominance hierarchy is instead ZIP>AA>ZIC>GDX. This is a novel result and is the key contribution of this paper. That is, we demonstrate here that previous use of simplistic simulation methodologies has produced sets of results which, although widely cited, can now no longer be trusted.

Section 2 of this paper gives a summary of the background to this work, and is taken largely verbatim from a position paper published by one of us at EMSS2019 (Cliff, 2019): our new results presented here can be read as an empirical illustration of the arguments made in that EMSS2019 paper. Section 3 gives more details of our methods; Section 4 shows key results; and Section 5 offers a discussion of the results, and of possible avenues of further work. TBSE is fully documented in (Rollins, 2020), including details of the TBSE source-code repository on GitHub.

2. Background

2.1. Traders, Markets, and Experimental Economics

The 2002 Nobel Prize in Economics was awarded to Vernon Smith, in recognition of Smith's work in establishing and thereafter growing the field of *Experimental Economics* (abbreviated hereafter to "EE"). Smith showed that the microeconomic behavior of human traders interacting within the rules of some specified market, known technically as an auction mechanism, could be studied empirically, under controlled and repeatable laboratory conditions, rather than in the noisy messy confusing circumstances of real-world markets. The minimal laboratory studies could act as useful proxies for studying real-world markets of any type, but one particular auction mechanism has received the majority of attention: the *Continuous Double Auction* (CDA), in which any buyer can announce a bid-price at any time and any seller can announce an offer-price at any time, and in which at any time any trader in the market can accept an offer or bid from a counterparty, and thereby engage in a transaction. The CDA is the basis of major financial markets worldwide, and tens of trillions of dollars flow through CDA markets every year.

Each trader in one of Smith's experimental CDA markets would be assigned a private valuation, a secret *limit price*: for a buyer this was the price above which he or she should not pay when purchasing an item; for a seller this was the price below which he or she should not sell an item. These limit-price assignments model the client orders executed by sales traders in real financial markets; we'll refer to them just as *assignments* in the rest of this paper. Traders in EE experiments from Smith's onwards are often motivated by payment of some form of real-world reward that is proportional to the amount of "profit" that they accrue from their transactions: the profit is the absolute value of the difference between the limit price specified when a unit is assigned to a trader, and the actual transaction price for that unit.

The limit prices in the assignments defined the market's supply and demand schedules, which are commonly illustrated in economics texts as supply and demand curves on a 2D graph with quantity on the horizontal axis and price on the vertical axis: where the two curves intersect is the market's theoretical competitive *equilibrium point:* indicating the *equilibrium price* (denoted here by P_0). A fundamental observation from microeconomics (the study of markets and prices) is that competition among buyers pushes prices up, and competition among sellers pushers prices down, and these two opposing influences on prices balance out at the competitive equilibrium point; a market in which transaction prices rapidly and stably settles to the P_0 value is often viewed by economists as *efficient* (for a specific definition of efficiency) whereas a market in which transactions consistently occur at off-equilibrium prices is usually thought of as inefficient: for instance, if transaction prices are consistently above P_0 then it's likely that buyers are being ripped off. By varying the prices in the traders' assignments in Smith's experiments, the nature of the market's supply and demand curves could be altered, and the effects of those variations on the speed and stability of the market's convergence toward an equilibrium point could be measured.

Smith's initial set of experiments were run in the late 1950's, and were described in his first paper on EE, published in the prestigious *Journal of Political Economy* (JPE) in 1962. The experiment methods laid out in that 1962 paper would subsequently come to dominate the methodology of researchers working to build adaptive autonomous automated trading agents by combining tools and techniques from Artificial Intelligence (AI) and Machine Learning (ML). This strand of AI/ML research converged toward a common aim: specifying an artificial agent, an autonomous adaptive trading strategy, that could automatically tune its behavior to different market environments, and that could reliably beat all other known automated trading strategies, thereby taking the crown of being the current best trading strategy known in the public domain, i.e., the "dominant strategy". Over the past 20 years the dominant strategy crown has passed from one algorithm to another and until very recently Vytelingum's (2006, 2008) "AA" strategy, was widely believed to be the dominant strategy, but recent results using contemporary large-scale computational simulation techniques indicate that it does not always perform so well as was previously believed, as discussed in the next section: Section 2.2 briefly reviews key publications leading to the development of AA, and the recent research that called its dominance into question.

2.2. A Brief History of Trading Agents

If our story starts with Smith's 1962 JPE paper, then the next major step came 30 years later, with a surprising result published in the JPE by Gode & Sunder (1993): this popularized a minimally simple automated trading algorithm now commonly referred to as *ZIC*. A few years later two closely related research papers were published independently and at roughly the same time, each written without knowledge of the other: the first was a Hewlett-Packard Labs technical report by Cliff (1997) describing the adaptive AI/ML trading-agent strategy known as the *ZIP* algorithm; the second summarized the PhD thesis work of Gjerstad, in a paper co-authored with his PhD advisor (Gjerstad & Dickhaut 1998), describing an adaptive trading algorithm now widely known simply as *GD*. After graduating his PhD, Gjerstad worked at IBM's TJ Watson Labs where he helped set up an EE laboratory that his IBM colleagues used in a study that generated world-wide media coverage when its results were published by Das *et al.* at the prestigious *International Joint Conference on AI* (IJCAI) in 2001. This paper presented results from studies exploring the behavior of human traders interacting with GD and ZIP robot traders, and demonstrated that both GD and ZIP reliably outperformed human

traders. A follow-on 2001 paper by Tesauro & Das (two co-authors of the IBM IJCAI paper) described a more extensively *Modified GD* (MGD) strategy, and later Tesauro & Bredin (2002) described the *GD eXtended* (GDX) strategy. Both MGD and GDX were each claimed to be the strongest-known public-domain trading strategies at the times of their publication.

Subsequently, Vytelingum's 2006 thesis introduced the *Adaptive Aggressive* (AA) strategy which, in a major journal paper (Vytelingum *et al.*, 2008), and in later conference papers (De Luca & Cliff 2012a, 2012b), was shown to be dominant over ZIP, GDX, and human traders. Thus far then, ZIP had been beaten by GDX, and AA had beaten GDX, and hence AA held the title. In shorthand, we had AA>GDX>ZIP.

In all of the studies discussed thus far, typically two or three different types of trading algorithm would be compared against each other on the basis of how much profit (or *surplus*, to use the economists' technical term) they extract from the market, so Algorithm A was said to *dominate* or *outperform* or *beat* or *be stronger than* Algorithm B if, over some number of market sessions, traders running A made more money than traders running B. Methods of comparison varied. Sometimes a particular market set-up (i.e., a specific number of sellers, number of buyers, and their associated limit-price assignments specifying the market's supply and demand schedules) would be homogeneously populated with traders of type A, and then the same market would be re-run with all traders instead being type B, and an A/B comparison of profitability *in the absence of any other trading algorithms* could then be made. In other comparisons, for a market with *B* buyers and *S* sellers, B/2 of the buyers would use Algorithm A and the remaining B/2 buyers would run Algorithm B, with the seller population being similarly split, and the A/B comparison then showed profitability *in the presence of the other trading algorithm*. A/B tests involving 50:50 splits, as just described, were commonly used to establish the dominance relationship between A and B.

Comparatively recently, Vach (2015) presented results from experiments with the *OpEx* market simulator (De Luca, 2015), in which AA, GDX, and ZIP were set to compete against one another, and in which the dominance of AA was questioned: Vach's results indicate that whether AA dominates or not can be dependent on the ratio of AA:GDX:ZIP in the experiment: for some ratios, Vach found AA to dominate; for other ratios, it was GDX. Vach studied only a very small sample from the space of possible ratios, but his results prompted Cliff (2019) to use the public-domain "BSE" financial exchange simulator (BSE, 2012) to exhaustively run through a wide range of differing ratios of four trading strategies (AA, ZIC, ZIP, and the minimally simple SHVR built into BSE), doing a brute-force search for situations in which AA is outperformed by the other strategies. Cliff reported on results from over 3.4 million individual simulations of market sessions, which indicated that Vach's observation was correct: whether AA dominates does indeed depend on how many other AA traders are in the market, and what mix of what other strategies are also present. Depending on the ratio, AA could be outperformed by ZIP and by SHVR. Subsequent research by Snashall (2019) employed the same exhaustive testing method, using a supercomputer to run more than one million market simulations (all in BSE) to exhaustively test AA against IBM's GDX strategy: this again revealed that AA does not always dominate GDX: see Snashall & Cliff (2019) for discussion.

In this paper we will talk about counting the number of "wins" when comparing an A algorithm to a B algorithm: in the experiments reported in Section 4, we create a specific market set-up, and then run some number *n* of independent and identically distributed markets sessions with some ratio A:B of the two strategies in the buyers, and the same A:B ratio in the sellers. In any one of those sessions, if the average profit per trader (APPT) of type A traders is higher than the APPT for traders of type B, then we count that session as a "win" for A; and *vice versa* as a win for B. In the experiments reported in Section 4, assignments to buy or sell are issued to the traders periodically, with all traders being updated at the same time, and the limit prices in the assignments came from symmetric supply and demand curves where the equilibrium price was varied dynamically using the sinusoidal offset function illustrated in Fig.4.8 of the BSE user guide (BSE, 2012).

Our experiments reported here are motivated by this progression of past research. In particular, we noted that Vach's results which first revealed that the ratio of different trading algorithms could affect the dominance hierarchy came from experiments he ran using De Luca's (2015) OpEx market simulator, which is a true parallel asynchronous distributed system: OpEx involves a number of individual trader computers (discrete laptop PCs) communicating over a local-area network with a central exchange-server (a desktop PC). But many of the other results that we have just summarized came from financial-market simulators that only very roughly approximated parallel execution: Cliff (1997) published C-language source-code for the discrete-event simulator he developed to test and compare ZIC with ZIP; and the BSE simulator (BSE, 2012) also uses a very simple time-sliced approach where, if any one trader is called upon to issue a response to a change in the market, it always does so within exactly one simulated time-slice within the simulation, regardless of how much computation it has to execute to generate that response. Snashall & Cliff (2019) noted that the actual reaction-times of the various trading algorithms varied quite widely, and in a true parallel simulation the slower traders might be expected to do much less well than when they are evaluated or compared in a temporally simplistic simulation. So, possibly Vach's results were as much to do with his use of OpEx as his varying of the ratios of trader-algorithms. That is what we set out to explore in this paper. In order to do that, we developed TBSE, a new threaded (parallel) version of the BSE financial-market simulator, discussed in the next section.

3. TBSE: Threaded BSE

Much of the functionality of TBSE was directly borrowed from the original BSE simulator. The key changes affect the way each market session operates. In BSE, each market session is executed on a single thread in which a core loop executes repeatedly until the session end-time is passed. Within this loop a single trader in the market's population of traders is selected at random, and that trader is polled to see if it wants to issue an order: this is achieved by calling that trader's `getorder` function. If the trader does issue an order, that order is then processed by the exchange and all traders in the population are notified of any resultant change in the market-data published by the exchange. At that point, all traders in the population are given an opportunity to update their internal values with the new information, via a function called `respond` that is particular to each specific trading algorithm (that is, `respond` is where much, perhaps all, of the detail of the trading algorithm is encoded); and then control loops back to another random selection of a single trader to be polled for an order. A single pass through this sequence of steps is implied to take a notional time of $1/N$ seconds, where N is the total number of traders in the market: in this way, each trader will on average get a

chance to issue an order once per simulated second, and will call respond *N* times per simulated second. But if Trader A's `respond` takes 1ms of real-time to execute, and trader B's `respond` takes 10s of real-time to compute an answer, BSE's record of simulated time is the same in both cases: the simulated clock advances in increments of 1/*N* seconds, regardless of the wall-time consumed by the differing `respond` functions.

The problem with this style of simulation is that each trader is allotted as much time as it needs to execute its `getorder` and `respond` functions, and it is guaranteed that nothing else in the market will change while it executes either of these functions. This means that the execution time of each trader has no impact on its performance: instead all that counts is its ability to generate an order price that will be accepted by another trader, whilst attempting to generate the greatest profit. This differs from a real financial exchange where all traders operating on the market will be operating asynchronously. In an asynchronous market, a trader may look at the exchange's market data, i.e. the currently available information on that market, and use this to calculate what it considers its best order to send to the exchange. However, if this trader uses a complex, slow running algorithm to do so, it may find that by the time it has completed its calculation, another simpler and faster trader has already succeeded in executing a trade, which as a result changes the position of the market which may render the order than the complex trader has posted less profitable than expected. This situation, which happens all the time in real-world financial markets, cannot be modelled in BSE.

To address this, we created an asynchronous threaded exchange simulator, TBSE. BSE was written in the *Python* programming language, and so we used that for TBSE too. In TBSE each trader executes on its own computational thread, as does the exchange. There is also the main thread of the Python code which continues to run during the market session and is responsible for the distribution of customer orders to each of the traders. Each trader thread consists of a loop which executes continuously until the end of the trading session. Within this loop the trader first receives information of trades which have been successfully executed at the exchange, it then updates its own internal record keeping if any of these trades involved itself and executes its `respond` function to update its internal variables based on the new market data. Once it is updated with the latest market data it then executes its `getorder` function which determines whether it should post a new order to the exchange, and if so at what price. This order is then put on a queue to be sent to the exchange.

The exchange operates on this queue, reading orders from the queue and then processing them by either adding them to the its list of available orders or executing a trade if the new order can be matched with any of its current list of available orders. If a trade can be executed it then places the details of the resultant transaction onto a queue for each trader to read before submitting its next order. The system for processing orders and trades is almost identical to that of BSE.

In Python, as with other languages that offer such a feature, *multi-threading* is where a processor can operate multiple threads of execution concurrently. This differs from multi-processing, which Python also supports, where different processes execute simultaneously. Both are forms of parallelism, but whereas in multi-processor programming multiple processes are executing at the same time on different physical processors, in multithreading only one thread can execute at any one time. The execution of each thread is divided into short segments and the processor regularly switches between each thread for a short period of time, giving the appearance that all threads are executing at the same time. It is not guaranteed that each thread will be given the exact same amount of execution time during each cycle, but over the course of executing an entire program, which may involve millions of cycles, the execution time given to each thread should tend towards the same value. In TBSE this means that each trader is given the same amount of processing time, so a faster algorithm can complete its execution and start processing a second order while a slower algorithm is still calculating its first order price. In Python, the execution of threads is synchronized by a Global Interpreter Lock (GIL) which ensures that only one thread is executing at a time. Multi-threading was chosen over multi-processing as most general-purpose CPUs have no more than 8 processing cores, but for our experiments we want to simulate the parallel execution of many more than 8 traders operating simultaneously.

The queues used within TBSE are synchronous first in, first out (FIFO) queues, meaning that the trader who places an order on the exchange queue first will have their order processed first. This is a good simulation of real exchange behavior as it is commonplace for exchanges to prioritize orders by their time of arrival.

4. Results

Table 1 shows a high-level summary of our results for AAvsZIC, AAvsZIP, GDXvsZIC, and GDXvsZIP. Characterizing each of these four experiments as Algorithm A vs Algorithm B, the central sub-table of Table 1 shows the count of "wins" for A and the count of wins for B in BSE, with the higher of the two counts highlighted in bold font; and then the right-hand sub-table of Table 1 shows the corresponding win-counts for A and for B operating instead in TBSE, again with the higher value highlighted in bold font.

As can be seen from Table 1, our results for BSE are consistent with those previously published in the literature: there is nothing in our BSE results to challenge the status quo: AA>ZIC & ZIP; GDX>ZIC&ZIP. However, the TBSE results in Table 1 tell a very different story. AA still beats ZIC, which is intuitively what one would expect. But the other three dominance relationships have been inverted: in TBSE, ZIC beats GDX while ZIP beats both AA and GDX – scoring more than twice as many wins as GDX, beating it by a larger margin than it beat ZIP in BSE.

If we chose, we could stop here. We have now established an answer to the question that we set out to explore: whether a switch to a more realistic (i.e., parallel) market simulation makes a difference to the dominance relationships previously reported: it clearly does. This is a significant finding and is the primary contribution of this paper. However, there is more to say.

Each number in Table 1 represents the number of "wins" scored by a specific algorithm, in a series of 19x500 market sessions, ranging over 19 ratios of TraderA:TraderB (denoted by R_a: ranging here from 1:19 to 19:1), with n=500 trials at each value of R_a. One obvious thing to do is to separate the aggregate results for any one pair of trader algorithms into the 19 sets of results, one for each value of R_a, and to look for any interesting changes in the patterns of wins as R_a is swept from one extreme to another. Table 2 shows such a data-set, for AA-vs-ZIC. As you can see, at the bottom of the table is the sum of all the wins in each column, and the relevant column-sums in Table 2 are the values that populate the first row of Table 1. Table 2 adds a third column to the results for BSE and TBSE: a variable we call $\Delta wins$, which is simply the difference between the two algorithm's win-counts. In an A-vs-B comparison

of two trading algorithms: if $\Delta wins>0$ then A outperforms B; and if $\Delta wins<0$ then B outperforms A. The two sets of $\Delta wins$ values shown in Table 2 can be plotted graphically in the style shown in Figure 1, a paired plot that we refer to as the *delta curves* for two trading strategies tested in BSE and then in TBSE.

Table 1. Summary of all our experiment results, showing total number of "wins" summed over 19 different A-vs-B trials in BSE (central sub-table) and in TBSE (right-hand sub-table). The 19 different trials vary the ratio R_a of trading algorithm A ("AlgoA") to trading algorithm B ("AlgoB") from 1:19 through 10:10 to 19:1, and at each ratio we conduct n=500 i.i.d market sessions, which are treated as contest between AlgoA and AlgoB: if, at the end of a session, the average profit per trader for AlgoA is greater than that for AlgoB, then that counts as a "win" for AlgoA. Hence, the maximum possible score is 19x500=9,500. In each row, bold-font text is used to highlight the larger number of wins in each A-vs-B comparison. As can be seen, switching from BSE to TBSE has no effect in the case of AA-vs-ZIC, but for the other three cases we see a reversal of the dominance relationship. See text for further discussion.

AlgoA	AlgoB	BSE # A Wins	BSE # B Wins	TBSE # A Wins	TBSE # B Wins
AA	ZIC	**7095**	2405	**7370**	2130
AA	ZIP	**7581**	1739	4383	**5117**
GDX	ZIC	**5517**	3988	4300	**5199**
GDX	ZIP	**6538**	2962	2574	**6926**

The rest of this results section shows selected highlights from digging deeper into the outcomes of our experiments. As it happens, a deeper dig unearths some thought-provoking results.

The delta curves from Table 2, i.e. for AA-vs-ZIC, look roughly the same as each other, modulo some noise from the stochastic elements in our simulation, and are not shown here, but Figures 1, 2, and 3 show the delta curves for AA-vs-ZIP, GDX-vs-ZIC, and GDX-vs-ZIP, respectively. These are the three sets of experiments in which the switch from BSE to TBSE inverted the dominance relationship between the trading algorithms, and they deserve some examination and discussion.

Table 2. The data that was aggregated into the top row of Table 1, tabulated to show the individual results from each of the 19 different ratios of the two algorithms used: n=500 at each ratio. The full table is not shown here, because the specific details do not matter. Column sums are displayed at the bottom of the table, and correspond to the values given in the top row of Table 1.

Ratio	BSE AA Wins	BSE ZIC Wins	BSE Δwins	TBSE AA Wins	TBSE ZIC Wins	TBSE Δwins
1:19	**279**	221	58	**297**	203	94
2:18	**355**	145	210	**357**	143	214
3:17			204	**375**	125	250
[illegible]	**373**	127	[illegible]	**384**	116	[illegible]
18:2	**361**	139	222			324
19:1	**324**	176	148	**346**	154	192
Sum	**7095**	2405	4690	**7370**	2130	5240

AA-vs-ZIP (Figure 1). Here there seems to be some coherent structure in the BSE delta curve: although AA consistently outperforms ZIP, the degree by which it beats ZIP seems to attenuate when the R_a is at either extreme of its range, and the maximum outperformance of AA over ZIP seems to be when R_a<10:10, i.e. when AA is in the minority of the population. The relevance of this is seen in the TBSE delta curve in Fig.2: we saw in Table 1 that ZIP wins on aggregate in this set of experiments, but Fig.2 makes clear that when AA is in a small minority in TBSE it can still outperform ZIP, yet as soon as the ratio of AA traders exceeds 50% its dominance disappears and it is outperformed by ZIP.

GDX-vs-ZIC (Figure 2). Here, for both BSE and TBSE, the delta curves have a clear coherent structure to them, which is something that we do not think has been reported before in the literature: we know from Vach (2015), Cliff (2019), and Snashall & Cliff (2019) that ratio matters, but none of those publications reported or explored such strongly coherent relationships between ratio and results, between R_a and $\Delta wins$.

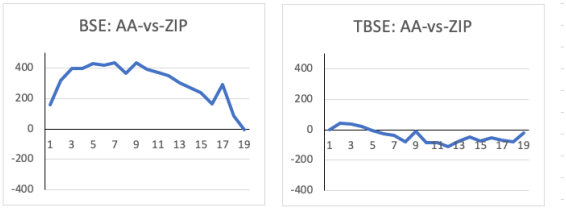


Figure 1. "Delta Curves" for AA vs ZIP in BSE (left) and TBSE (right). This pair of graphs shows the $\Delta wins$ data, as was tabulated for AA vs ZIC in Table 2: the horizontal axis shows R_a and is labelled with the number of AA traders in the ratio; the vertical axis is the value of $\Delta wins$. When $\Delta wins>0$, AA outperforms ZIP, and when $\Delta wins<0$, ZIP beats AA.

Figure 2 shows fairly unambiguously that GDX outperforms ZIC when GDX is in the minority, and ZIC outperforms GDX when ZIC is in the minority. Figure 3 also reveals that the qualitative nature of the swing from GDX dominance in BSE to ZIC dominance in TBSE differs from that from AA to ZIP that was illustrated in Figure 1: whereas the two delta plots in Figure 1 are markedly different, the two curves in Figure 2 are remarkably similar: the TBSE curve could plausibly be described as what happens when the BSE curve is shifted slightly to the right and slightly down. Also notable is that in both BSE and TBSE the maximum outperformance of ZIC by GDX happens at roughly a ratio of 1:3, and the maximum outperformance of GDX by ZICs seems similarly to happen at roughly 3:1. This strikes us as curious in that experience tells us that usually in these kind of experiments maxima would normally be expected to occur either at the endpoints of the scale (i.e. at ratios of 1:19 or 19:1) or near the midpoint (i.e. at 10:10) why this is so is something we aim to investigate in further work.

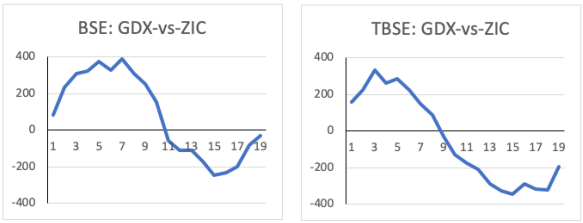


Figure 2. Delta curves for GDX vs ZIC; format is the same as in Figure 1.

GDX-vs-ZIP (Figure 3): again, there is a clear coherence to the delta curves, although the relationship between the BSE and TBSE curves is not as similar as that in Figure 2, and not as different as that in Figure 1. In a point of similarity with the GDX-vs-ZIC curves, again the peak performance of the two strategies come at ratios of roughly 3:1 and 1:3 (and, again, we do not know why this should be so) but for GDX-vs-ZIP the relative performance of the two algorithms does not level out to zero near ratio values of 10:10 and then cross into underperformance for the algorithm that is in the majority; instead in BSE GDX pretty consistently outperforms ZIP and in TBSE the situation is the inverse: now ZIP is the dominant algorithm at almost all ratios, and again the degree of dominance falls steadily as the proportion of ZIPs increases beyond 1:3). Further work is required to understand why these delta curves have this particular qualitative shape.

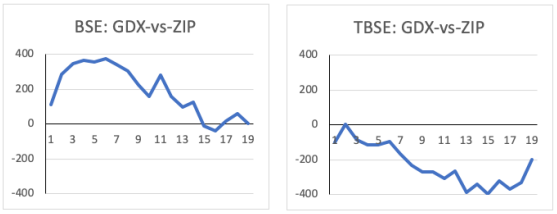


Figure 3. Delta curves for GDX vs ZIP; format is the same as in Figure 1.

5. Discussion and Further Work

One thing that is notable from the delta curves in Figures 1, 2, and 3 is that, despite the points of similarity highlighted above, the effect that shifting from BSE to TBSE has on the delta curves is qualitatively different for each pair of trading algorithms: the data we have studied thus far reveals no conveniently simple patterns that allow us to make a priori predictions or generalizations about when Algorithm A will outperform Algorithm B. Figure 2 is the starkest illustration of this point: recall that GDX was developed at IBM TJ Watson Research Labs, was preceded by two earlier versions (GD, then MGD), and when published was described by IBM as the best-performing trading algorithm in the then-published literature. We do not seek to criticize the IBM team, but our results show that, when the conditions are right (i.e., when the ratio of GDX:ZIC is in the right range), in fact GDX can be consistently out-performed by ZIC. Gode & Sunder's ZIC paper had been published six years before IBM's GDX paper, and while GDX mixes the construction of a probabilistic belief function with techniques from dynamic programming, the ZIC algorithm takes up one line of code.

One avenue of further work comes in attempting to understand what features, if any, of the trading algorithms interact in such a way that they give rise to the delta curves that we have plotted here, and the extent to which those delta curves are affected by changes in other significant factors, such as the market's supply and demand schedules. Also, here we presented results that focus on four trading algorithms that have been used repeatedly in studies of artificial trading systems over the past 20 years, but there are several other algorithms which could be added into this analysis. And, when we're done with analyzing interactions between A/B pairs of trading algorithms, we can move on to A/B/C triples (in which case the delta curves could be plotted as points on a simplex), and then on to various ratios of four or five or six different algorithms, etc. But as the number of algorithms involved in any one comparison increases, so do the number of trials required (the combinatorics are explosive, and the computational cost even of the experiments shown here was measured in days of CPU time), and so do the difficulties of visualizing and analyzing the results.

6. Conclusion

Prior to this paper, anyone reading the trading-agent literature would have been likely to form the opinion that there was widespread agreement that, in general, AA beats GDX, GDX beats ZIP, and ZIP beats ZIC. Only a careful reading of the literature would reveal that many of the relevant results came from single-threaded simulations. Our results presented here show that while the AA>GDX>ZIP>ZIC dominance hierarchy may hold true in simple simulations, as soon as real-time factors matter, i.e. as soon as the various algorithms are operating in parallel, the computational costs of a sophisticated algorithm such as GDX count against it, and it can be outrun by simpler but faster algorithms. So, the primary contribution of this paper is our demonstration here that the old single-threaded dominance hierarchy is not maintained in multi-threaded TBSE. As Table 1 shows, in A/B comparisons on TBSE we have AA beating ZIC, ZIP beating AA, ZIC beating GDX, and ZIP beating GDX; results that can be summarized as ZIP>AA>ZIC>GDX.

Our work calls into question not just the truth of specific claims of dominance, but also whether it is ever worth trying to make such claims at all, because any trading-algorithm's performance, and hence dominance, is clearly so heavily affected by factors exogenous to that trading algorithm, chief of which is what other algorithms it is competing against, and in what proportion or ratio those different algorithms are present in the market: it's a manifestly game-theoretic situation, but game theory offers no help here. We question whether, in markets that are sufficiently realistic to be relevant to the real world, there can ever really be a single specific trading algorithm that is "dominant", that actually beats all the rest. Hence, we conclude with this: if you think your trading algorithm really is the dominant one, you have probably tested it in simulations that are too simple.

References

BSE, 2012. *Bristol Stock Exchange*. GitHub repository at https://github.com/davecliff/BristolStockExchange

Cliff, D., 1997. *Minimal-Intelligence Agents for Bargaining Behaviours in Market-Based Environments*. HP Labs Tech. Rep. HPL-97-91.

Cliff, D., 2019. Exhaustive Testing of Trader-Agents in Realistically Dynamic CDA Markets. In *Proceedings ICAART-2019*.

Cliff, D., 2019. Simulation-Based Evaluation of Automated Trading Strategies: A Manifesto for Modern Methods. *Proc. EMSS-2019.*

Das, R., Hanson, J., Kephart, J., Tesauro, G., 2001. Agent-Human Interactions in the CDA. *Proc. IJCAI-2001*, pp.1169-1176.

De Luca, M., Cliff, D., 2011a. Agent-Human Interactions in the CDA, Redux. *Proceedings ICAART-2011*.

De Luca, M., Cliff, D., 2011b. Human-Agent Auction Interactions: Adaptive-Aggressive Agents Dominate. *Proceedings IJCAI-2011*.

De Luca, M., 2015. *Adaptive Algorithmic Trading Systems.* PhD Thesis, University of Bristol, UK.

Gjerstad, S., Dickhaut, J., 1997. Price Formation in Continuous Double Auctions. *Games & Economic Behavior*. 22(1):1-29.

Gode, D., Sunder, S., 1993. Allocative Efficiency of Markets with Zero-Intelligence Traders. *JPE*, 101(1):119-137.

Rollins, M. 2020. *Threaded BSE: Experiments with Parallel Asynchronous Algorithmic Trading Systems.* Master's Thesis, University of Bristol.

Rust, J., Miller, J., Palmer, R., 1992. Behavior of Trading Automata in a CDA Market. In Friedman, D., Rust, J. (eds) *The Double Auction Market: Theories and Evidence.* Addison-Wesley, pp.155-198.

Smith, V., 1962. An Experimental Study of Competitive Market Behavior. *Journal of Political Economy* 70(2):111-137.

Snashall, D., 2019. *An Exhaustive Comparison of Algorithmic Trading Strategies: AA Does Not Dominate*. Master's Thesis, Uni. of Bristol.

Snashall, D., Cliff, D., 2019. Adaptive-Aggressive Traders Don't Dominate. In van den Herik, J., Rocha, A., Steels, L., (eds) *Agents & Artificial Intelligence: Papers from ICAART 2019*. Springer.

Sommerville, I., Cliff, D., *et al.* 2012. Large Scale IT Systems. *CACM*, 55(7).

Tesauro, G., Das, R., 2001. High-performance Bidding Agents for the CDA. *Proc. 3rd ACM Conf. on E-Commerce*, pp.206-209.

Tesauro, G., Bredin, J., 2002. Sequential Strategic Bidding in Auctions using Dynamic Programming. *Proceedings AAMAS2002*.

Vach, D., 2015. *Comparison of Double Auction Bidding Strategies for Trading Agents.* MSc Thesis, Charles University in Prague.

Vytelingum, P., 2006. *The Structure and Behaviour of the Continuous Double Auction.* PhD Thesis, University of Southampton.

Vytelingum, P., Cliff, D. Jennings, N., 2008. Strategic Bidding in Continuous Double Auctions. *Artificial Intelligence,* 172(14):1700-1729.