Centre of Intelligent Systems and their Applications

A Multi Agent System for Real Time Adaptive Smart Order Routing in Non-Displayed Financial Venues (Dark Pools)

Abdul (Abyd) Adhami

MSc (by Research)
School of Informatics
University of Edinburgh
2010

Abstract

The increase in electronic financial trading venues, fuelled by recent regulatory directives, opened up the space for more choice in the market. No longer is the main primary exchanges the only option for executing orders; other venues, independently run or operated by major financial firms, offer an alternative choice. Non-displayed venues, commonly referred to as dark pools, are fast becoming a significant part of this constantly evolving space. They allow large institutions to trade sizable orders against each other "in the dark" without being shown to the market until they're complete; therefore, preserving pre-trade anonymity and in theory, limiting price impact. With the increase in these venues, and subsequent market fragmentation, order routing technologies rose to prominence. They aim to automate both the allocation and routing of orders efficiently across the many available destinations.

The objective of this project was to study and explore order routing within non-displayed venues. We propose a real-time routing algorithm, in the form of a probability formula, which could channel order flow across different dark pools. The algorithm achieves this by ranking each venue, taking into account multiple factors when reaching its decisions. We develop and implement a smart order routing system which utilises our routing algorithm, and evaluate it through a multi agent simulation. The system was demonstrated and tested, across a set of simulated experiments, and was found to be largely effective in many scenarios. We show that the combined use of both historical and immediate trading indicators allows the router to respond -and adapt- to different changes in the market. The multi agent simulation designed and developed for the project also incorporates many of the industry standard and real life characteristics found in trading systems today.

Dedicated to my mother and father for whom I shall always be indebted to.

Acknowledgments

I would like to thank my supervisor, Prof. Dave Robertson, for his valuable comments and direction. I would also like to thank my wife, Iman, for her unwavering support throughout the many long hours I dedicated for this project. I also thank my previous and current employers, National Australia Bank and JP Morgan, for their support in providing me with study leave to focus on my research.

Declaration

I declare that this thesis was composed by me, that the work contained herein is my own except where explicitly stated otherwise in the text, and that the thesis work has not been submitted for any other degree or professional qualification except as specified.

Abdul (Abyd) Adhami

Contents

Chapter 1

Introduction

1.1 Overview

Advances in trading technology and innovation constantly alter the financial markets industry, gradually increasing its complexity and sophistication. With more trading venues being created, order routing technologies become increasing important. Given the rise in the factors and complexities involved in routing, a newer breed of routers began to surface, aiming to answer to the ever more demanding needs of traders and the market. In the past number of years, the so called smart order router, as a term, began to be used; this refers to those routers that offer a greater degree of intelligence and sophistication in their routing behaviour. The raise in non-displayed trading venues created a further challenge for the routers and in turn fuelled the need for more work in this area; this resulted in a greater amount of resources being allocated for the development of routing systems in this space.

For what is considered a good and fertile ground for further search [14, 18, 26], our motivation is hence set to explore this interesting area and to ultimately develop and evaluate a smart order routing system that is able to allocate orders efficiently across non-displayed venues. We start by formalising our aims and objectives, then presenting an outline of the project, highlighting the main activities and overall structure and layout of the work.

1.2 Aims and Objectives

The core objectives and scope of the project can be summarised as follows:

A. Propose and develop a smart order routing system, specifically optimised for non-displayed venues, able to utilise multiple elements in making and reaching its routing decisions for stock orders.

B. Evaluate the routing system within a multi agent simulation, taking into account all the necessary factors (where possible) to achieve a realistic test environment. This includes the development of trading venue and trading

party entities as well as other elements such as market pricing feeds which are necessary to enhance the realism of the simulation and experiments.

C. Design and conduct several experiments as part of the evaluation process that highlight and demonstrate the effectiveness of the different elements developed in the routing system.

D. As a secondary objective, develop the components of the simulation system to be as interoperable as possible, utilising industry standard technologies, especially those used by the financial industry today. This helps create a flexible and expandable system, sharing more common attributes with reality.

1.3 Project Outline

To set the project in context, Chapter 2 provides an overview of trading venues and further background on the development and rise in non-displayed liquidity within an increasingly fragmented market. Research in this area is discussed and multi agent simulations for evaluation purposes are introduced.

Chapter 3 focuses on the core theoretical design of the smart order routing system, presenting the different elements and routing indicators developed. The proposed routing formula is discussed and placed in context to the overall trading system, covering other areas such as order splitting and execution management.

Chapter 4 and chapter 5 detail the design and implementation of the multi agent simulation, highlighting the main components of the system and discussing its behaviours, attributes and overall functional specifications. The implementation chapter covers the technical details of how the system was developed and deployed, along with the key areas considered.

Chapter 6 focuses on the evaluation of the system, conducting different experiments and discussing their results. A range of tests are made, demonstrating and evaluating the routing behaviours of the system developed.

Finally, chapter 7 concludes the work, highlighting the overall project findings and observations made, as well as outlining some further development areas for the future.

Chapter 2

Background and Related Work

In this chapter we give an overview of electronic trading venues and their increase resulting in subsequent liquidity fragmentation within the financial markets space. We explore the emergence of non-displayed venues and their use in trading. This leads us to one of the primary goals of our research, intelligently routing and channelling trades across non-displayed venues. We look at research and related works in this area, helping set the scene for the project.

2.1 Background

2.1.1 Electronic Trading

The financial industry's rapid shift towards electronic trading both in exchange and so called Alternative Trading Systems (ATS) in the late 1990s, where orders are automatically matched at specified prices, fuelled the evolution and transparency of trading. In the US alone for example, there were 11 electronic trading systems in 1997, 40 in 1999 and more than 80 by 2000 [12]. A similar shift can also be found in Europe, where the bulk of trading now occurs electronically.

As markets increased in size and sophistication, it gradually became apparent that automating decisions on how, when and where to execute orders enhanced the profitability and efficiency of these trades to complete successfully. These factors lead many researchers to explore automated trading and execution through simulations, utilising dynamic learning behaviours to trade effectively in various market conditions.

Electronic Order Book

The electronic limit order book is at the heart of any exchange or trading venue. Market participants submit orders and trade against each other on these systems [15]. Continuous bid and ask (or offer) prices are quoted throughout the operating hours of the exchange for each tradable stock. On many exchanges, such as the London Stock Exchange, participants are also given access to view a subset of the latest orders submitted into the book showing bid and ask prices alongside

volume and time. A typical snapshot of an order book for Vodaphone on the London Stock Exchange for a given intraday is shown in fig 2a below.

VODAPHONE GRP. ORD
BID: **160** ASK: **160.1** Mid: **160.05** Total Volume: 24,120,302

BUY Orders (BID)			SELL Orders (ASK)		
Time	***Volume***	***Price***	***Price***	***Volume***	***Time***
10.15	6,700	160	160.1	1,278	10.21
10.21	13,200	160	160.2	3,200	10.16
10.20	5,348	159.9	160.2	2,500	10.18
10.18	2,210	159.8	160.3	7,434	10.20
... etc					

Fig 2a: Typical order book showing buy and sell orders

The order book system shows at a glance all the open orders for a stock, which allows the market to gauge supply and demand. This in turn aids in price discovery and publication. Orders generally execute based on price and time. If a buy order is submitted at a price equal to or more than the best sell order within the opposite side of the book, then a trade will execute successfully.

Market participants, whether human traders or algorithmic systems, are hence able to view and analyse this data when making judgements on how and when to trade on the exchange.

2.1.2 Dark Pools

Unlike the primary public markets operated by stock exchanges, non-displayed markets, known as dark pools, are alternative venues that run in parallel and do not quote prices or order volumes. They match buyers and sellers anonymously, often using pricing feeds from the primary public markets to complete trades at mid point or better. The order book in these venues are hidden from view, hence no participant will have visibility of live or active orders present within them.

The initial aim of dark pools was to facilitate trading of large orders that have become difficult to complete on traditional displayed exchanges without incurring significant market impact. Aitken [3], Harris [31] amongst others suggest that hidden liquidity helps limit order exposure risk; this exposure can lead to higher execution costs by revealing the trader's motives to the rest of the market. Today dark pools cater for a wide range of orders while also minimising information leakage often with lower direct trading costs.

Many of the major financial firms such as investment banks have been operating internal pools for some time. These systems match client orders in-house, saving firms the fees and commissions associated with filling orders on public exchanges and other venues. Several of these firms have now opened their systems for clients to access directly as dark pools. Credit Suisse launched its CrossFinder system [24] handling up to 60 million shares a day. Goldman Sachs rolled out Sigma X [22] while UBS promoted its Price Improvement Network [15]. A number of independent competitors including Liquidnet and POSIT [24] also run and market major, successful dark liquidity pools.

2.1.3 Market Fragmentation

The rapid increase of these venues was accelerated by reforms introduced by the Securities and Exchange Commission (SEC) in the US and the Markets in Financial Instruments Directive (MiFID) in Europe [17]. They opened up the market, increasing competition and paving way for new venues to operate. They also required firms to actively seek the best execution for their clients' orders. All this led however, to further liquidity fragmentation in the market. Liquidity is fragmented when multiple trading venues or destinations are necessary to complete a given order size.

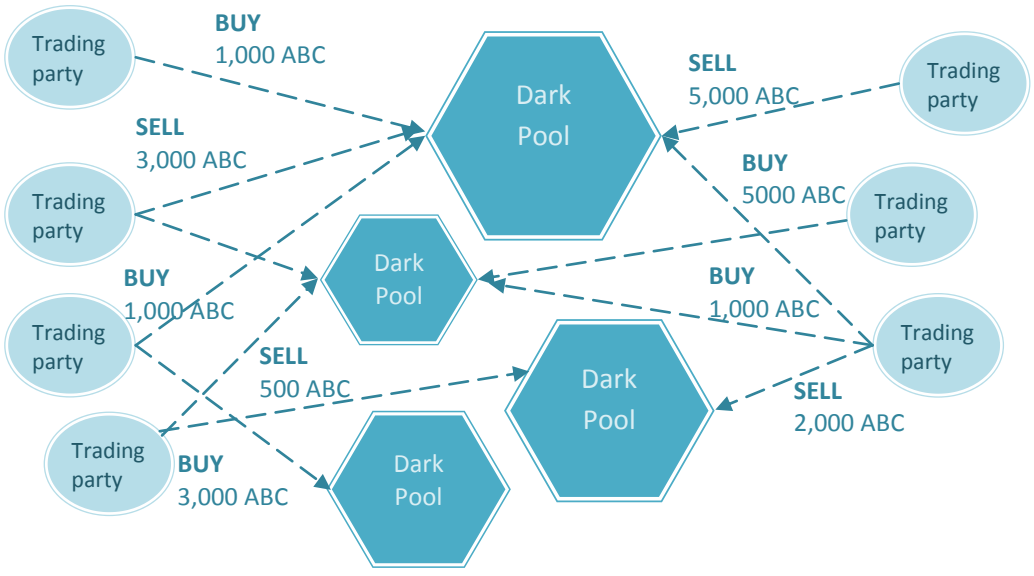


Fig 2b: Shows an illustration of trading parties interacting across multiple dark pool venues. Each party is seeking liquidity for stock ABC to execute and complete its orders anonymously. The type, price or volume of liquidity residing at any moment of time in each pool is not known to the parties.

With over 40 active dark pools in the US in 2009 [16] and an increasing number in Europe all running alongside the existing venues and exchanges, the need for effective execution and routing of orders across these becomes very important. In fig 2b above and in this project, we specifically look at the interaction with dark pools, which forms the focus of our research. Other venues of course, including the primary exchanges also come into play, which we can discuss later.

2.1.4 Smart Order Routing

When liquidity was confined to a single primary exchange, routing orders to the venue was a simple matter. But with market fragmentation and increase in alternative trading venues, this routing process becomes more interesting. When you have multiple venues, which do you route to first? How do you decide this and what logic do you build? Smart Order Routing (SOR) technologies evolved for this very reason. They help traders route their orders to the most appropriate venue or venues. Early SOR systems were very basic, simply routing orders based on stock type to a handful of venues. These evolved further to factor basic trading costs and preferential venues based on a predefined logic.

Challenges in SOR with Dark Pools

With non-displayed venues, order routing is far more trickier. The routing logic cannot directly access the depth of the order book for each of these venues. Hence it cannot easily determine beforehand if liquidity is available on them or not, unlike public or displayed markets. Early SOR typically "pinged" each dark pool sequentially sending orders that are designed to be immediately cancelled if unsuccessful, allowing the router to try the next venue inline.

SOR may also split the order into smaller child orders before sending them. This however raises the question; how many splits does it perform and to which venues does it send to first? How does it determine the likelihood of liquidity in one venue over the other? If it routed an order to too many wrong venues, resulting in little or no successful trades, what impact does this have? There would of course be costs incurred both in terms of delay and lost opportunity in contrast to a router that hits the "right" venues early on. Lost opportunity may occur when a venue with good liquidity is missed early on and hence taken away by other competitors. The market may also move on, therefore making liquidity that was available early on no longer tradable at the price required.

Finding liquidity in dark pools is not the only challenge for SOR. The system must also route to the venue offering the best price and lowest execution fees. So if two venues contain identical lots of hidden liquidity, a decision must be made on which offers the best overall value. Multiple factors would hence play into the routing logic.

2.2 The Research Landscape

Although there have been countless works and research published around order execution in financial markets and trading systems, most tend to deal with open exchanges or ones where market pricing and volume indicators are accessible. Less work and research, in comparison, can be found for routing and execution systems dealing with pure non-displayed, dark pool venues. Although this is changing as more is being done in this area [62, 63], it is still considered to be a fertile ground for continued research.

2.2.1 Hidden liquidity

Many of the open exchanges, well before the proliferation of dark pools, developed functionality in their order books enabling parts of a trade to be electronically hidden. This was sometimes referred to as an "iceberg" order, where only a proportion of the order volume is revealed, leaving the rest hidden from view [15]. Researchers began to look into this in recent literature and a number of papers were published. De Winne and D'Hondt [2] analyse a set of full historical order book data from the Euronext exchange. They quantify the existence of hidden liquidity and present various ratios detailing this accordingly. They estimate that around 20% of the liquidity on Euronext (in 2002) is hidden. Pardo and Pascual [6] examine data from the Madrid Stock Exchange, estimating that nearly one fifth of all trades completed (in 2000) would have involved a level of execution against hidden liquidity.

With dark pools, where all venue activity is hidden, a study by Tabb Group [18], a markets research firm, estimates that these venues account for 11.5% of the average daily volume of equity trading in the US for 2009. In Europe this figure is lower, though increasing, where dark pools are estimated to rise to 8% by 2011.

2.2.2 Adaptive 2nd Generation SOR in Dark Pools

Research published by industry professionals such as Sofianos [7] et al look at dark pools and how algorithmic trading may be used to access and interact with hidden liquidity. They outline three main components of an algorithm that can be used to slice an order into smaller pieces before pricing and sending them to be executed against different venues. The routing logic would access each dark pool sending small order lots hoping for a successful execution to occur. Orders that were not successful are then pulled and rerouted to a different venue. The development of these algorithms to interact with the dark pools, seeking out hidden liquidity, gradually emerged. A number of products were developed and released over the past years, including Goldman's Sonar [22], Selero's SORBET and CAPIS's Nocturnal [26] algorithms. These systems attempt to trade various order lots across multiple venues to sample the greatest range of offers and hence factor this, amongst others, to efficiently route the bulk of the order flow to the optimum venue.

Research by Almgren and Harts [5] look at the SOR problem and present a simple algorithm which aims to maintain a dynamic estimate of hidden liquidity across a particular venue. This could then be used in the routing process. By observing trades that take place, the algorithm's estimate of hidden liquidity is increased each time the reported executed volume exceeds that of the displayed. Of course in pure dark pool venues, no trade volume is ever displayed; hence the estimate would rely on the number of successful orders executed through time. This estimate would decrease as time passes, especially if fewer executions are reported on the venue. Other research by Ganchev and Kearns [63] further study the dark pool problem and devise an allocation algorithm that aims to maximise the number of submitted shares for each routing interval, according to the volume executed by each dark pool. A re-estimation step is also factored to redistribute the allocation based on subsequent feedback.

Rawal [19], Pujol [20] et al in recent publications look at the existing and next generation SOR systems. These papers cite the increased use of real-time analysis in order to create an adaptive routing system that can seek and execute against dynamic liquidity. They also mention the importance of incorporating historical execution as well as venue data in SOR to predict where liquidity is going to be.

The competitive advantage that an effective SOR can bring to banks and investment firms has lead the industry to seek and move towards the development of more advanced and adaptive systems. This can be seen today with technology and product announcements being made regularly by financial institutions and third party companies. It is partly this competitive advantage that the inner workings of these systems are kept closely guarded and proprietary to these firms. Which is why, although gradually changing, a relatively small number of detailed research can be found in this area so far.

This project will aim to explore and demonstrate some of the evolving SOR concepts, through a multi agent simulation. Most of the research and simulated tests around order routing and dark pools rely heavily on "back testing" against historical data from real venues (e.g. [63, 64]). Whilst this does give a good indication of how the models are performing, it does not take into account its own effects on the data. This is where multi agent simulations may help.

2.2.3 Multi Agent Systems in Market Simulations

Agent based simulators, unlike traditional simulators, do not hold a single decision maker for the entire system. It rather represents different independent entities, or agents, each having its own decision making process. These agents would typically interact with each other and adapt their strategies based on the success or failure of previous efforts. This may help in the evaluation of the models being built in the system, especially when viewed from an individual, agent's perspective. Many projects and research groups have designed and developed market simulators that aim to provide a better understanding of financial markets and different trading models.

Kearns and Ortiz [8] outline the Penn-Lehman simulator project. This project aims to simulate a typical exchange by combining market data with automated client orders. It is structured in a way that multiple automated traders or clients can be setup to compete against each other in the simulator, all trading against the same virtual exchange. Many other multi-agent based market simulators have also been presented and cited in recent literature. Kendall and Su [9], Preus and Golke [10], Market Based Control TAC/CAT Competition [11] amongst others, demonstrate dynamic learning behaviours between multiple participating agents –again all trading in a simulated environment.

As noted earlier, this project will demonstrate smart order routing concepts in dark pools via the development of a simulated multi agent market environment. While there will be similarities with some of the previously mentioned market simulators, there are a number of differences in our case. Firstly, many of the current simulators cited deal primarily with a single centralised venue or exchange, where all agents are set up to trade in. This is not the case here where the simulation must model both multiple unique venues (dark pools) and trading agents all in the same environment.

Secondly, although we model and cover similar trading dynamics and rules, where agents essentially "buy and sell" against each other, one of the main focuses of our research is liquidity seeking and order routing. This routing is also within a specific set of markets, namely non-displayed dark pool venues. Another objective is to demonstrate a simulated system utilising current industry standard technologies and protocols as used by the banking and financial world today. The combination of this environment along with the use of standard protocols will help present the possibility of "plugging in" real near production-ready components into the simulation for testing.

Chapter 3

The Dark Routing Algorithm

This chapter details the logic and design utilised in our Smart Order Router (SOR), including the technical indicators and parameters used. We present the routing algorithm in the form similar to a probability formula and show how this can be modelled in the system. We then outline the design of an execution management system that will be used, looking at how it will process, manage and route orders.

3.1 Routing and Execution Probability

When trading in dark pools, where liquidity is hidden, the routing ability of a system becomes vital. When an order is submitted for execution, the routing logic should channel this to the best market venue possible; one that will offer the best price depending on the order requirements. However there is little point in constantly routing an order, or the bulk of an order, to a venue that never executes it successfully. Therefore, knowing the probability of execution of this against different venues plays a significant part in enhancing the overall quality and efficiency of the system.

Probability

In mathematics, a probability of an event is represented by a real number in the range from 0 to 1. The closer that number is to 0 the less likely that such an event will occur. Conversely, the closer it is to 1 the more favourable and likely it will occur. So for example when predicting the weather for tomorrow in a particular place, and assuming that it can either be sunny or rainy, then without knowing much else, one might estimate that there is a 50% or 0.5 chance of getting rain. This probability figure might change if the time of year is known. It might also change if it is currently raining there. This figure can change even further if current wind, pressure and cloud cover is known.

Similar principles can also be applied in the probability of execution, so that to maximise the chance of orders being routed to the best venues possible.

3.2 Routing Indicators

In order to develop the probability formula for our routing algorithm, the input components must be identified. These comprise of indicators or values that might be captured and derived during the running of the SOR system or even preset at the start. They provide the router with the ability to calculate and draw upon different conclusions taking into account the original order requirements and the various market factors.

With dark pools, the system will be dealing with non displayed liquidity, therefore no volume, price or order book data will be available to the router from any of the venues. It will hence have to rely on other factors which can aid in the probability formula. Also, while determining liquidity is a priority for the router, obtaining a good price is another factor which must be taken into consideration. This can include the fees that each venue charges and other cost based areas that can impact the overall efficiency of the transaction. The following summarises the main dimensions that must be taken into consideration when reaching to the best execution probability and making the routing decision.

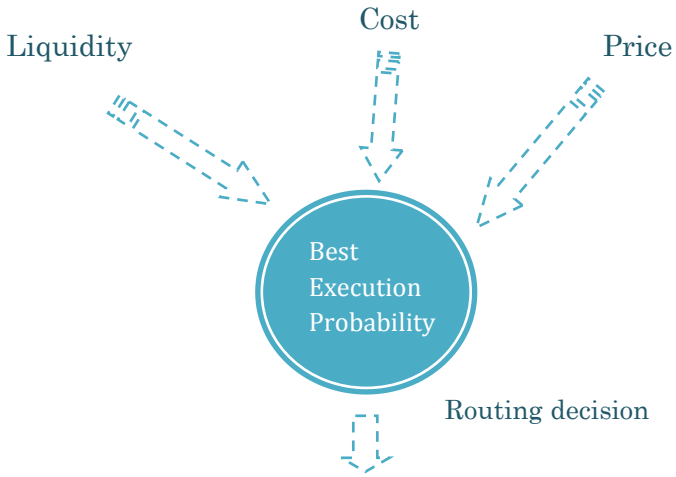


Fig 3a: Key attributes for Smart Order Router and best execution

To be adaptive, the SOR system must monitor and gauge where liquidity is most likely to be across the different venues. It would need to use both historical and real time market data. It must also determine where the lowest execution cost is depending on order requirements, as well as who is providing the best price. All these factors must be made for each stock to be routed, including the side of the order, buy or sell.

This section will identify some of the indicators that can be obtained and how they can be utilised in the probability formula.

3.2.1 Historical Trading Volume

Each time an order successfully executes against a venue, the feedback can be stored in the SOR system. This historical data can be used to build a picture of the dark pools that are more frequently executing particular orders rather than others. To explain how this might be achieved, a detailed example follows below:

A sell (bid) order of 10,000 shares of stock ABC is required to be executed in dark pools. There are four dark pool venues, DPa, DPb, DPc and DPd. Equal amounts of 2,500 lots of the order is submitted to each of these venues at 9:00am. If the post-trade feedback, i.e. the results of this order submission, shows that DPa and DPc execute half of their 2,500 order, and nothing is completed from either DPb or DPd, then the following could be captured in the system:

Time	DPa	DPb	DPc	DPd
9:00	1,250	0	1,250	0

Table 3.1: Historical trading volume for Stock ABC(bid), showing shares executed on each venue

At a later time of 2:00pm, another order of the same stock is submitted by the router, similar as before. This time however DPc and DPd execute their orders successfully (2,500 each) while the others return nothing; the feedback table will now look as follows:

Time	DPa	DPb	DPc	DPd
9:00	1,250	0	1,250	0
14:00	0	0	2,500	2,500

Table 3.2: Accumulated historical trading volume for Stock ABC(bid)

From the historical snapshot view collected, DPb seems the least likely to execute our particular order from among the others. While DPc looks like the most favourable of them to have executable liquidity.

The total accumulative volume experienced on each venue through time, which we will call the Historical Trading Volume (HTV), can be added up and calculated as follows:

$$HTV\ (per\ venue) = \sum_{j=1}^{N} (Tvol_j)$$

where N is the total number of trade entries recorded through time; the routing system experiences successful trades $Tvol_1$,..., $Tvol_j$ at different intervals during this time.

A preference or probability value can be calculated to better represent the proportion of volume executed across each of the venues. This can be derived by dividing the HTV from each venue over the total HTV executed across all.

$$HTV\ Probability\ (venue) = \frac{\sum_{j=1}^{N}(Tvol_j)}{\sum_{i=1}^{k}\left(\sum_{j=1}^{N}(Tvol_j)_i\right)}$$

Going back to the example, the total HTV for the stock executed across all four venues is 7,500. This is calculated by adding up all the volume in table 3.2 above. Hence, the probability values for each venue would be:

Venue	Probability
DPa	0.166 (1,250/7,500)
DPb	0
DPc	0.5
DPd	0.333

Table 3.3: HTV venue probability values

So from this simple representation, the routing system can interpret and make some use of the historical data it collects from previous experiences it has on these venues. An observation to make from table 3.3 above is that venue DPb has never executed any orders so far, and hence has a probability of zero. This indicates to the router that in the current conditions, based on track record, DPb may have a very little chance of executing any subsequent orders.

3.2.1.1 Router and Market Historical Trading Volume

The trading activity described earlier is based on the experience of the SOR system itself. It hence only takes into account the trades that have occurred through its own routing activity. This will be referred to as the *router* Historical Trading Volume (rHTV). Many other trades of course are executed and completed

on the dark pool venues by other participants. These will also need to be monitored and captured each time they are known.

While dark pools are not obliged by the regulatory bodies to provide pre-trade transparency, they must report post-trade activity. In other words, once a trade is executed, they must report this to the market [17]. This post-trade activity is sent to a regulated trade reporting service, which in turn publishes and notifies all its subscribers in the market. The SOR system will be one of those subscribers, it will continuously observe any trade volume completed and record it for use in future routings. This activity will be referred to as the *market* Historical Trading Volume (mHTV).

Both *router* HTV and *market* HTV will be captured and calculated in the SOR system for each dark pool venue.

$$combined\ HTV\ Probability\ (venue) = \frac{(rHTV_{venue}) \,.\, f_1}{\sum_{i=1}^{k}(rHTV_i)} + \frac{(mHTV_{venue}) \,.\, f_2}{\sum_{i=1}^{k}(mHTV_i)}$$

The f here is the adjustment factor, which will be explained in detail in the probability formula section later. But for the combined HTV formula above, it is used to distribute and represent rHTV and mHTV together. A possible value for each f can be 0.5 which will result in an evenly balanced weighting between the two.

However, a larger weighting should be placed on the rHTV as details of this activity is much more crisp and clear to SOR than the mHTV, which as we said, is be based on market trade reports. The main reason being is that while all dark pool venues publish the size of their trades, the "tape" shown by the trade reporting service may not always necessarily give enough details to determine the actual venue that executed the trade. At times trades may be published together as a consolidated report and may even come from multiple venues. All this means that it will not always be possible to determine the exact sources of execution for each trade reported. It is hence reasonable to place more importance on the activity that the SOR system experiences from its own routings rather than the market as a whole. This is mostly true for routing in dark pools. However when incorporating the SOR system to deal with both displayed and non-displayed liquidity then the balance and importance must be revisited.

3.2.1.2 HTV and Time-Weighted Decay Factor

While the probability values calculated for HTV (whether rHTV or mHTV) are inherently volume weighted, i.e. take into consideration the proportion of shares executed on each venue, they do not necessary consider the time factor. In a fast moving trading environment many conditions may change. Liquidity its self may move very quickly from one venue to the other. It is therefore unreasonable to treat all the historical data with the same weight. Almgren and Harts [5] suggest the use of an exponential decay factor, reducing the weighting of data as time passes by. A similar approach will be taken here as well, to ensure that the system can adapt to changes in market conditions.

Phase (p)	Time passed	Decay factor
1	Less than 4 hrs	1 (i.e. no decay)
2	Half day	0.7
3	1 day (T-1)	0.5
4	2 days (T-1)	0.2
5	Over 2 days	0 (discarded)

Table 3.4: Showing decay factor to be applied to each traded volume based on how old it is.

The router will limit its use of historical trading data to a set period of time. In the above example, a couple of days are used, however this may defer based on the market capitalisation of various stocks. The question of how far back remains open, as there will be no clear right or wrong answer; on one hand, if weeks or months of data is taken into account, then the router risks becoming less adaptable to market trends and fluctuations; one or two very large trades executed a while back on a venue will continue to significantly influence the probability for some time to come. But of course, on the other hand, if only a few hours of historical data is taken into account, then the bigger market picture might be lost, which could hold some value. In reality, this parameter would need to be monitored regularly and adjusted depending on market conditions.

The decay factor d_p can be represented in the HTV formula, where p is the phase (1...5) that determines the factor of the decay based on table 3.4 above.

$$HTV\ (per\ venue) = \sum_{j=1}^{N} \left(Tvol_j \ .\ d_p \right)$$

To demonstrate this in action, going back to the HTV example, let's say further routing is made and subsequent execution feedback is recorded from each of the venues. This can be shown in the following table 3.5[1]. If the time decay factor is now to be taken into consideration and applied on the historical traded volume, then the new adjusted totals would look as detailed in table 3.6.

Time	DPa	DPb	DPc	DPd
Mon 9:00	1,250	0	1,250	0
Mon 14:00	0	0	2,500	2,500
Tue 11:00	3,000	50	0	100

Table 3.5 Before applying decay factor adjustment

Time - *[Decay factor]*	DPa	DPb	DPc	DPd
Mon 9:00 [*0.5*]	625 (1250 *0.5)	0	625	0
Mon 14:00 [*0.5*]	0	0	1,250	1,250
Tue 11:00 [*1*]	3,000	50	0	100

Table 3.6 After decay factor adjustment (assuming time is 11am Tue)

The adjustments applied here ensure that more recent activity has a larger influence on the probability calculation than less recent ones. Again, as discussed earlier, how far back to go in historical data remains a refinement parameter, which may require regular tuning according to the market.

3.2.2 Immediate Trading Activity

It is safe to assume that if a routed order executes fully (gets filled) against a particular dark pool venue, then there is a good chance that further similar liquidity might exist on that venue. This activity cannot be regarded as simply historical, as the urgency of the information would require immediate action if it is to become useful. When the router sends an order to be executed against a dark pool, then knowing if this order has completed fully –once the feedback from the venue is received– provides a powerful piece of information that can be used by the router for immediate subsequent trades.

Since liquidity may change hands quickly, and hence what may remain in a venue a minute (or second) ago may no longer be available, it would be important for the router to act fast. The routing system may act by sending further orders – which it has not filled yet– of similar nature to that venue in the hope of more

[1] In the interest of a simple demonstration, only a few data entries are shown here. In reality however, a much larger number of historical trading data would be captured.

successful fills of the same liquidity. To represent this concept as a usable indicator, which we will call the Immediate Trading Activity (ITA) flag, a value will be set to either 0 or 1 for each stock traded on each venue.

Execution feedback	ITA flag
Order filled completely	1 (expires after 5 seconds back to 0)
Order partially completed	0
Order not completed	0

Table 3.7: Shows ITA flag values based on how well an order executes

The ITA flag can be thought of similar to a probability value. In this context we assume that a probability value of 1 means there is a high chance[2] that further liquidity would be available at that venue; this is true if a previously routed order just fully executes. If an order never or only partially completes then the ITA flag is set to zero; in the case of partial completes it would imply there was not enough liquidity at the venue to start with.

When the ITA flag is set to 1 due to a recent trade, then this is expired shortly after; since as time passes, this information no longer becomes "immediate". Five seconds is used to start with, which might be re-tuned, possibly to a shorter length depending on market conditions. The main goal for ITA is to indicate to the routing system that if a full trade occurs then it could signal further immediate opportunities; these however must be acted upon quickly. Hence choosing a longer expiry time for ITA would defeat this purpose.

3.2.3 Venue Trading Cost

Each dark pool or trading venue will have a business model and hence charge different execution fees from buyers and sellers trading on their environment. These fees, normally quoted in bases points (1bp = 0.01%) is the commission they charge per share executed successfully. Many of the dark pools in operation today typically charge anything between 0.2bps to 1bp. Nasdaq Neuro charges 0.2bps, while Turquoise's dark pool charges 0.3bps per share [25].

[2] Although a probability of 1 in mathematics implies that there is a 100% chance of something occurring, here we simply assume that it is very high, while 0 is very low.

As a typical example, if the venue charges 0.5 basis points for 10,000 shares of a stock trading at £1.00 each, then the total commission would be £0.50. This may seem little, but the economies of scale play its part here where a large number of trades pass through the venues to make them commercially viable. Some venues may charge membership fees or provide rebates and other promotions depending on how participation of liquidity in the venue is made. In order not to overly complicate our model, we will only focus on the standard commission rates per share. Following table 3.8 shows what each venue might charge.

Venue	Trading Cost Fee
DPa	0.3 bps *per share*
DPb	0.7 bps
DPc	0.5 bps
DPd	0.3 bps

Table 3.8: Shows possible trading cost fees for each venue

Of course, the venue trading cost here will not necessarily aid in liquidity seeking; it is more likely to be used for ensuring that execution fees are factored into the overall decision making process. It is worth noting however, that it is sometimes inefficient to place great emphasis on this indicator, since any potential savings from lower commissions from venues could easily be swamped by other factors, such as opportunity costs, market movements etc. This is mentioned in some literature [e.g. 67] discussing elements to factor in execution strategies.

3.2.4 Price Improvement Indicator

When trades execute on different destinations, the execution price may vary from one venue to the next[3]. Dark pools, like many other venues, generally execute orders at a price within the best bid and ask (offer) as published by the open market. In the US, this can refer to the regulated National Best Bid Offer (NBBO) price [40]. The spread of the bid-ask is essentially the difference between the highest price that a buyer is willing to pay for an asset or stock and the lowest price for which a seller is willing to sell it.

So for the stock shown in the graph below (fig 3b), for any price to fall within its spread at (for example) 10:33am, it would need to lie in-between 145.35p and 145.55p.

[3] This is discussed and covered in detail under experiments III of the evaluation chapter.

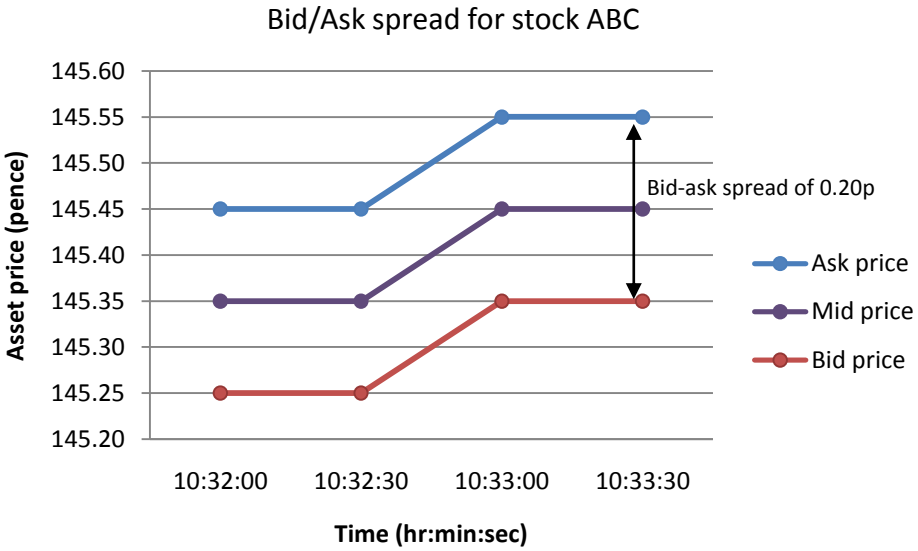


Fig 3b: Shows the bid/ask spread for a sample stock ABC against time

Since dark pools are non-displayed, there is no way of knowing exactly what price within this bid/ask spread can be achieved. This would depend on what liquidity is in the venue and what limit price may have been set by the participants. It will also depend on the inner details of how the dark pool venue crosses and executes the trades[4]. All these issues can ultimately impact the execution price received from each venue. Therefore if the SOR system can collect data on how well its orders execute against each venue in relation to the open market price at that time, then this may aid in future routings.

A possible indicator to represent this, which we will call the Price Improvement Indicator (PII), would simply re-adjust a scoring depending on how each order executes within the bid-ask spread. This PII scoring, kept for all traded stocks against each venue, can be updated based on the following logic:

Executed price at venue	PII score
In the top qtr of the bid-ask spread for a sell order; or the bottom qtr for a buy order	+3
Above the mid price for a sell order; or below for a buy order	+1
At the mid price; or below the mid price for a sell order or above for a buy order	0

Table 3.9: Shows updates to PII based on how well order executes

[4] The venue matching rules built for this project are explained later in chapter 4.

So for example, stock ABC, from the graph in fig 3a shows that the current mid market price at 10:33am is 145.45p. If a sell order, routed by SOR to one of the venues at that time executes at 145.52p, i.e. in the top qtr of the bid-ask spread, then the PII score for that venue would be incremented by 3. This indicator, as with all the others, will be kept and maintained by the SOR system.

Each venue would have a PII score for each stock traded, representing price performance based on all the completed historical orders previously sent by SOR. Again, as with the venue trading cost, this indicator can be used to help the router send to the venues that may provide better prices. A time decay factor can also be applied, similar to that discussed in the historical trading activity indicator earlier. This would ensure that the PII values can adjust as time goes by and in different market activity.

3.3 SOR Probability Formula

In order to present a meaningful probability value for each venue, the routing indicators discussed so far would need to be weighted and combined in some way. All of liquidity, price and cost play their part in determining the overall ranking of each dark pool venue when it comes to which one is most likely to execute trades best. A simple formula can represent these indicators as follows:

Execution Probability, for each dark pool venue n :

$$P(n) = \frac{(rank_n)}{\sum_{i=1}^{k}(rank_i)}$$

Where venue $rank_n$ is calculated as follows:

$$rank_n = \frac{(VTC_n)^{-1}.\ f_1}{\sum_{i=1}^{k}(VTC_i)^{-1}} + \frac{(rHTV_n).\ f_2}{\sum_{i=1}^{k}(rHTV_i)} + \frac{(mHTV_n).\ f_3}{\sum_{i=1}^{k}(mHTV_i)} + \frac{(PII_n).\ f_4}{\sum_{i=1}^{k}(PII_i)} + ITA_n.f_5$$

Where

k = Total number of venues (where $0 < n \leq k$)

VTC = Venue Trading Cost (in basis points).

$rHTV$ = Historical Trading Volume experienced by router, i.e. the adjusted volume of shares traded for a given stock with time decay factor applied.

$mHTV$ = Market-based Historical Trading Volume, i.e. trade reports received from the market; again volume is adjusted with time.

ITA = Immediate Trading Activity (set to 1 when a recent order executes fully by venue); this expires to 0 within a finite time (5sec)

PII = Price Improvement Indicator (Represents how well a price of an order executes on a venue compared to the mid market price).

f = Adjustment Factor (where $\Sigma f_i = 1$) This is the weighting given for each element of the equation to achieve an appropriately balanced probability. Default: f_1 = 0.15, f_2= 0.2, f_3= 0.1, f_4= 0.15, f_5= 0.4

From this formula, the venue trading cost calculation (VTC) is inverted, since the lower the cost the better or higher the preference value should be. For each part of the probability formula (VTC, rHTV, mHTV, PII and ITA) the adjustment factor f is used to vary the weighting and place more emphasis on one element over the other based on the requirements that can be set by the trader. The formula shares similar volume sensing concepts with the algorithms presented by Almgren, Ganchev [5, 63] et al, in that it samples available liquidity and then

subsequently re-adjusts its venue probability distributions after receiving feedback from each routing. However, our formula not only considers the router's own traded volume but also volume received from the market, combined with price and venue execution cost factors. As noted earlier though, any experiences by the router its self is more favoured than market trade reports, hence rHTV data is considered better quality than mHTV. This is again reflected in the adjustment factor weightings.

The almost real time, immediate trading activity (ITA) that the router may experience from a venue or venues is considered the most valuable and is extremely time sensitive, as explained earlier. The adjustment factor should hence be set so that it places a lot of emphasis on this activity ensuring that the overall probability balances in favour of those venues. This would only of course be the case for a very short period of time immediately after such activity is detected. The ITA would then be neutralised back to nil, bearing no effect in the overall calculation, as explained earlier in section 3.2.2.

The trading cost and price indicators (VTC and PII) are both given a lesser but equal weighting, ensuring that they too play their part in the routing decision. Again, the adjustment factors applied here could be customised by the trader, responding to various needs of both the order and general market conditions.

Hence going back to the formula calculation, and in light of the above, the following adjustment factor weightings shown in fig3c could be used for the probability formula:

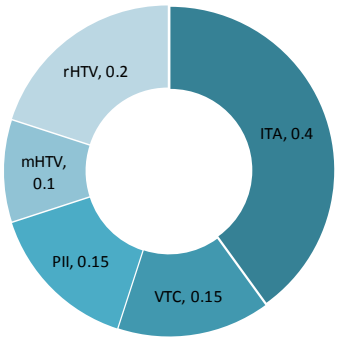


Fig 3c: Shows the weighting distribution of each element of the formula

The ITA will clearly play a significant part of the routing decision when it is detected. However, its effect is short lived and subsequent routing decisions will ignore it all together as soon as it expires.

An example will now be used to demonstrate the full formula in action. This will show how the routing decision will be made based on the outcome of the probability calculation. Again, four dark pool venues are present to choose from and the following analytical data is captured and stored within the SOR system:

Venue	VTC	rHTV	mHTV	PII	ITA
DPa	0.3 *bps*	12,000	54,500	25	0
DPb	0.7 *bps*	5,800	42,000	14	0
DPc	0.5 *bps*	1,000	4,000	5	0
DPd	0.3 *bps*	9,500	21,000	32	0

Table 3.10: Shows an example data set for each routing indicator across the venues

By applying the probability or preference for each indicator (VTC, rHTV, mHTV, PII and ITA), based on the probability formula discussed earlier, the following values in table 3.11 are obtained.

Venue	VTC probability	rHTV probability	mHTV probability	PII probability	ITA
DPa	0.330	0.424	0.448	0.328	0
DPb	0.141	0.204	0.345	0.184	0
DPc	0.198	0.035	0.032	0.065	0
DPd	0.330	0.335	0.172	0.421	0

Table 3.11: Shows the results of each indicator based on the probability calculation

So for example, the overall execution probability for DPa would be calculated by obtaining the rank for each venue as follows:

$$\text{rank (DPa)} = (0.330\,.\,f_1) + (0.424\,.\,f_2) + (0.448\,.\,f_3) + (0.328\,.\,f_4) + (0\,.\,f_5)$$

Using the adjustment factors defined earlier for each indicator (i.e. f_1 = 0.15, f_2= 0.2, f_3= 0.1, f_4= 0.15, f_5= 0.4) the calculation is made as follows resulting in an overall ranking for the venue:

$$rank\ (DPa) = (0.0495) + (0.0848) + (0.0448) + (0.0492) + (0) = \mathbf{0.228}$$

Similar calculations can be applied in order to get the rest of the venue rankings. The execution probability can then be obtained for each venue by: $\frac{(rank_n)}{\sum_{i=1}^{k}(rank_i)}$

Venue	Rank	Execution probability
DPa	0.228	**0.367**
DPb	0.148	**0.238**
DPc	0.049	**0.078**
DPd	0.196	**0.315**

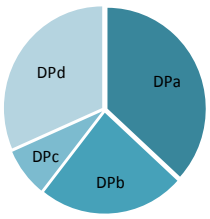


Table 3.12: Shows the final execution probability values for each venue

The previous example in table 3.11 did not show the ITA (Immediate Trading Activity) flag being used in any of the venues. To demonstrate its effect, suppose the router receives a successful execution for a recent order it made against DPb, the ITA flag would hence be set, as shown in table 3.13. This would significantly increase the execution probability value for DPb.

Venue	VTC probability	rHTV probability	mHTV probability	PII probability	ITA
DPa	0.330	0.424	0.448	0.328	0
DPb	0.141	0.204	0.345	0.184	**1**
DPc	0.198	0.035	0.032	0.065	0
DPd	0.330	0.335	0.172	0.421	0

Table 3.13: Similar to *table 3.11* but the ITA flag is now set for DPb venue

Venue	Execution probability
DPa	**0.228**
DPb	**0.548 <**
DPc	**0.049**
DPd	**0.196**

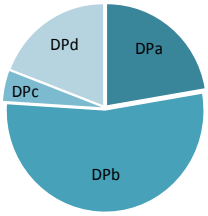


Table 3.14: Shows the increase in probability value of DPb when the ITA flag is set.

The top venues favoured by the router from the above would be DPb followed by DPa. The venue DPc is seen less likely to offer good liquidity based on the data captured at this stage.

The SOR system will calculate instant real time probability values for each venue at the time of routing. These probabilities, as explained earlier, will change and adapt as the system "learns" more about where liquidity is executing, therefore continually adjusting its valuations on each of the dark pool venues.

3.4 Order Splitter and Re-routing logic

The smart order routing system will send an order to the dark pool with the highest execution probability, based on the formula detailed so far. However, as discussed earlier in chapter 2, there may be value in splitting up the order into smaller sub orders and routing them across not one, but multiple venues.

So continuing from the previous example of the venue probability calculations in table 3.14, the SOR system would rank DPb as being the first choice to use, since it has the highest probability value[5]. Hence a single order lot of, let's say 9,000 buy shares, would be routed in full to that venue, as shown in fig 3d below. If we assume that the actual hidden tradable liquidity available at this stage in each venue is 5,500 sell shares in DPa, 2,500 in DPb, zero in DPc and 500 in DPd, then only a small proportion of the order will be completed by the initial routing decision.

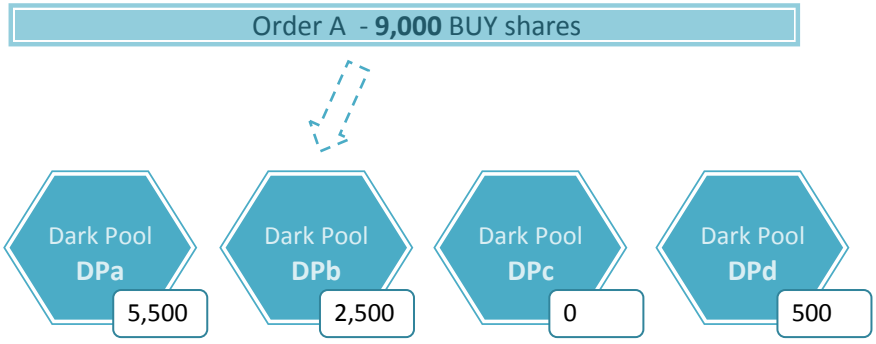


Fig 3d: Shows the routing of a sample order to a single venue (DPb)

Since the SOR system decided to route the entire order to a single dark pool venue, albeit the one it considers best based on the probability calculation, only 2,500 of the initial 9,000 share order was completed, i.e. another 6,500 was still outstanding.

It would make better sense to split the order and allocate smaller proportions across multiple venues. This order splitting mechanism is featured in different implementations of SOR systems in use today, including GSAT [22], SORBET

[5] Note that while DPb may have the highest probability ranking calculated by the router as discussed, it does not necessarily mean that it's actually the best, or has the most liquidity at that moment of time. The router can only predict but can never have 100% certainty. This is reflected in the example above where although DPb is good, DPa is actually better.

and CAPIS [21]. Going back to the above example, rather than send the whole order as a single lot, if it was split in half, with each 50% of shares being submitted to the top two venues based on the probability ranking, then this would increase the chances of the order being filled more quickly.

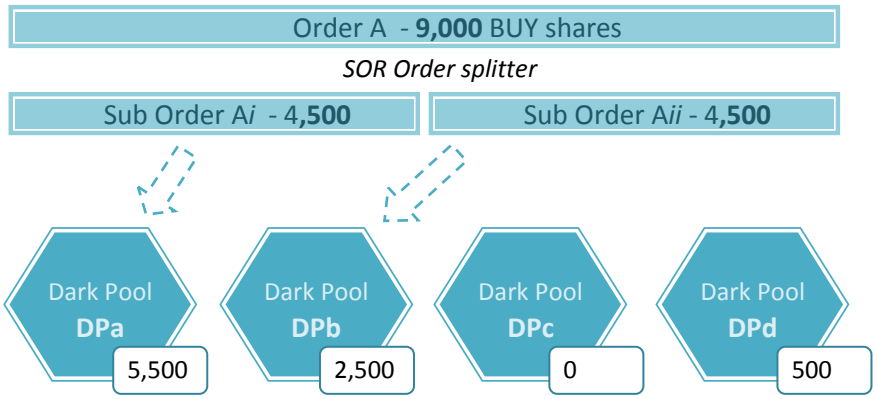


Fig 3e: Shows the splitting of the order and routing to multiple venues

With the order split in half and simultaneously sent to the top two venues at the same time (DPa and DPb), the result is that more of the order was executed successfully in this example. Only 2,000 of the initial 9,000 share order remained after this routing decision.

Another approach is to further split the order, submitting equal amounts across all dark pools. However this may not be the most optimal way since the liquidity is not spread uniformly throughout these venues. This will be an area that will be evaluated later. A better approach could be to split the order into different uneven sizes, creating large and small lots; the router can then submit the larger lots to the higher probability ranked venues and smaller lots to the lower ranked ones.

Subsequent Order Re-routing

After the first round of routing, the SOR system will receive feedback on how well its orders executed against the venues, and how much of the order remains. A decision must then be made on what to do next, whether to leave the remaining unfilled order in the dark pools hoping to find liquidity from future participants, or to pull them and re-route elsewhere.

Going back to the example above, the "before" and "after" order volume and remaining liquidity in venues is as follows:

1st round (to DPa)	**Before**	**After**
Sub Order A*i*	4,500 BUY	0
DPa (hidden liquidity)	5,500 SELL	1,000

1st round (to DPb)	**Before**	**After**
Sub Order A*ii*	4,500 BUY	2,000
DPb (hidden liquidity)	2,500 SELL	0

Table 3.15: Shows the 1st round remaining volume from the orders and dark pools used

Hence 2,000 shares from Sub Order A*ii* remains incomplete, sitting in DPb. As for Sub Order A*i*, then this has fully executed and completed against DPa. From the discussions earlier, this feedback when received by the SOR system will be used to update the router's analytical repository of indicators, which in turn feeds into the probability formula.

Since a complete execution of an order has just been experienced in DPa, the ITA (Immediate Trading Activity) flag for this venue will be set to 1, causing its execution probability ranking to increase. The routing logic can now act immediately on this indicator by sending a cancel request to its remaining incomplete order in DPb and re-routing the order to DPa, where it is perceived that further liquidity still exists at this very moment of time.

The re-routing to DPa will only occur however once the SOR system receives a cancel acknowledgement from DPb to ensure that no double execution happens.

2nd round (to DPa)	**Before**	**After**
Sub Order A*ii*	2,000 BUY	1,000
DPa (hidden liquidity)	1,000 SELL	0

Table 3.16: Shows the 2st round volume before/after the re-routing occurs

After the second round of routing is made, the remaining incomplete portion of the order is 1,000 shares. Again further re-routing can occur to redistribute this or leave it in the venue.

The sequence of events explained so far can be summarised in fig 3e as follows:

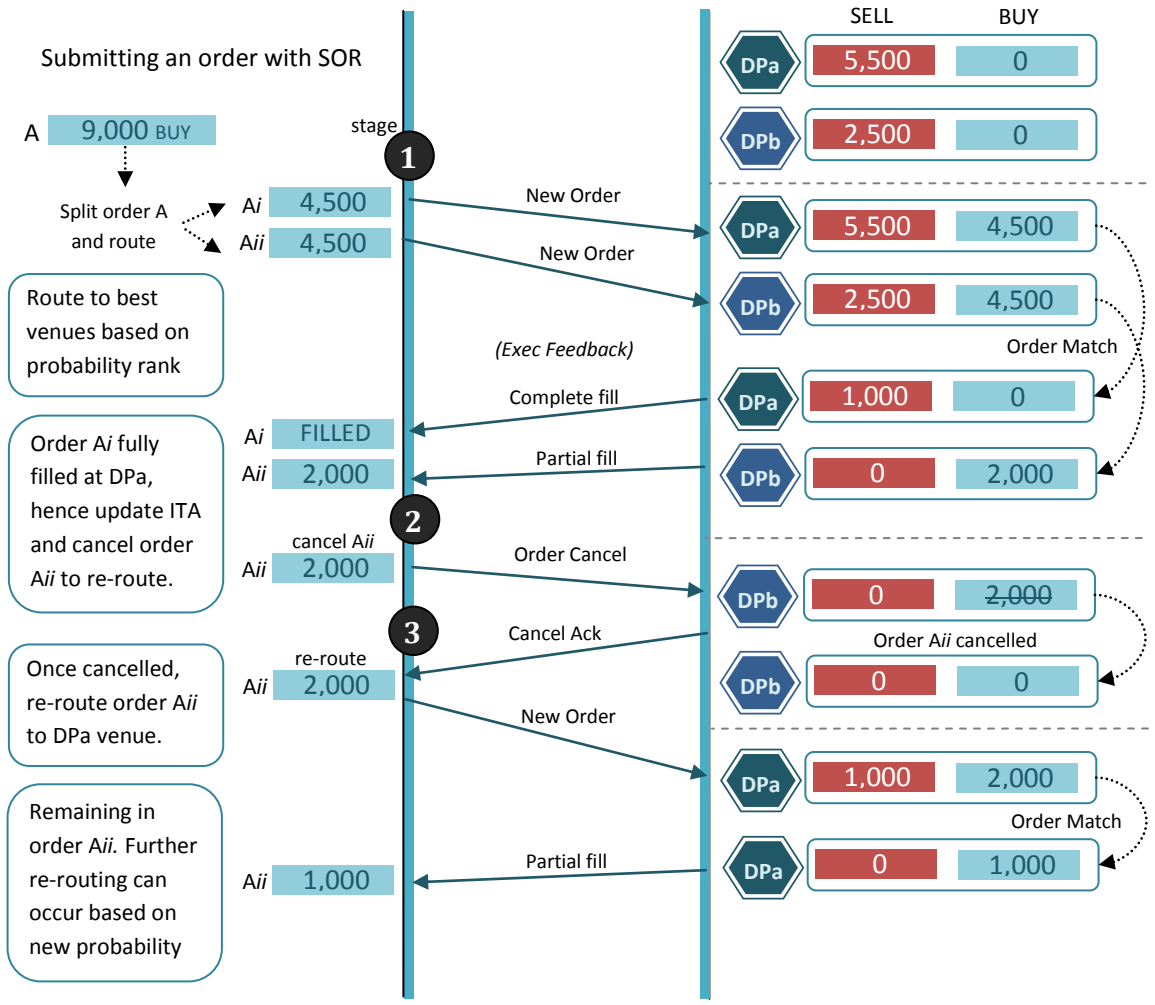


Fig 3e: Shows the order routings and actions by SOR to the dark pool venues

By splitting the order into smaller parts, the SOR system can send them simultaneously against multiple venues, as shown above in stage 1. However, after feedback of execution from the dark pools is returned, and when the router identifies a highly ranking venue, it cannot easily re-direct further liquidity to that until it has cancelled its unfilled orders from previous venues. This is shown in stage 2 above.

Only once it has received an acknowledgement of the order cancellation -at stage 3- can it then re-route to a new venue. This process introduces extra latency in the execution process, which could result in the liquidity that was initially available in the venue of interest disappearing by the time the cancellation and re-routing occurs.

Split and Reserve

A possible way to reduce cancellation/rerouting is to optimise the order splitting logic by allowing the SOR system to only route a proportion of the order to the venues and keeping a remaining part as reserve. This reserved part of the order could be used in subsequent routings when immediate opportunities are identified; therefore aiming to reduce the need for constantly cancelling orders.

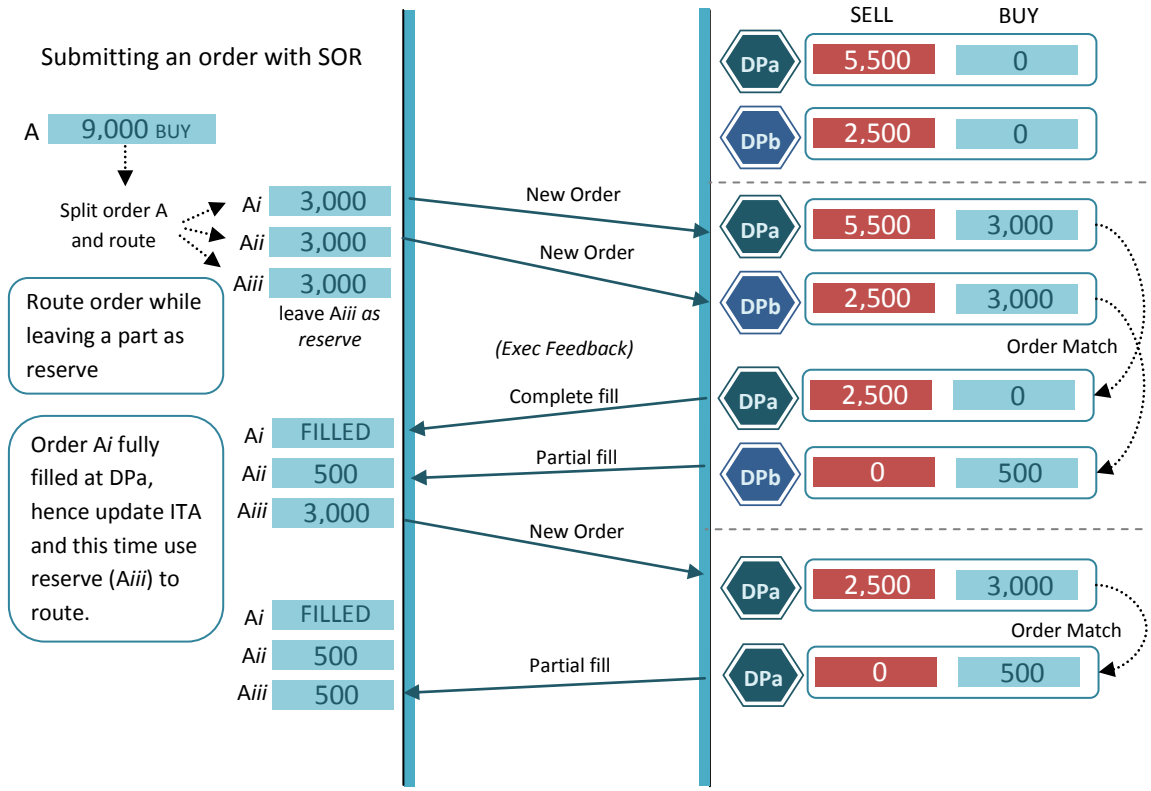


Fig 3f: Shows the order routings and actions by SOR to the dark pool venues

Using the same example as before, the SOR system now splits the order in three parts. It routes two parts to the top venues based on the probability rankings, and keeps the remaining 3rd part on hold as reserve; that way, once it discovers that DPa has a high chance of executing further liquidity, it can immediately route its reserve order A*iii* to it without having the need to cancel its existing A*ii* order first.

Of course this is a very simple example, and in reality many factors play their part, including the size and type of the order; but it does demonstrate a concept, which many implementations in industry utilise today. Our SOR system will feature similar (order split and reserve) concepts, which can be both adjusted and refined. This will be touched upon later in the evaluation chapter.

3.5 Execution Management System and SOR

In order for the router developed and discussed so far to function, it must be integrated into an appropriate system that can manage the full automation and deal with the order flow, allowing access to the many different execution venues. Such a system is often called an Execution Management System (EMS) [27]. This software platform manages the complete execution of the order from entry to completion, aided by an array of components, including algorithms and real time market data feeds.

The following shows the order flow and design of a possible EMS, detailing how the SOR components would integrate into the overall system.

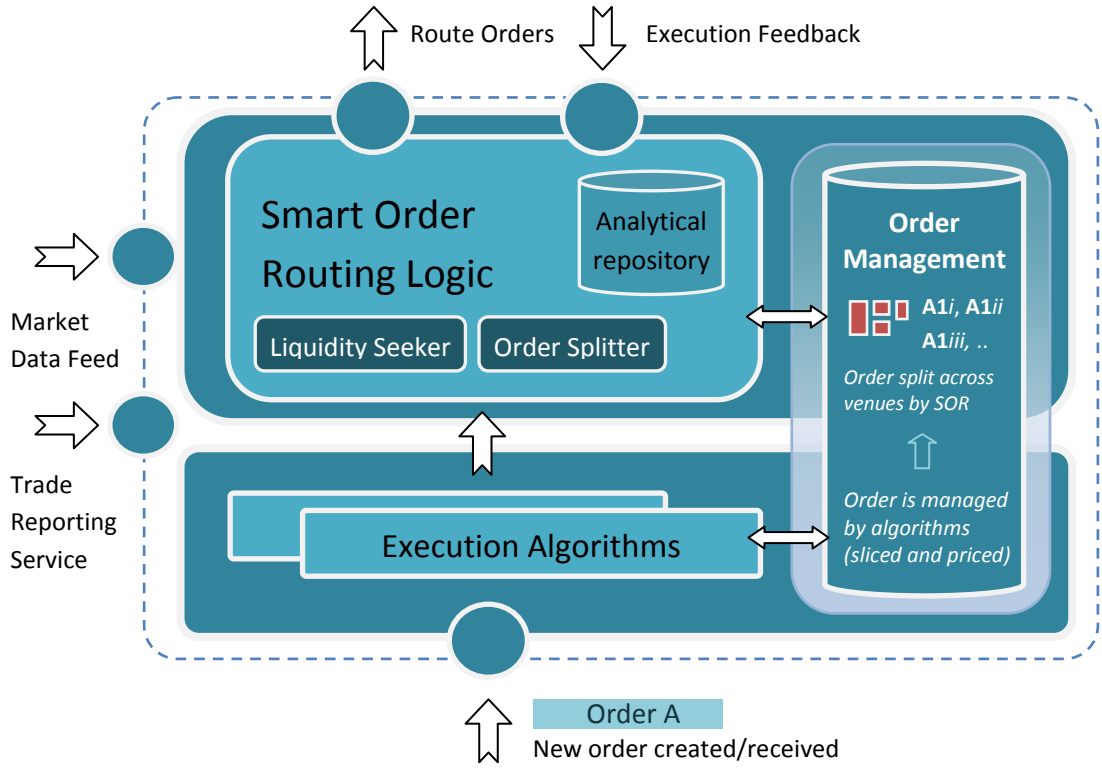


Fig 3h: Shows the SOR components within an Execution Management System

When a new order is created, it can either be sent directly to a specific venue, or be put through an execution algorithm and be submitted via smart router. The later would require a system similar to the above.

A trader would submit the order, specifying an algorithmic execution strategy. This strategy would handle the entire overall management and execution of the order, sending slices of order chucks to the SOR system ready to be split, routed and executed against venues. There are different execution strategies in use

today, with the most common being the Implementation Shortfall, Time Weighted Average Price (TWAP) and the Volume Weighted Average Price (VWAP) strategies[6]. It is beyond the scope of this project to go further into this part of the system, but these are strategies that aim to track the overall progress of the order against different market indicators and benchmarks. They may slice the order and prepare it to be executed across different "time buckets" throughout the day, spreading the risk involved in trying to execute all at once, which may otherwise prove difficult and costly.

When "chunks" of the order are passed from the main execution algorithm to the SOR system, it becomes immediately active and starts to be "worked". The smart router begins to split and route the order across the venues based on their execution probability rankings, which are calculated from the routing indicators as discussed in detail earlier. These indicators along with all the data used in the calculation are stored in the Analytical Repository, shown in fig3h above.

Inputs from both real time market data and trade reports are also used to feed into the EMS and utilised in SOR. When feedback of execution (from routed orders) is received from the different venues, this is captured in the Analytical Repository, further aiding in subsequent routing decisions.

The progress and state of all the orders are managed within the Order Management database, shown in fig3h above. Further details on how this will be developed, as well as the other components of the system, will follow in the implementation chapter later.

In summary, this chapter discussed the routing algorithm for a SOR system to be developed, including the execution probability and the main indicators to be used. It also looked at order splitting, rerouting and liquidity seeking functionality and how this fits into an EMS design.

[6] In a nutshell, TWAP is the average market price over a particular time interval. A strategy based on this would slice and execute trades aiming to achieve TWAP across different time intervals. VWAP on the other hand balances execution with the volume of shares traded. [39]

Chapter 4

Multi-Agent Simulation Design

In this chapter we outline the main entities and elements of the multi-agent simulation, detailing the design concepts, characteristics and roles of each part. We show how each model compares to the real world as well as the data and assumptions being made. The aim is to present a realistic simulation model that can be used to test and evaluate the SOR system developed so far. This chapter focuses on outlining the business logic, roles, parameters and overall characteristics of the models, while the following chapter will discuss the technical details and actual implementation of the system.

4.1 Overview

As mentioned earlier, the use of multi agent simulation within the financial markets is an important and growing domain, highlighted by the many projects and research available today. Streltchenko, Yesha et al [28] published works reviewing the use of agent based simulations in predicting different market behaviours, moving away from the limitations of traditional computational mathematics. The development of such a multi agent system here -in this case- will help test and demonstrate the smart order routing capabilities discussed and developed so far. The following sections will detail the complete model structure.

4.2 Entities Model

4.2.1 Dark Pool Venues

The dark pool venue is the key place where orders will be matched and executed in the simulation. A total of four (DPa, DPb, DPc and DPd) will be developed, each having its own order book and able to manage, match and execute orders from any of the participating parties. Market data feeds -discussed later- are consumed by each venue, providing timely prices for all the stocks traded. This allows them to cross orders correctly, especially market orders[7]. Since dark pools are always secondary market venues for the stocks it trades in, it does not

[7] Market orders are those that are to be executed at the current market price. This is different to a limit order, where it can only be executed at the price set with the order.

directly influence or make the market price. As mentioned earlier, it simply bases the price published by the main market or exchange as the indicator for matching and executing its trades. Therefore, the dark pool venues developed in the simulation will not affect the published price directly, and hence only act as a "passive" consumer of it.

Each venue will also send out a post execution report of all orders it completes (within three minutes) to the trade reporting service. This reflects the real world regulatory requirements for trade reporting imposed within financial markets [17]. This data will also be used by the router. The high level model of each dark pool and its interactions in the simulation will look as follows:

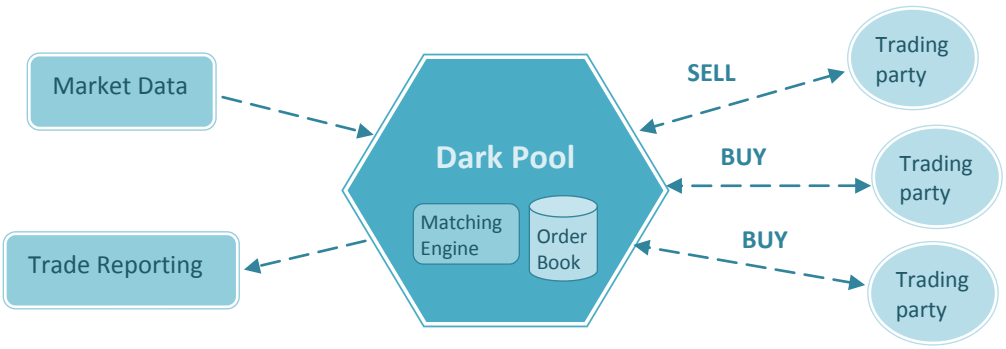


Fig 4a: The dark pool venue model along with its interactions with the other entities

Order Book and Matching Engine

At the heart of each dark pool -as with any electronic trading venue- lies the limit order book, which will contain a live list of the entire bid and ask orders submitted by the trading parties. The venue's matching engine will scan these orders and when a valid match between a buyer and seller is found it will cross the trade and complete the order. When multiple bid and ask orders are in the order book the matching engine will follow the following criteria:

Criteria	Description
Price priority	Orders with the most favourable price are matched first.
Time Priority	Where more than one order shares the same price, then the first order to enter the order book is matched first.

Table 4.1: Order matching criteria

When an order is received in the venue's order book, it is assigned a timestamp. This timestamp is used to prioritise orders in the book with the same price. The

order entered earliest at a given price gets matched and executed first. When a new order is entered, a scan is made to check if it is immediately executable, i.e. if the price is at or better than the best bid or ask order; if so then the match occurs and execution is made. The use of price and time priority rules in the matching engine to be developed here mirrors closely to the behaviour of most venues out in industry today. All exchanges and displayed venues as well as dark pools such as Turquoise, NYFIX, ITG Posit, Sigma X, PIN Cross, Knight et al [29] use price and time priority in their order matching engines.

Orders may not always execute at a single price, but may generate several sub transactions at different prices depending on the total available quantity at a given price level on the opposite side of the order book. This can be demonstrated in an example as follows:

Order Book - Stock ABC					
BUY Orders (BID)			**SELL Orders (ASK)**		
Time	***Volume***	***Price £***	***Price***	***Volume***	***Time***
11.16	3,000	172.5	172.3	1,000	11.15
			172.5	1,500	11.02
			172.5	500	11.08
			172.7	7,000	11.00

Table 4.2: Sample order book for stock ABC showing bid and offers

A 3,000 buy order for stock ABC enters the order book priced at £172.5, as shown above. There are four sell orders active on the opposite side of the book, of which three are priced at or below the bid of £172.5 and hence tradable. Since the 3,000 bid order is large, it requires several sell orders to complete the execution, as shown below:

Executed 1,000 at £172.4 (completed at mid-point of bid/ask)

Executed 1,500 at £172.5

Executed 500 at £172.5

Complete order FILLED: 3,000 @ 11.16am

All four dark pools to be developed in the simulation will follow a similar order matching and execution behaviour to above. The order types supported by the venues will be:

Order Type	Description
Market Order	No specific price limit is set with order and hence it's matched to the best available opposite bid/ask price (based on the market price).
Limit Order	A specific price limit is set with the order, and hence it may only be executed at a price better or similar to that limit

Table 4.3: Order types supported by the venues

The following cases demonstrate the rules built into the trading venue to match orders depending on their price/type. Fig 4b and 4c show limit orders, and hence a match only occurs when the limit price is satisfied. Fig 4d shows a combination of both limit and market orders. Here the matching price will always be that of the limit order. This can create a price improvement for the market order –as is the case for fig 4d, where it receives a sell price of 171.7 rather than 171.

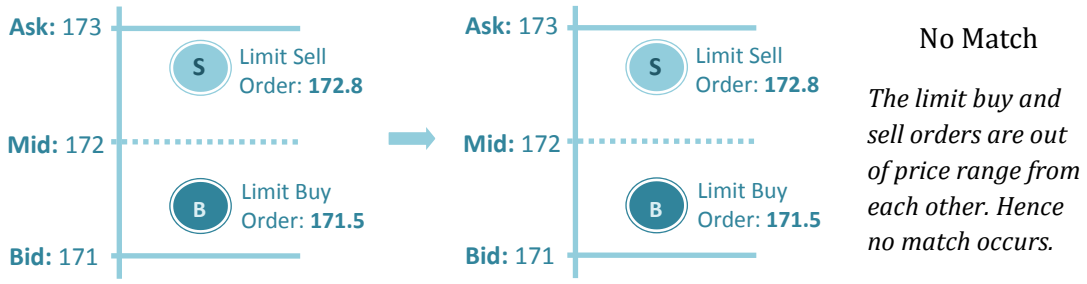


Fig 4b: No match can occur as both orders have untradeable limit prices

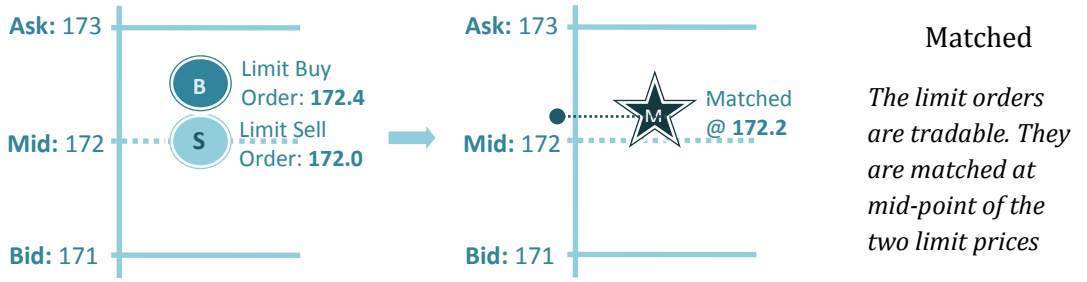


Fig 4c: The match occurs at the mid-point of the limit buy and limit sell orders

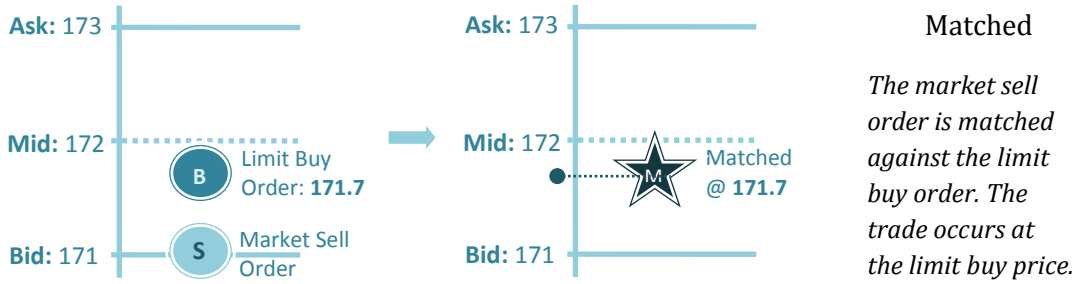


Fig 4d: In the case of a market and limit order, the match always occurs at the price-limited order

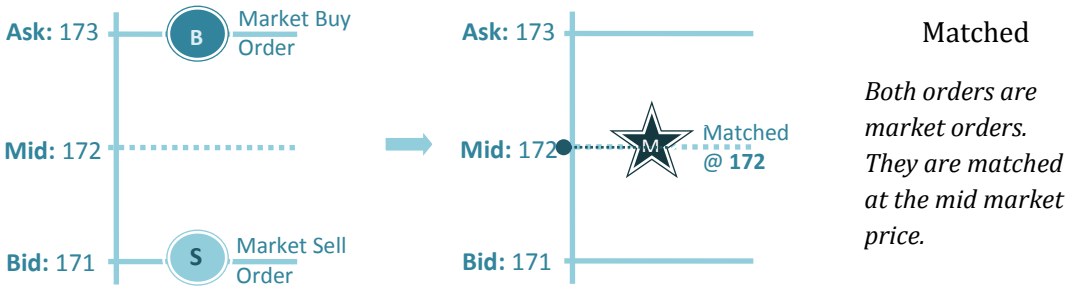


Fig 4e: Where both are market orders, they are always matched at the mid price point.

For venue fees, each dark pool will have its own basic per-share charging model. As noted earlier, many of the venues in operation today typically charge anything between 0.2bps to 1bps. Nasdaq Neuro charges 0.2bps, while Turquoise's dark pool charges 0.3bps per share [25]. Again as mentioned before, some venues may charge membership fees or provide rebates and other promotions depending on how participation of liquidity in the venue is made. In order not to overly complicate our model, we will only focus on the standard commission rates per share. The following will be used in the simulation:

Venue	Trading Cost Fee
DPa	0.3 bps *per share*
DPb	0.7 bps
DPc	0.5 bps
DPd	0.3 bps

Table 4.4: Dark pool trading cost charges in bases points (1bp = 0.01%)

This section attempted to explain the general functions of the dark pool entity to be used in the simulation. The entity's interaction with the others as well as all the possible inputs and outputs to the model will be covered shortly.

4.2.2 Trading Parties

The trading parties are the independent entities that will trade against each other at the dark pool venues. They will reflect the trading population within the simulation. A total of 20 will be developed with each sharing some basic functionality with the others, and able to trade autonomously at the venues. Every trading party is also able to accept orders on behalf of clients -acting as a broker- and submit them to any of the dark pools for execution. They may amend an order -adjusting the size or price- or even cancel and move it to other venues. Trade reporting and market data feeds are consumed by each trading party, providing updated market prices throughout the simulation for the stock traded.

This may influence how each party trades or adjusts any of its existing open orders. Again, since all the trading in the simulation is happening in dark pools, the assumption is that the stock price coming from the market data feed is not directly impacted by this type of trading. This is because the stock price is mainly formed by trading in the primary and displayed exchanges as explained later.

In order to evaluate and demonstrate the SOR capabilities, one of the trading parties will use the SOR system developed so far. The rest of the parties will use other routing mechanisms such as trying each venue in sequence, favouring a subset over others as well as random routing behaviours, all discussed later in the report. The following diagram in fig4f shows the trading parties along with SOR usage as well as the other main entities within the simulation:

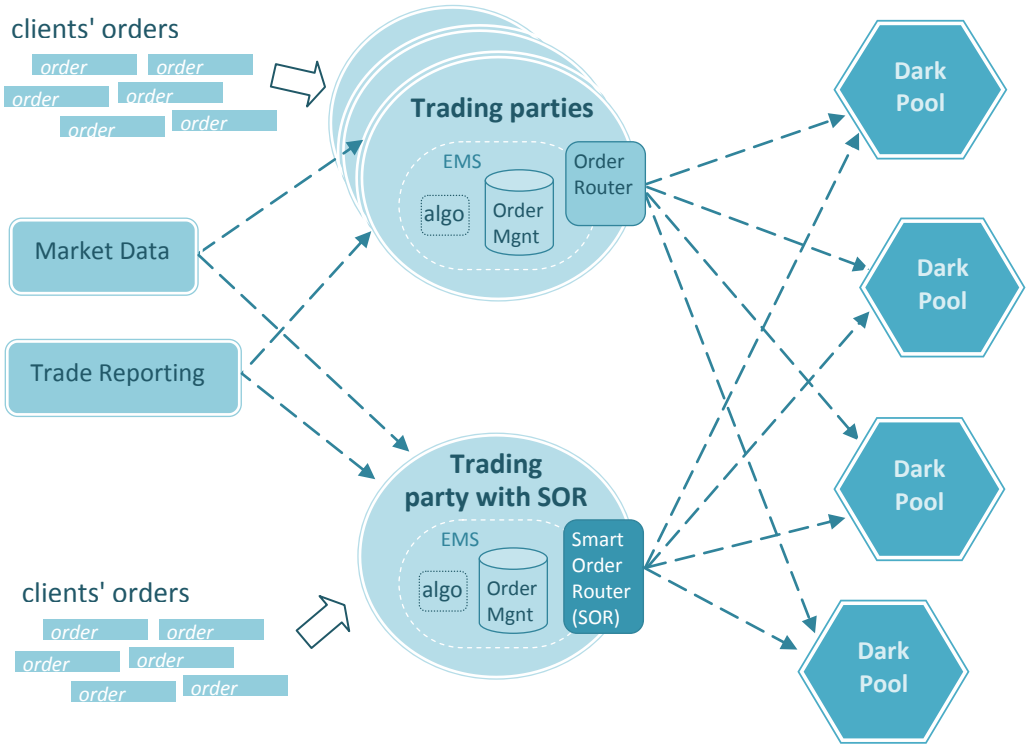


Fig 4f: The trading parties in the simulation with SOR utilised in one of them.

It is assumed that all trading parties in the simulation will trade algorithmically; in other words the execution and routing of orders are fully automated. Each trading party takes on orders on behalf of clients, processing and completing them through its Execution Management System. The routing capabilities of the trading party that uses SOR will be compared against the other trading parties. Further details of the simulation environment will follow later in the chapters.

4.2.3 Market Data Service

This represents the market data feed, publishing continuous price updates for stocks (through time) to all the entities in the simulation. Every time a price update occurs, the market data service publishes a message reflecting this, which is consumed by everyone that is subscribing to the service. The dark pool venues receive these updates which are used to assist in executing trades within their order book. The trading parties also receive this to feed into their trading decisions. A number of major and substantive market data providers in industry today include those operated by Bloomberg [37] and Reuters [38]. They provide the data though messaging systems which clients can subscribe to in order to receive updates.

A similar, but simple market data service was developed for the simulation. It uses a real historical price feed[8] for a stock and then "re-plays it" through the simulation timeframe –as shall be discussed later. Each price update message published also includes a timestamp along with the market bid, ask and mid price for the stock. Although a wide range of other market data is normally published, our market data service resorts to providing just these simple price updates for use in the simulation. Following is a sample of these messages:

Stock	Time	Mid Price	Bid Price	Ask Price
ABC	10:29:20	218.45p	218.50	218.85
ABC	10:31:18	218.25p	218.30	218.50
ABC	10:31:27	217.75p	217.20	218.00

Table 4.5: Market Data sample price updates for a stock

As mentioned earlier, the price updates are formed mainly by trading activities in open and displayed exchanges; hence although the trading parties in our simulation will affect the prices in the dark pools, they will not directly affect the main (externally published) market price. In other words, the market price will be passively consumed by these entities. Although in reality it is not as simple as that -where market price effects are far more complex- this assumption is frequently found and made in literature; example with Ye [4], Kratzm [30] *et al* looking at price formation and liquidation in exchanges and dark pools.

[8] The tic-by-tic price feed was captured and sourced via the Reuters Market Data Service [38] for a given sample stock across a few days. Sample in appendix 6.3.

4.2.4 Trade Reporting Service

After an order executes and completes, the market must be notified of the trade. The venues must send an execution report message to a trade reporting service which in turn publishes this information to everyone. This post-trade data can be utilised by the smart router as noted in the previous chapter. There are many reporting facilities available in industry today. Many of the main regulated exchanges as well as third party vendors, such as Markit BOAT [36], offer trade reporting services.

A simple trade reporting service was developed for the simulation. It accepts execution reports from venues for completed trades and then republishes them back to the market shortly after. It will also aggregate multiple reports together and publish them as a single report. This follows closely to the real world where it is often not possible to tell which venues executed which trades [32] as they are published in a consolidated form or simply as "over the counter" trades by the reporting service[9]. Each trade report in the simulation will contain a timestamp along with the stock and volume traded.

4.3 Interaction Model

How entities -the agents- interact with each other in the simulation is an important factor in ensuring that the system behaves as intended and as realistically as possible. The main interactions between the different entities must be modelled to reflect near real world behaviour.

4.3.1 Overview of Main Interactions

A trading party interacts mainly with the dark pool venues. It sends a new order message to a venue seeking an execution. The venue receives the order and then places it in its order book. Many of such orders may be sent simultaneously, split across more than one venue. The venue's matching engine checks if the order is tradable against its current list of live orders. When an execution occurs, the venue sends an execution report message back to both trading parties informing them of the trade and of any remaining unfilled order. It also sends a message to

[9] On the time of writing this, there have been some discussions from the regulatory bodies and industry to improve post-trade transparency and force the identification of venues (that executed the trades being reported). However it remains to be seen how this evolves in reality [32].

the trade reporting service, to inform the market of the trade. A trading party may cancel or adjust an existing order it has with a venue. An acknowledgement of this is then sent back from the venue. All entities, be it venues or trading parties receive constant market data updates informing them of price moves in the stock they are trading throughout the simulation timeframe.

The following input and output summarises the main interactions between the venues and trading parties as well as the other entities:

Input to venue	Description	Received from
New Order	A new order is received	Trading party
Cancel Order	An order cancel request is received	Trading party
Amend Order	An order amend request is received	Trading party
Market Data Update	Stock price updates are received	Market Data Feed

Output from venue	Description	Sent to
Order Execution Rpt	An execution report is sent after an order is completed	Trading party
Order Cancelled Ack	An acknowledgement is sent when an order is cancelled	Trading party
Order Amendment Ack	An acknowledgement is sent when an order is amended	Trading party
Execution Report	A consolidated execution report is sent after an order is completed	Trade Reporting Service

Input to trading party	Description	Received from
Pre-trade	*No pre-trade order book/volume data is received from dark pool venue*	*n/a*
Order Execution Rpt	An execution report is received after an order is completed	DP Venue
Order Cancelled Ack	An acknowledgement is received from venue when an order is cancelled	DP Venue
Order Amendment Ack	An acknowledgement is received from venue when an order is amended	DP Venue
Market Data Update	Stock price updates are received from open market	Market Data Feed
Execution Report	Consolidated execution report from the market	Trade Reporting Service

Table 4.6: Summary of messages used in the main interactions

Again, as mentioned in the earlier chapters, no trading party is able to receive or request any details of the orders -i.e. liquidity- currently active in the order book from any of the dark pool venues; since by the very nature of these venues, all liquidity in them are always hidden and not published.

4.4 Stock and Agent Characteristics

4.4.1 Tradable Stock

In order to control the variation and complexity of the simulation, the trading parties will trade one single stock amongst each other at the venues. The characteristics of it will be modelled against real stocks including ensuring that the volume being traded during the simulation is realistic and comparable to a similar type stock in reality. For example, trading in large cap stocks such as Vodaphone and British Petroleum can be in their tens or even hundreds of millions a day, while mid-cap stocks can be in their millions or even less. Data in appendix 1 shows a selection of traded stocks and their historical volume distribution across lit and dark pools[10]. A stock, ABC, will be modelled and used in the simulation and its volume will be comparable against those.

4.4.2 Agent Population and Trading Characteristics

As noted earlier, a number of trading parties will participate in the simulation, each trading against each other on the four dark pool venues. They are mainly acting as brokers, submitting trades on behalf of their own clients. Hence each will have many trades to complete across the simulation period. Financial trading environments are very complex and dynamic, where different trading characteristics occur. There will be those that are looking for urgent execution and within a short time space, while there will be others that will take a more passive approach, gradually working their orders and looking for a better deal. Some trading parties may need to trade very large orders, representing big financial institutions, while others may have smaller but possibly frequent lots. The routing and choice of execution venues also varies, where some may favour a selection of venues over others. The degree of order splitting and rerouting will also be different, depending on the nature and size of the order.

Although it is very difficult to create an accurate model of the trading that occurs in reality, one can reasonably incorporate -and make assumptions- on some of the characteristics discussed. Further details on this will follow in the evaluation chapter as different experiments and tests are made and analysed.

---

[10] Data was obtained from Fidessa's Fragulator tool (see appendix 1), which provides a breakdown of volume traded for a given stock and period, showing the distribution across the execution venues, which includes dark pools.

Chapter 5

Implementation

In this chapter we detail the actual implementation of the smart order router and the multi agent system presented. The technical solutions used as well as the design and development of each element will be outlined and discussed. The aim here is to provide a reasonable picture of how the system was implemented, the architectural design and technology choices made, as well as how this compares to industry.

5.1 Current Simulator Frameworks

There are a number of generic agent based simulator frameworks available which can be used to develop the models and the simulation. These include Repast, Ascape, Swarm and NetLogo [41, 42, 43 and 44]. They essentially provide the functionality and environment to manage the simulation, creating, executing, displaying and collecting the data from the models developed. Many are fully object oriented and are implemented in popular languages, such as Java and C#.

However, while some of these frameworks can be utilised to some extent in our solution, the decision was to develop the system and simulation without the use of these frameworks. This is based on a number of factors. While the main objective of this project is to evaluate and demonstrate some of the smart order routing capabilities discussed, another objective and challenge is to do so utilising current industry standard technologies and protocols, as used by the banking and financial world today. This secondary objective may not in its self directly influence the simulation results; however by standardising the technology/interface used, it could open the possibility of extending the simulation by allowing real components, such as other trading systems, to take part. This of course will require further work, but it does keep this possibility open. Such would be difficult by using any of the simulator frameworks mentioned above, since the components and models are essentially locked down to the specifics of each framework.

While these frameworks do provide out of the box implementations making it easier to collect and present the simulated data, the bulk of the effort, especially in our case, is largely around the building and coding of each component of the simulation (dark pool venues, trading parties, market feed etc.). This effort would be required regardless of the framework used.

Another aspect is the concept of time, where the majority of simulator frameworks use discrete time/events as opposed to continuous -scaled- real time. This of course has its many advantages in that time is defined by a set of atomic "ticks" on which various events can hook on to; this guarantees the order of execution of these events in each run. Such an approach however would be difficult to implement when building an open system that takes into account real time – or historical- market data and that could in the future also accept other outside agents (e.g. external trading systems) to participate in the simulation. The simulation will hence be developed around a real time clock, which may be scaled as mentioned earlier in the previous chapter. This of course will introduce cases where the order and timing of events may not be executed in the exact same way in each simulation run –e.g. due to latency or variations in CPU order execution of threads, and hence can affect the overall results. This is acknowledged and will be factored into the model testing and evaluation process by running each simulation several times to identify any reoccurring patterns.

5.2 Technical Considerations

In order to proceed with the development, the implementation of the system requires a combination of suitable development tools, software code as well as appropriate infrastructure to support the running of the simulation.

5.2.1 Development Tools

The main programming language chosen for the bulk of the development is Java -via the Eclipse IDE (Integrated Software Environment) platform [45]. Java is a mature, well supported object oriented language used extensively in both research and industrial strength applications. The Eclipse IDE, originally developed by IBM, is one of the most popular development environments for Java; being open source, it is widely used by both the academic and commercial community. A number of plug-ins and extensions to Eclipse will also be used to provide logging and data handling within the simulation. For data analysis and

graphing, Microsoft Excel will be used. Hence the simulation will output data which is imported automatically via Excel, leveraging the powerful built-in features of this application.

5.2.2 Messaging and Hardware Infrastructure

One of the technical objectives is to ensure that the multi agent system and simulation is scalable and readily expandable across not one, but multiple servers. This will allow the venues and trading parties etc to run on separate servers or platforms, creating an expandable more powerful overall simulation. To enable this, a suitable underlying messaging system that can carry the communication between the servers –and therefore agents- must be deployed. There are a number of high performance messaging systems used in the market today, with the most common being IBM's Websphere MQ and Tibco/Reuters RMDS messaging systems [46]. These are commercial industrial-scale systems that ensure the effective and efficient transmission of messages from sender to receiver, managing the queues and overall flow. Of course there are other possible solutions including building our own interface to handle the messaging traffic or using web services or even low level socket programming. However all these could significantly deviate from the main focus of this project; it may also be impractical when there are more suitable solutions that can be used. Websphere MQ was chosen due to the good level of support documentation available, however other systems would have equally been as suitable.

Following is a brief outline of the Websphere MQ implementation:

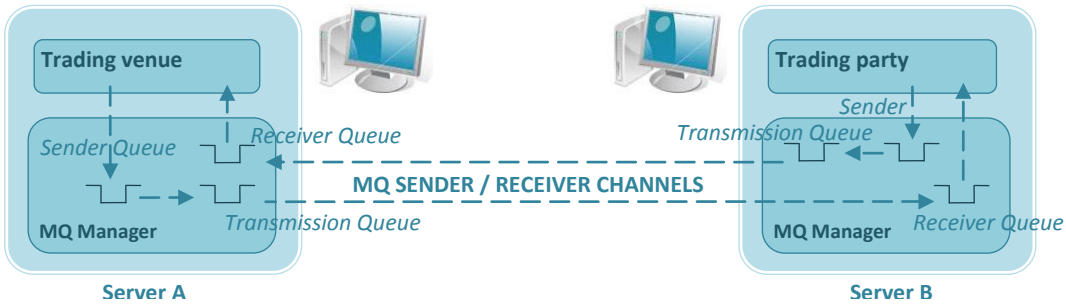


Fig 5a: Shows the Websphere MQ implementation across two servers

If a trading venue is deployed on Server A and a participating trading party in Server B, then each would communicate over the messaging system outlined above. Without going in too much detail into the technicalities of this, an MQ

manager is installed on each server, which takes care of the overall management and connectivity and allows message queues to be setup and configured. Message queues are required to hold incoming and outgoing messages from -and to- the different trading counterparts. Messages are transmitted over MQ channels between the different MQ managers [47]. The trading party for example would place a message on its local *sender queue*, which will then ultimately reach the remote *receiver queue* at the other end; there it will be picked up by the venue and processed. This works the other way round as well, where messages are sent back from venue to the trading party server, but over a different channel.[11]

Hardware Setup

A total of four server PCs have been utilised and configured for the simulation. Each is deployed with Websphere MQ and fully connected and set up to send and receive messages to each other, similar to the implementation above. By using multiple servers, components of the multi agent simulation can be spread across them and hence balance the load, allowing more interactions and messages to be processed within the simulation timeframe.

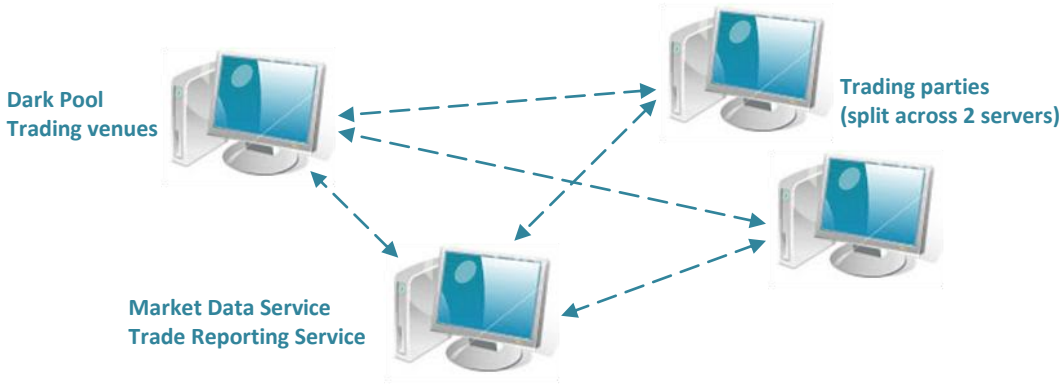


Fig 5b: Shows the distribution of agents in the multi agent simulation across the servers

Of course there is no magic number of servers, as it all depends on their power as well as the nature and complexity of the simulation; however four were practically available for this project and utilised. Their specifications can be found in appendix 2. It would have been difficult to conduct this type of simulation, given the processing requirements of the agents and components involved, with only one server. These processing demands include the venues'

[11] Further details of the technical and interworking of the messaging system can be found in the Websphere MQ Fundamentals book, published by IBM [47]

matching engine, trading parties and router logic calculations, as well as continuous and scaled real time -historical- market data feeds. When comparing our simulation to reality, and if we were designing industrial strength systems, then load balancing techniques would be very important. An array of servers would be configured in clusters to serve the one application to ensure that no failure or unacceptable load hits one server; as this could potentially cause financial losses due to e.g. considerable processing delays, especially in mission critical trading systems where timing is crucial. The importance of this area is widely covered in both the industrial and academic communities; publications include D. Thiben, T. Bourke et al [48, 49]. This is also important for our evaluation and results; hence the simulation will be monitored to ensure that its scale and complexity is kept at a level that is manageable for the servers.

5.2.3 The FIX Protocol

Having looked at the underlying infrastructure that will support the multi agent simulation, the message protocols used to exchange and coordinate between the agents will now be discussed. In order to build an interoperable system, with agents that are able to communicate in a standard that is common within the domain, i.e. the financial trading industry, a mature and widely accepted protocol will be used. This is the Financial Information Exchange (FIX) protocol; initiated in 1992, it is a messaging standard developed specifically for the real time electronic exchange of securities transactions, supported and adopted by the majority of the industrial financial trading systems today [50].

Although XML based implementations of FIX are available, the most common FIX messages are flat text-based and consist of tag-value pairs separated by a special delimiter character (ASCII 0x01). The tags are made up of digits and the values may include anything from strings, integers and timestamps. The messages are organised into header, body and trailer with the body holding the main business context of the message. The protocol specification covers a wide range of message types, with a definition of over 1,600 fields used across them, spanning the full interaction cycles required in electronic trading; from pre-trade market data requests, to new trade orders and execution report messages.[12] The following set of FIX messages will be used in the simulation:

[12] A full detailed structure of all the FIX messages can be found on the FIX website [50].

FIX Message	
NewOrderSingle	A trading party sends this message to a venue to submit a new order.
OrderCancelRequest	A trading party sends this to a venue to cancel an already submitted order.
OrderCancelReplaceRequest	A trading party sends this to a venue to cancel and replace a previous order.
ExecutionReport	When a successful trade occurs on a venue, the venue sends this message to the trading parties involved to confirm full or partial execution of their order. The message is also sent to confirm order cancellation.
OrderCancelReject	A venue sends this message to a trading party to confirm the rejection of an originally submitted order cancel request.
MarketDataIncrementalRefresh	A market data message sent to both the venues and trading parties from the Market Data Service.

Table 5.1: Shows the main FIX messages used in the simulation

The FIX messages in table 5.1 above implement the interactions described and discussed in the previous chapter (table 4.6). As an example, the FIX message for submitting a new order request (*NewOrderSingle*) is outlined as follows:

FIX Tag	Field Name	Comments	Example value
8	BeginString	Identifies beginning of new message and protocol version.	*FIX4.4*
9	BodyLength	Message length, in bytes.	*118*
35	MsgType	Defines message type	*D*
49	SenderCompID	Assigned value used to identify firm sending message.	*TradingParty1*
56	TargetCompID	Assigned value used to identify receiving firm.	*DPa*
52	SendingTime	Time of message transmission, in UTC (Coordinated Universal Time)	*20100512-20:30:31.254*
11	ClOrdID	Unique identifier for Order as assigned by trading party.	*A843.0.0*
55	Symbol	Ticker symbol of the security to be traded (the stock in this case)	*ABC*
54	Side	The side of the order, buy or sell (1=Buy, 2=Sell)	*2*
38	OrderQty	Quantity ordered, i.e. represents the number of shares.	*50000*
40	OrdType	Order type (1=Market order, 2=Limit order etc)	*2*
44	Price	The price willing to buy/sell if set as a limit order.	*145.0*
15	Currency	Identifies currency used for price.	*GBp*
10	CheckSum	Three byte, simple checksum	*123*

Table 5.2: Shows an example of a NewOrderSingle FIX message

The full FIX implementation for *NewOrderSingle* message can contain a far greater number of fields; however the example shown in table 5.2 contains the minimum required for this message. The actual data that is transmitted is very small, as intended by the FIX designers. The entire raw message would look as follows:

```
8=FIX4.4/9=118/35=D/49=TradingParty1/56=DPb/52=20100512.20:30:31.254/
11=A843.0.0/55=ABC/54=2/38=50000/40=2/44=145.0/15=GBP/10=123/
```

In the multi agent simulation system, this message will therefore be transmitted from a trading party to a dark pool venue -via Websphere MQ- with an order to sell 50,000 of stock ABC at 145.0p. A full list of fields used across the different messages in the simulation, along with a more detailed explanation can be found in appendix 3.

5.3 Architecture and Development

The implementation of the main entities utilising the messaging infrastructure detailed so far is covered here, including their architectural design and build. The functionality and end-to-end flow of the core processes are highlighted and discussed using tables, flow charts, code snippets, and class/sequence diagrams. The entire software development process resulted in the delivery of a relatively sophisticated trading and routing system, with over 150 Java classes spread across the main entities (i.e. trading venues, trading parties, data feeds) all communicating with each other over the messaging protocol as detailed earlier.

5.3.1 Execution Management and Smart Order Router

As discussed back in chapter 3, the order and execution management system takes care of the entire trade, from its entry to completion (with the exception of post trade activities such as settlement and clearing). It also integrates with the smart order router along with other components such as the probability calculator, used to aid in the routing decisions. A summary of the main functional features of our system (as covered in the previous chapters) is as follows: A) maintain the state of all orders entered; B) provide quantitative routing algorithms that can monitor the execution progress of its trades and general market movements, submitting or adjusting orders accordingly; C) ability to slice

orders and split them into smaller child lots, allowing routing to multiple venues simultaneously; D) build and maintain an analytical repository containing static and dynamic data elements useful for venue probability calculations and hence order routing. These are the main features, which also make up the bulk of the trading party setup within our multi agent simulation. Following is a high level architecture of the system, showing the main components developed.

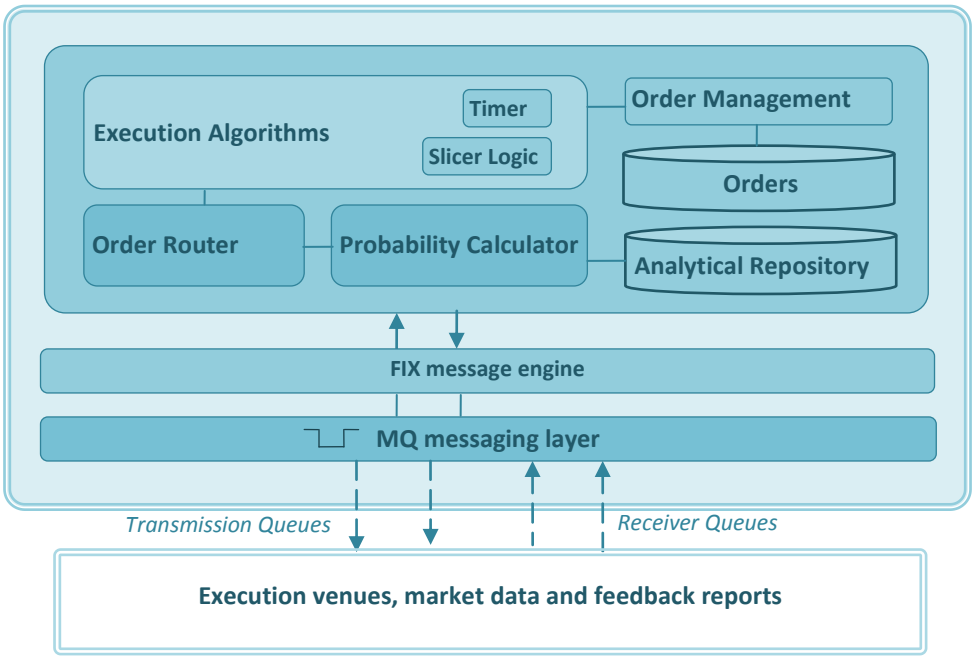


Fig 5c: Shows the main components of the EMS/SOR architecture used in the trading party

The execution algorithm together with the order router control how a parent order is handled and executed. It may be sliced across several "time buckets" then further split and routed to multiple venues. A timer is used to continuously scan all the orders in the system to ensure that they are in a competitive state – in terms of their pricing and routing requirements. Overall however, the architecture is primarily event-driven, i.e. actions mainly occur based on the occurrence of different events (to be discussed later). The probability calculator component, shown above in fig 5c, is called by the order router –in real time- when an order is about to be routed. It uses the analytical repository to draw upon relevant data for its venue ranking calculation. When a routing is about to be made, the *FIX engine*[13] converts the message from a Java object to a FIX message –similar to the sample message shown earlier (and vice versa).

[13] QuickFix/J engine is used as the FIX message parser for SOR. This is an open source fully featured Java based package, supporting the FIX messaging protocol [65].

Some of the methods and classes built –as part of the SOR system and its routing algorithms– can be seen in the class diagram below. Only the main ones are shown here, highlighting the key functions. A further list of classes as well as the overall package structure developed can be found in appendix 5.1.

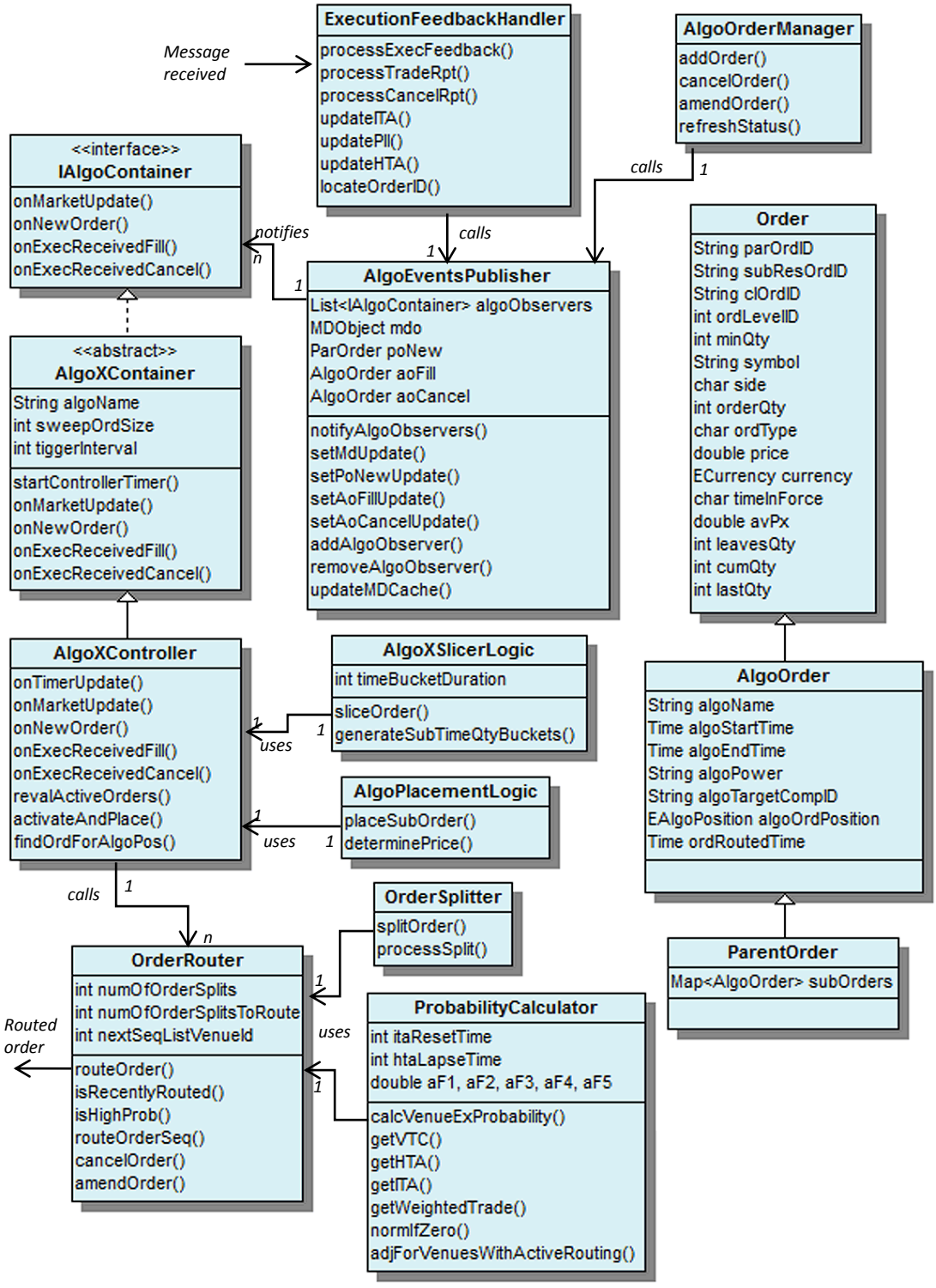


Fig 5d: Shows a class diagram of the main components of the SOR system

To help present an accurate picture of the system developed, some of the main classes in fig 5d is discussed further. Starting from the top, a message is received at the MQ receiving queue, which is then picked up by a message listener class. This class (which is not shown in the class diagram) validates and forwards it to the FIX engine to be converted into a binary Java object. Once this is complete, the *ExecutionFeedbackHandler* picks it up. This class is shown as the starting process in the class diagram above.

ExecutionFeedbackHandler Class

The primary objective of this class is to handle FIX feedback responses as it's received from others (e.g. venues), carrying out any required tasks accordingly. It processes execution reports and then feeds it back into the EMS/SOR system. For example, a report may indicate that one of its routed orders has successfully completed on a particular venue, and hence the *ExecutionFeedbackHandler* must ensure that both its routing algorithms and order management system is notified in a timely manner; this is so that the state of the order(s) can be maintained. It must also do the same by updating the probability indicators (i.e. HTV, ITA etc) in the analytical repository to reflect this latest state of play. Following is a summarised portion of the Java code from the main method of this class.

```
private void processTradeRpt(FixObject fixObj) {
        //Locate order and update order qty details
        AlgoOrder ao = locateOrderID(fixObj.getClOrdID_11());
        ao.setLeavesQty(fixObj.getLeavesQty_151());

        //Tigger event to notify order manager and routing algorithms
        AlgoEventsPublisher.getInstance().setAoFillUpdate(ao);

        //Check order status and update ITA, HTA and PII
        if (fixObj.getOrderStatus_39() == FILLED {
                updateITA(fixObj, 1);
        } else if (fixObj.getOrderStatus_39() == PARFILLED) {
                updateITA(fixObj, 0);
        }

        updateHTA(fixObj);
        updatePII(fixObj);

        //Register the executed order in posttrade
        ExecutedOrderFactory.addExOrder(ao, fixObj)
}
```

AlgoEventsPublisher Class

Any updates that occur regarding the orders, their state and market price movements are passed to this class, mainly by the *ExecutionFeedbackhandler*. The *AlgoEventsPublisher* maintains a list of classes that subscribe and listen to it for events, and hence it would notify them in turn of any new updates as and

when they occur. This ensures that the routing algorithms, order management system and other elements of the application are promptly informed (via callback) of any changes that occur, allowing them to take necessary actions. This part of the architecture follows an established software engineering design pattern, often referred to as the *observer pattern* [61].

AlgoXContainer and AlgoXController Classes

The algo container abstract class implements the generic *AlgoContainer* interface and is extended by each algorithm developed and used within the system. The container provides necessary interfaces for the algorithms to use, ensuring that it is able to receive market prices and act on different events based on the execution feedback received. One main algorithm (AlgoX) was developed for the SOR system, and is controlled by the *AlgoXController* class; this provides the core logic of how it behaves and directs the execution of an order. Extracts for this can be summarised as follows:

```
public void onNewOrder(ParOrder parOrder){
        //New parent order is added, process it/call slicing logic etc
}

public void onTimerUpdate(){
        //Loop through all orders in the order book, if order is resident and its
        start time is equal to the current simulated/real time then price and
        forward order to SOR Order Router to split and route
}

public void onExecReceivedFill(AlgoOrder ao){
        //When an order returns as executed, update its status to filled and look
        for further orders not routed yet. If found, activate and pass to Order
        Router as there's a high chance of fill due it ITA indicator currently set.
}

public void onExecReceivedCancel(AlgoOrder ao){
        //If order cancelation is confirmed, the algo is ready to route order
        elsewhere. Hence pass to Order Router to submit to the best venue.
}

protected void onMarketUpdate(MDObject mdo, AlgoOrder ao){
        //Market data update received; hence ensure that all orders currently being
        managed are still competitive; set order price within the market spread
        (exact details covered in evaluation chapter)
}
```

OrderRouter and OrderSplitter Classes

These classes manage the actual routing of an order to the venues. They may split the order and route across multiple venues. The exact strategies used will be discussed later in the evaluation chapter. From an implementation perspective however, the *OrderRouter* class has knowledge and access to the available venues and is aided by the *ProbabilityCalculator* to rank and determine

which ones to hit first. It also has the ability to cancel or amend an order based on the advice of the execution algorithm or trader.

ProbabilityCalculator Class

The calculations to determine the overall venue probability ranking is held in this class and is called directly by the *OrderRouter*. The *ProbabilityCalculator* class has direct access to the analytical repository which stores an up to date snapshot of the latest data that can be used in the routing formula.

```
public List<VenueProbability> calcVenueExProbability (AlgoOrder ao) {
        //Loop through each venue and get data from analytical repository based on
        stock used in order
        {
          nVTC = getVTC(exProbability.getVenueID());
          nHTA = getHTA(exProbability);
          nPII = exProbability.getPii();
          nITA = getITA(exProbability.getImmediateTA());

          //main probability calculation for each venue
          prob = ((nVTC*F1)/sumVTC)+((nHTA*F2)/sumHTA)+((nPII*F3)/sumPII)+nITA*F4;
          prob = adjustForVenuesWithActiveRouting(prob, exProbability);
          venueProb.setProbability(prob); venueProbList.add(venueProb);
        }
        //return venue list, sorted with probability rankings
        return venueProbList;
}
```

The *ProbabilityCalculator* is of course much larger than the very short snippet of code shown above, but this does give an idea of its overall function.

SOR's Analytical Repository

The repository serves the main purpose of providing data that can be used as part of the routing decision for any order that is about to be submitted. It has past trade knowledge, capturing and maintaining analytical data for both the selling and buying of every type of stock across each supported venue.

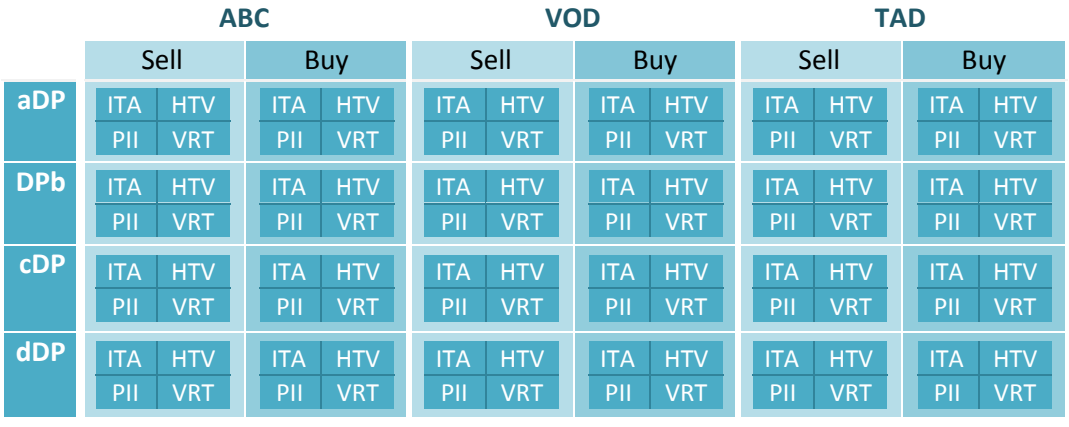


Fig 5e: Shows the main elements stored in the analytical repository for each stock traded

The layout in fig 5e above shows the level of detail the analytical repository holds for every stock (three are used for illustration). The dynamic data stored include ITA (Immediate Trading Activity), HTV (Historical Trading Volume), PII (Price Improvement Indicator) and VRT (Venue Routing Time), which is the last time a routing was made to a venue for a particular stock/type by SOR. The repository also holds reference data about each venue such as VTC (Venue Trading Cost), and in reality would include opening times and tradable instruments for each.

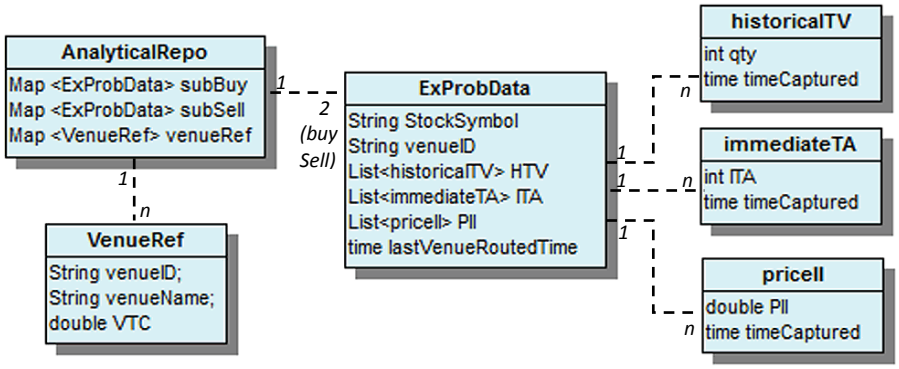


Fig 5f Shows the structure and implementation of the Analytical Repository developed for SOR

Order Management

The system handles all orders entered to it, managing its life cycle appropriately. It works closely with the execution algorithms and order router to ensure that the state of each order is kept up to date and valid. The following table lists the different states that an order can be in –as built into the system:

Order State	Description
NEW	Newly created parent order
INPROGRESS	Order is in resident state awaiting placement/routing
ROUTED	Order (or sub order), is active and routed to venue
ROUTED_PARFILLED	Order has just partially filled on a venue
ROUTED_FILLED	Order has just filled completely
CANCELLED_INPROGRESS	Order was cancelled from a venue, awaiting rerouting
COMPLETE	Order is fully executed, and is complete –i.e. closed

Table 5.3: Shows the states orders go through in the system

In-Memory Database

For data handling and storage the implementation used an in-memory Java object graph to store and maintain both the analytical repository and orders. Although this solution is not persistent, it is faster; in reality a full backend database approach would also utilised to ultimately save the data on disk –for replication, failsafe and backup/restore purposes.

5.3.2 Trading Parties

All 20 trading parties in the simulation utilised the implementation as detailed in 5.3.1 so far. However, only one party was coded to actually use the analytical repository and SOR capabilities during the tests. The others were configured to use various routing behaviours such as random routing, sequential routing, or favouring particular venues over others. The objective here was to simulate a busy trading period with different characteristics for testing the SOR concepts developed –to be discussed later in the evaluation chapter. Each trading party is an independent application able to make its own routing decisions in the simulation. The implementation therefore configures each with a separate messaging queue and is connected to all four dark pool venues. Each would be deployed separately on its own hardware, although in the simulation they were spread across two servers.

5.3.3 Dark Pool Venues

The implementation and structure of the dark pool venues follow a similar approach to the trading parties where each is independently run, communicating across separate queues. The high level architecture of each venue is as follows:

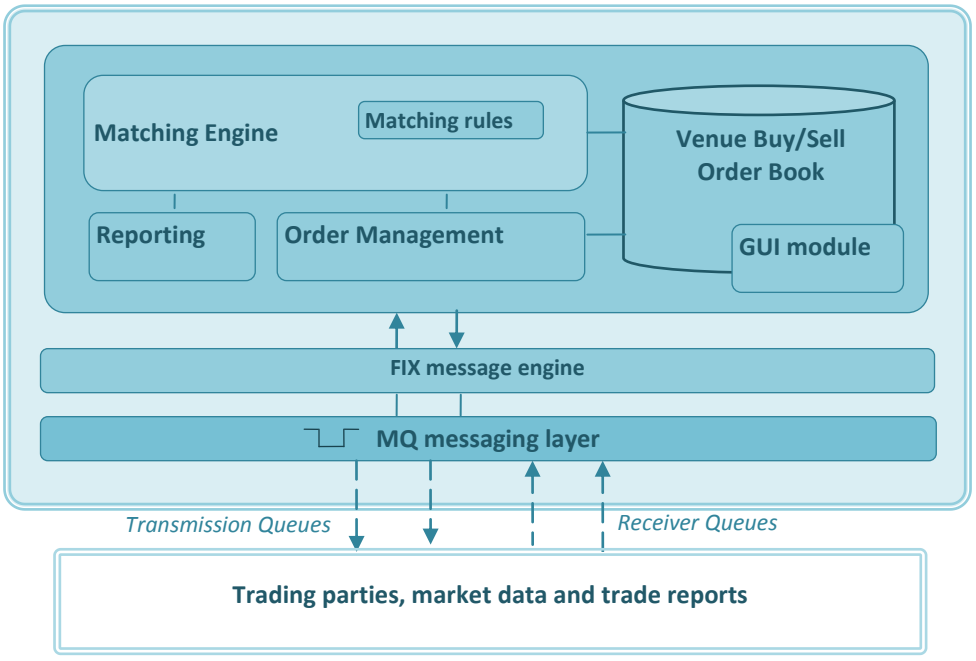
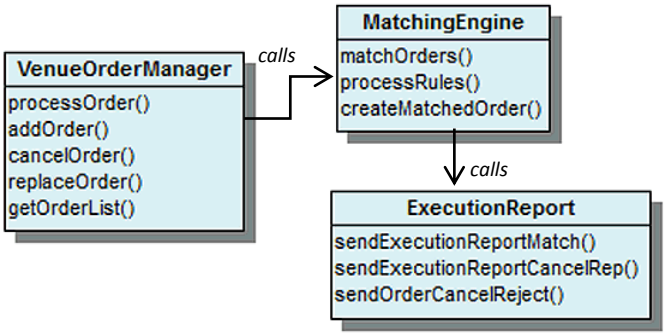


Fig 5g: Shows the main components and architecture used for the dark pool venues

An order is received and picked up by the venue -as a FIX message- which is then converted into a Java object by the *FIX engine*, similar to the trading party

implementation before. This is then processed by the *Order Management* component which checks order instructions in the message; it may be anything from a new order or an amendment/cancellation of an existing order. The venue's order book is updated based on this and the *Matching Engine*, which constantly scans and matches both sides of the order book (buy and sell), outputs any newly matched orders. Those matched orders are then said to be "executed", triggering an order match report and confirmation to be sent to both participating parties of the trade. The following class and sequence diagrams show some of the main methods and interactions between the order manager and matching engine.



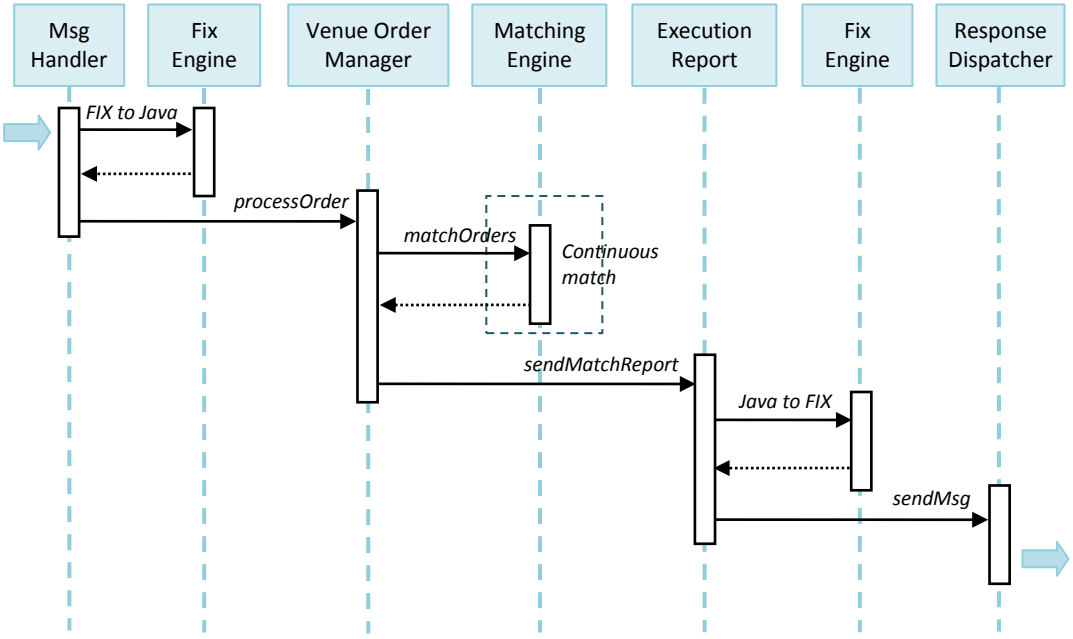


Fig 5h: Shows some of the main end-to-end interactions for an order entering the venue

Further classes and the package structure from the application can be found in appendix 5.2. The matching engine, together with the order book, serves as the main components of each venue; everything else is there to support this function.

Venue Matching Engine Classes

The logic for the matching engine is explained in detail back in chapter 4. From an implementation perspective, the engine consist of a number of classes, with the main being the core matching method. The following code snippet summarises this in sequence.

```
public synchronized List<MatchedOrder> matchOrders(Spring tickerSymbol){

        //Get current venue order book for required stock
        DPOrderBook ordBook = DPOrderBooks.get(tickerSymbol);

        //Create new ArrayList for matched orders to go in
        List<MatchedOrder> matchedOrders = new ArrayList<MatchedOrder>();

        //Return no match if either Buy or Sell lists have no orders in them
        if (ordBook.buy.size() == 0 || ordBook.sell.size() == 0){
                return matchedOrders;
        }

        //Sort Buy and Sell lists before matching (by price and time priority)
        Collections.sort(ordBook.buy,new OrderSort(asc));
        Collections.sort(ordBook.sell,new OrderSort(desc));

        //Begin matching, by comparing Sell and Buy order lists
        //Loop through each order in the Sell order list
        {
                //Loop through each order in the Buy order list and compare
                {
                 if (LimitOrMkt(oBuy.getPrice()) >= LimitOrMkt(oSell.getPrice())) {
                        //FOUND MATCH
                        //Determine matching rule (e.g. mid price)
                        processMatchRule(oBuy.getPrice(), oSell.getPrice())
                        //Update any remaining quantities in both orders
                        //Add and trigger match report
                        matchedOrders.add(createMatchedOrder(oBuy, oSell));
                } else {
                        //no need to continue through rest of buy orders for current
                        sell, hence move to next
                        break;
                }
        }

        //Matching complete, hence return list of matched orders (if any)
        return matchedOrders;
}
```

The matching engine first locates the correct order book based on each stock and creates a new array for matched orders to be added to. Before it proceeds further it checks if orders are present in both sides of the order book (buy and sell), if not, then simply returns no match for this iteration. Sorting of the book is then made, with the buy in ascending order and the sell in descending order, so that price priority can be established. Time priority is also considered where two or more orders share the same price. The matching then begins by comparing each sell order against the buy orders, with any match being processed based on the matching rules (as described in chapter 4). The total matched orders are then returned back for post trade processing and the sending out of execution reports.

Venue GUI Module

A graphical module was built (in Java/SWT) to view each of the dark pool venue's order books (in real time) as the simulation runs and progresses. A screenshot of this module is shown in fig 5i below.

Fig 5i: Shows the dark pool order books for stock ABC at a particular point in time

Whilst detailed results are always captured and analysed at the end of each simulation, the above module offers something more interesting. The view into the venues' order books in real time allows one to easily see and "experience" the general movements in liquidity (from venue to venue) as well as overall trading activity as it happens. This proved to be very useful in better understanding the different experiments conducted throughout the simulation.

5.3.4 Market Data Service

The market data service was developed and deployed as a standalone application. It establishes a link with all the trading venue and trading party queues in the simulation, enabling it to publish market data messages to them. The main components are the *MarketDataLoader* and the *MarketPublisher* classes. They essentially load historical market data pricing from file into memory and then "re-play it" throughout the simulation, publishing price updates. These updates are triggered as the timing of each price entry (from the data) matches the simulation time[14]. The application built for the market data service is relatively simple, and its class and package structure is included in appendix 6.2.

5.3.5 Managing Simulation Time

The tracking of timing in the simulation is an important factor to consider. This was discussed earlier in the chapter, where it was decided that continuous timing is to be used. Since multiple independent servers are deployed (hosting the many standalone trading parties and venues), timing must be synchronised together at the start of each test. Also, in order to have the ability to "fast forward" or scale the simulation, a uniform timer class entity, independent of the servers' clock time, must be deployed and used across all the different parties in the overall system[15].

To achieve this, all parties in the simulation are loaded into memory and lined up ready to run as soon as they receive a "start-timer" signal. This message is fired simultaneously across the messaging queues to all party applications. The message triggers each party's timer class (which runs on a separate thread), passing it configuration data such as the speed factor (to sets itself at) and the end time of the simulation. This is then referenced instead of the system clock, allowing timing to be defined independently. Of course, the big question here is how this can be synchronised, since it can be argued that the start time in each server will be intrinsically different, and that the time "gap" could even drift away further as the simulation progresses. This is understood and extensive tests were made to ensure that such a difference is kept to a minimum.

---

[14] A sample extract of the data file can be found in appendix 4.

[15] This of course will not be required for real time simulations (which our setup supports). In this case all trading entities can go by their own system clocks, which should anyhow be accurate enough, and is in fact a reflection of reality.

For example, at the start of the simulation all timers are triggered simultaneously with minimal latency involved since the time taken for the start-timer message to reach each party is negligible, considering the low CPU power needed to process the message as well as the close proximity of the servers to each other, connected via fast Ethernet cables. The other possible issue of drift between the different timers (as the simulation moves on) was also considered. Latency threshold checks were added in the code to compare each timer against their system clock to measure any significant changes. This was only found to be an issue with high CPU utilisation and server load, which was the case when the simulation speed was pushed over the 80-100x time factor mark –especially in some of the more demanding experiments. Hence, this was monitored and simulation was kept at a manageable speed.

5.3.6 Logging and Simulation Data Capture

In order to analyse and evaluate the results from experiments, they need to be captured and stored appropriately. A number of data capture components where built to keep track of all the changes and updates in the system as the simulation is run. The following table highlights the data captured:

Time series data	Captured by
Available buy/sell liquidity (order volume) over time	Venues
Total executed orders on each venue over time	Venues
SOR executed orders on each venue over time	SOR-enabled party
Executed price points on each venue over time	Venues & SOR party
SOR total remaining unexecuted order volume over time	SOR-enabled party
SOR venue probability rankings over time	SOR-enabled party
Total SOR order routings and cancellations over time	SOR-enabled party

Table 5.4: Shows the main time-series data captured during each simulation

The data is output from each venue and trading party into a simple CSV file, and is then imported into MS Excel for charting and further analysis. Each full run generates over 2,800 rows of numerical data (70,000 fields). Based on the hundreds of completed experiments (discussed in the next chapter), well over a million rows of data were analysed.

Chapter 6

Evaluation and Discussion

The system we are trying to model and simulate can be very complex and difficult to evaluate, especially as a single block. Therefore, to begin with, various elements will be evaluated in a controlled and simplified environment, progressing gradually towards more complicated areas. We do not expect to obtain a golden formula or "one-size-fit-all" approach for our SOR system; but we do expect to explore the strengths and limitations of the concepts developed and presented in this project.

6.1 Overview and Approach

6.1.1 Market Structure and Noise

In literature looking at market structure, e.g. A. Shleifer [34], the efficient markets hypothesis (EMH) as well as other later works around market dynamics and behavioural finance are frequently discussed; in these propositions, the concept of rational and irrational traders -or in other literature "informed" and "uninformed"- can exist alongside one another. These established concepts of rational/irrational behaviours when explaining market efficiency and price can also be borrowed and used in the area of order routing.

When trading parties' order routings are rational, we can for example say that they value each destination venue (dark pool) based on their execution ability and overall trading costs. Therefore their routing decisions will be based accordingly, and can hence be assumed to be "informed". When trading parties are irrational on the other hand, they may route trades randomly to any venue or venues, not necessarily based on any specific or combined structured logic. In this case, the trading parties' routing decisions can be said to be "uninformed". These general concepts are similar to the noise trader models, as formalised by F. Black in 1986 [35], and have long been used by many in the academic community to help understand various market behaviours [e.g. 55]. They are also frequently applied in market simulations [56] used to model different trading characteristics and outcomes which can be studied through empirical analysis of the simulated

results. In order to evaluate and compare the different smart order routing elements, similar concepts in the form of informed and uninformed parties will be applied in the routing decisions across the trading population in our simulation.

6.1.2 Model Validation

While it is difficult to produce accurate models or results that relate exactly to the real world, one can nevertheless demonstrate and recreate -to some degree- some of the perceived attributes and characteristics. A substantial amount of literature can be found around validation of simulation models, including R. Sargent, K. Troitzsch, K. Carley [51, 52 and 53]. The common approaches discussed around validity include model checks against real data, comparison with other models, historical data validation, as well as internal validity where several runs of the simulation are made to ensure that the stochastic variability in the model is reasonable.

In the previous chapters, we discussed the entities within the simulation model and their design and characteristics in relation to reality. The aim was to ensure that the simulation design, including the trading parties, venues and overall characteristics were both comparable and realistic. In this chapter it is important to understand each of the tests as well as the results obtained and be able to relate them to real world behaviours. This area will be looked at and discussed throughout the different experiments and tests that follow.

6.2 Simple Baseline Run

To begin with, a set of simple trading simulations are run and experimented with. These will include the participation of trading parties as discussed in chapter 4, each trading against each other on the four dark pool venues. The following characteristics and settings will be used:

Stock price and venue trading cost

For now all trading parties will submit their trades as market orders (as opposed to limit orders) and the market price will be assumed constant throughout the simulation. Hence the price should not affect any of the trading or results. Similarly, all four venues are considered equal in their ability to execute trades and at the exact same cost; so the venue trading cost is not taken into consideration at this moment.

Simulation time and stock volume

A simulation length of 12 hours will be modelled (~1½ days worth), during which a busy period of trading takes place where a total of approximately 1 million shares of stock ABC will be executed. While it can be argued that the number of shares and length of simulation are both arbitrarily selected here, the volume of trades against time are realistic in comparison to reality (Appendix 1). They can of course be adjusted further to create and experiment with a wide range of other scenarios. It must be noted however that the aim here is not to model a second-by-second exact replica of a real trading snapshot, but more to capture a series of trading activity that may take place within a realistic timeframe.

Uninformed routing

All trading parties in this instance of the simulation will route their orders to any of the four venues randomly. Here the concept (as explained earlier) of uninformed and irrational decision making in order routing will take shape across all trading that occurs in the simulation. Where orders do not execute on a given venue, a proportion of the trading parties will cancel and reroute their orders (again at random) to any of the four venues. The "randomness" in routing is largely reproducible in each run, as it uses a pre-initialised set of random values. In later tests, this will be replaced so that completely different randomly generated routes are created for each run.

Distribution of order volume across time

The focus of the research is primarily around the routing of orders, i.e. ones where the decision of what -and generally when- to trade has already been made; hence the determination and timing of when a trading party chooses to trade its initial "parent" orders shall be predefined and scripted. This ensures that the test environment is better controlled and understood. However, in saying that, the subsequent decisions that occur once each initial "parent" order has been released depends on the routing decisions in each run and is not scripted. The following (in fig 6.2a) shows the distribution of buy and sell order volume which will be preset and triggered during the simulation across the dark pool venues.

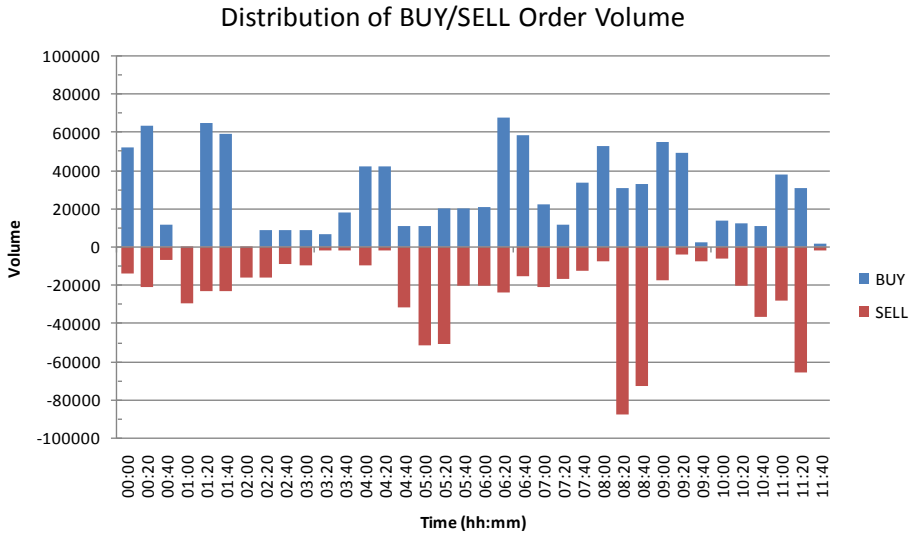


Fig 6.2a: Shows the distribution of order volume preset to be triggered across the dark pools

The total volume of the two sides, buy and sell, are assumed roughly the same (1 million each). They are distributed unevenly across time which creates more realistic and varying supply and demand stresses during the length of the simulation. In later tests, the distribution will be re-adjusted to simulate more sell or buy volume present in the venues.

Running the simulation

Now that the properties and conditions of the simulation have been discussed, we can begin with the first runs (accelerated/scaled real time) with the results recorded. This was completed multiple times and the data was output to file, which was then subsequently imported and analysed in MS Excel. The following graph in fig 6b shows the executed volume of a typical run across the four dark pools based on the controlled settings and parameters discussed so far.

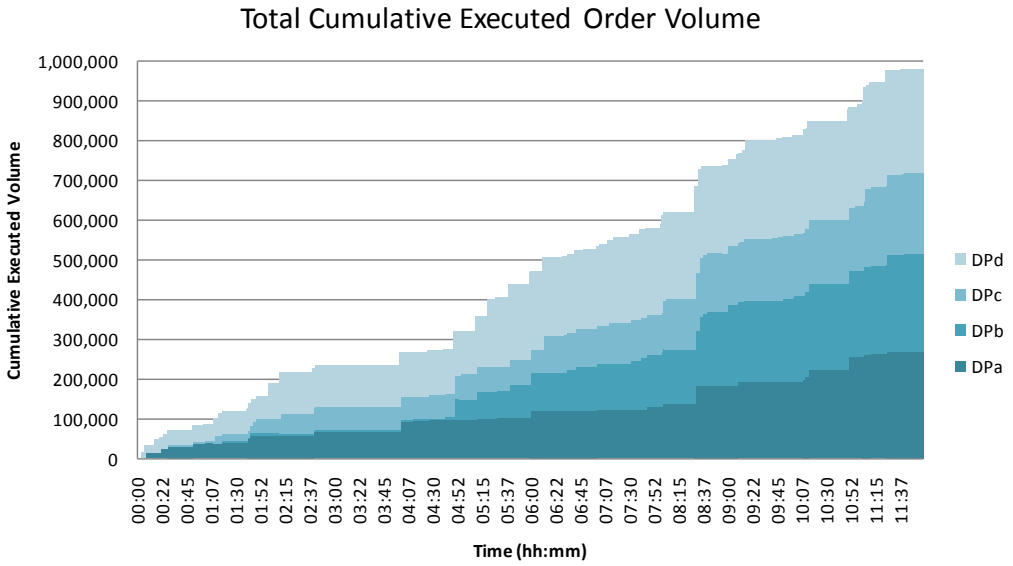


Fig 6.2b: Shows the total executed order volume across the dark pools over time

As time in the simulation progresses, an increase in the executed order volume is expected across the dark pools since the trading parties are submitting more and more buy and sell orders; this is based on the general buy/sell volume distribution in fig 6a. One observation that is clear from the results is that although the level of volume increase is gradual, it is not evenly linear. This depends on the amount of tradable buy/sell orders at any moment of time, which in our model, as with reality, varies. For example, the results in fig 6.2b above show a sudden jump in the executed volume at around 8:30. This can be attributed to a high number of both buy and sell orders in the market around this time, as is shown in fig 6.2a. This is further enforced when examining a snapshot of remaining buy/sell liquidity in the venues over time (fig 6.2c). This view chases liquidity as it moves from one venue to the other providing a picture of when and where unexecuted order volume is present in the dark pools.

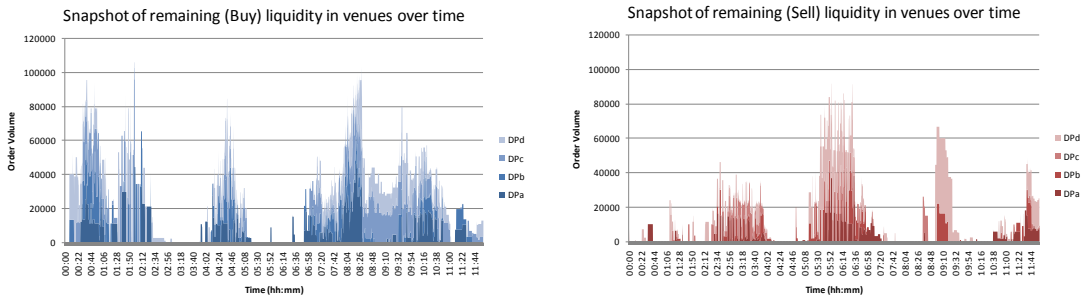


Fig 6.2c: Shows a snapshot of remaining unexecuted buy/sell liquidity over time. Note: only remaining liquidity present for at least 10 seconds is recorded; executed liquidity is not included.

The random routing decisions made by all trading parties also reflect well in the results, where the volume of orders executed across each dark pool varies, as shown in fig 6.2d below; however, ultimately the resultant overall total for each venue is similar, since in this experiment there is no difference between any of them and no "intelligent" routing is being made by any of the trading parties. Essentially, orders are being routed and rerouted at random. This does not necessarily relate to reality, where each venue has its own varying market share. However, in this test we are assuming that they are all equal for now.

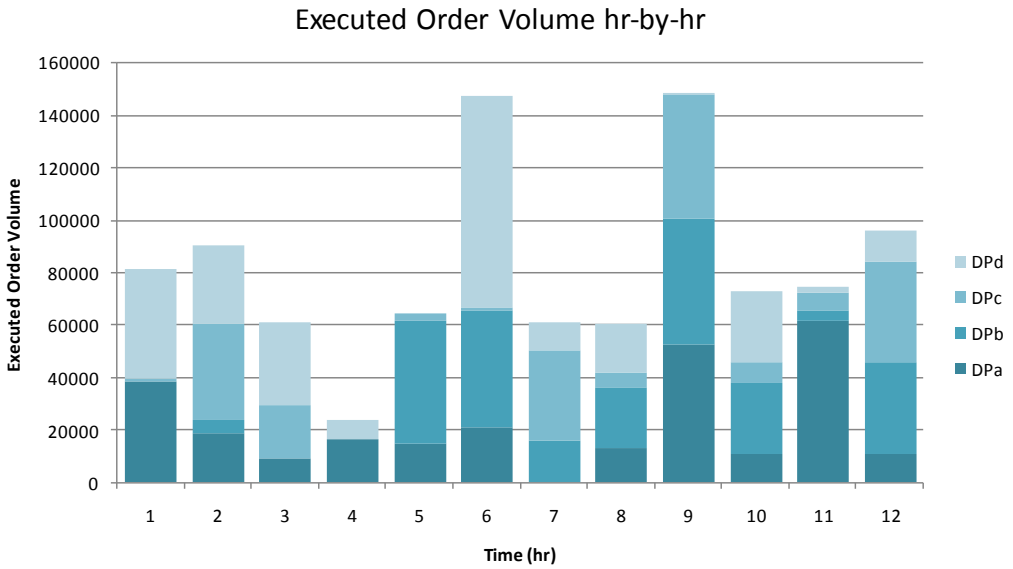


Fig 6.2d: Shows the aggregated executed order volume across the venues hour by hour

It is interesting to note here that since the stock price and venue trading costs are assumed the same throughout the simulation, the volume of executed orders follows closely with the supply and demand as laid out in fig 6.2a. Both buyers and sellers are willing to trade immediately and there are no price constraints, hence as soon as two opposite orders are present in a trading venue, the execution occurs. However, since there is no structured logic to route orders to the dark pool venues, the overall completion of orders may not necessarily be the best in terms of either cost/value or timely execution.

6.3 Empirical Experiments I

Following on from the first set of simulations, a number of different experiments will be looked at and studied. Each will examine a particular aspect of the smart order router within various trading environments. The experiments are grouped into several sections, with each progressing from basic structured tests to more complicated areas, combining multiple elements together. Again the aim here is to demonstrate and have a better understanding of some of the concepts discussed in our SOR system.

A proportion of the sell volume from the supply and demand distribution used in the simulation (shown in fig 6a earlier) will be allocated for the upcoming experiments in this section; namely 200k sell shares (out of the one million) will be solely handled by a trading party using the SOR system; i.e. as an informed trading party. This is a significant proportion, which will help highlight the effects of using different routing strategies in the simulation. This proportion will also be adjusted for some of the tests, examining cases where order size and volume may impact routing results. The individual elements and functionality of the SOR implementation will be gradually introduced and utilised in each test, and hence behaviours can be studied and examined further.

6.3.1 Routing Based on Venue Trading Cost and Order Splitting

The following tests aim to demonstrate the effects of basing the SOR decisions primarily on the venues' trading cost (VTC), as outlined in section 3.2.3. Some of the simulation parameters will be borrowed from the above test in section 6.2, where the market stock price is assumed constant and the one million shares traded will generally follow the buy/sell volume distribution in fig 6a; however we now allocate 200k of the sell volume to uniquely use VTC as its routing decision and is submitted at the start of the trading simulation. Each dark pool venue will have its own unique trading cost fee, as outlined in section 3.2.3 (where DPa: 0.3 bps, DPb: 0.7, DPc: 0.5 and DPd 0.3).

All other trading parties not using SOR's VTC as their routing decision will continue to use random routing, similar to the previous simulation. However, to simplify the model, no rerouting will occur; i.e. once orders are submitted by their respective trading parties -to any of the venues- they stay there until they are fully executed. Assuming that the 200k sell shares are executed as one single

large order, the entire routing would be submitted to the lowest (i.e. cheapest) dark pool venue. In this case, either of DPa and DPd have the lowest basis points per share. The router in this test will submit the order to DPa at the start of the trading simulation. Given that all other factors are the same, the router will have, in theory, achieved the lowest venue trading cost for the order.

The simulation was run and the results are shown in fig 6.3a below. Since this instance of the simulation is heavily controlled and almost reproducible, subsequent repeated runs of the same test showed very similar results.

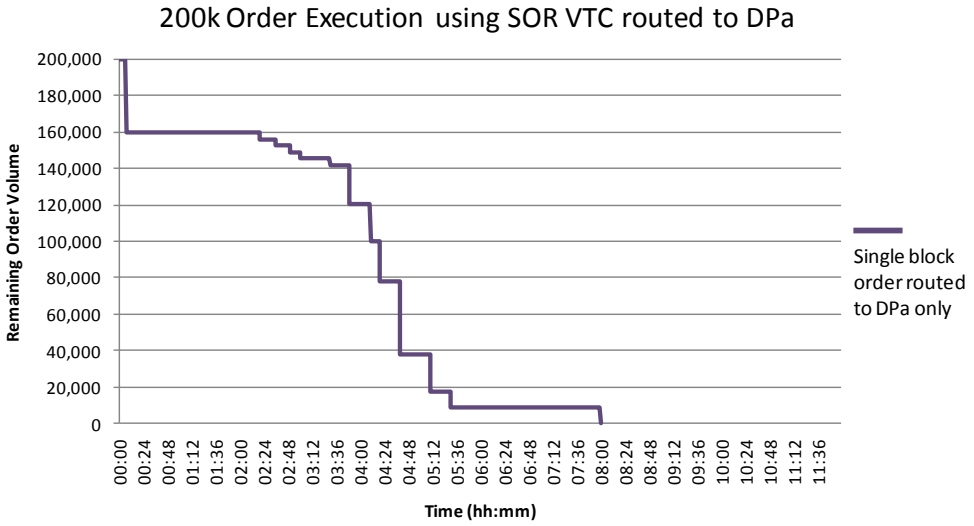


Fig 6.3a: Shows the execution of the 200k order over time - routed based on VTC to DPa

With the SOR using the lowest venue trading cost (VTC) and hence submitting the order (all in one go) to DPa, the entire order completes at approx 8:00 hrs, based on the results above. For the first half of the simulation, the executed order volume across all the dark pool venues (fig 6.3b) show a heavy bias towards DPa as the 200k order, placed entirely in that venue, is being executed. Since we deliberately allowed no rerouting of orders between venues to occur in this test, it further added stresses to the supply and demand on each of the venues.

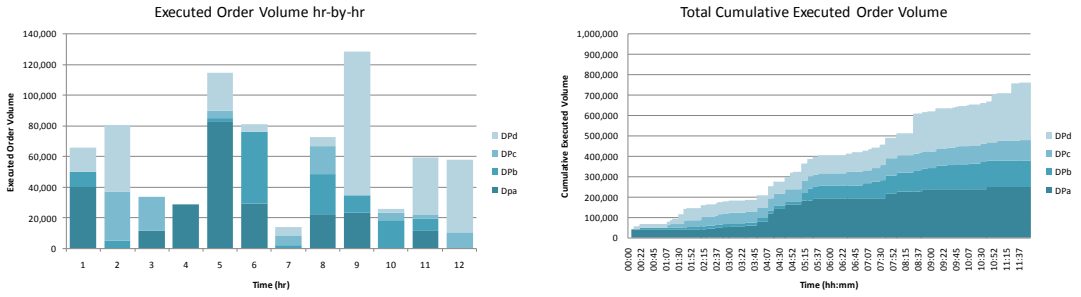


Fig 6.3b: Shows the execution volume over time for all venues when the 200k order is sent to DPa

If the venue trading cost is taken as the SOR's main decision point in this test, then routing to multiple venues, i.e. to DPa and DPd who both have similar low trading costs of 0.3bps, at the same time, should logically increase the chances and speed of execution. The order should therefore complete quicker while maintaining the same venue trading cost fee. Since the dark pool venues do not provide any data on current buy/sell levels in their order books, the router will have to decide if splitting the order up is more effective. Therefore, an interesting test that can be progressed here is to examine whether or not splitting the order –say in half- and routing to both of these venues simultaneously will improve the overall execution probability and speed of the order.

With the exact same conditions as before, the simulation is repeated again, but with the SOR system splitting the 200k order now into two equal 100k lots and routing each to DPa and DPd respectively at the same time. The results are shown in fig 6.3c below and compared against the previous run.

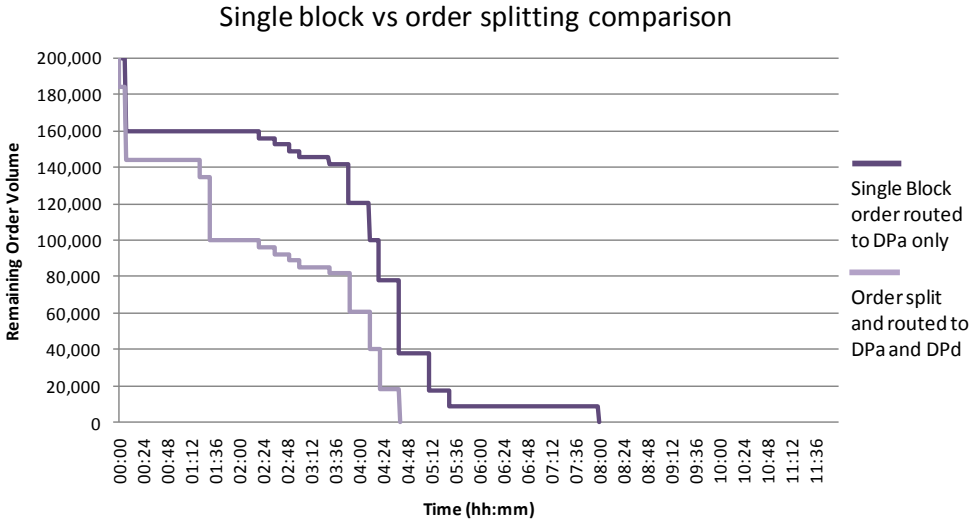


Fig 6.3c: Shows the execution over time of 200k single block vs splitting in two 100k orders

From the comparison of the results above, there does seem to be an improvement in execution time when splitting the order and routing it to multiple venues. Based on the simulation conditions and parameters used, the quicker time in order completion can be attributed to being able to capture more liquidity from both DPa and DPb simultaneously. However, as noted earlier, the above tests were made under tightly controlled conditions; all the other participating trading parties in the simulation used a fixed set of reproducible randomly generated routing decisions to decide which venues to direct their orders to over time. While

the results obtained are clear, they may not always reflect the general case as they are only really valid based on the fixed data used in them. In order to better validate the results and simulation, each test will now be repeated multiple times with each run using an entirely different and unique set of randomly generated routings. These routings impact the liquidity available in each venue, as trading parties decide on which dark pools to submit their orders to, while retaining the general characteristics of the supply and demand volume distribution outlined in fig6.2a (presented earlier). The multiple repeated tests are again compared against each other, with new results shown in fig 6.3d below.

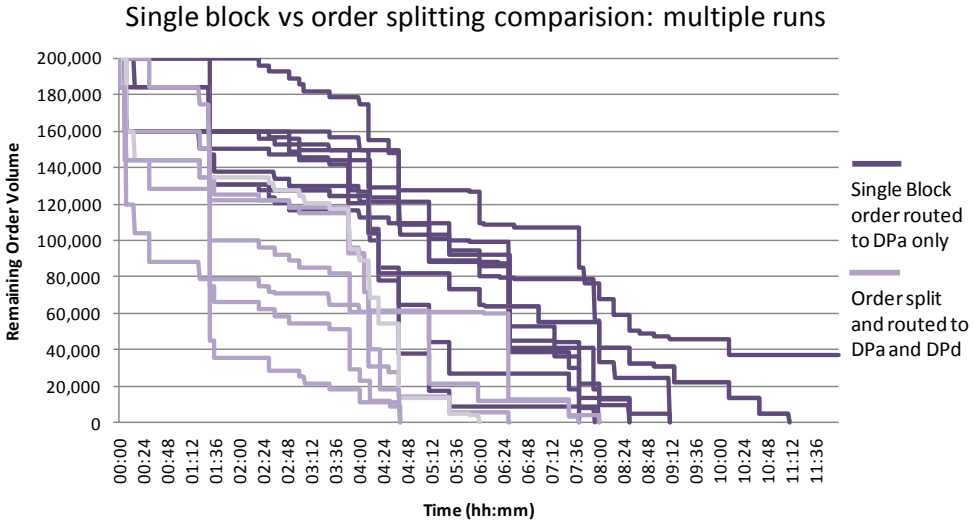


Fig 6.3d: Shows the execution over time for multiple repeated tests: single block vs order split

It is clearer from the above tests that one can generally expect an improvement in the execution of an order when split and routed across multiple venues. When the order was submitted as a single lot to DPa only, there were some instances when it didn't fully complete within the timeframe; this may imply that not enough opposite side (buy) liquidity was being directed to DPa venue in some of the simulation runs. Also, there were instances where a single block order routed to DPa executed faster than when order splitting was deployed. This type of variation reflects somewhat to reality, where all venues do not generally execute the same level of orders across any timeline. The overall observation however is that, given the conditions set in our simulation, the execution of very large orders are more likely to complete faster when split across multiple venues; in this case, the two lowest VTC were used (DPa and DPd).

6.3.2 Order Size Considerations in Order Splitting

Having demonstrated possible improvements in order execution when splitting and routing to multiple venues, especially to those with similar trading costs, one can look at some of the other factors which may impact the results. Market price is of course a significant one, where the longer the execution of an order takes, the greater the risk that the market may move, especially negatively, and hence impact the final overall execution price. However, this will be examined later when market price is factored into the tests. For now, market price is still assumed constant, to simplify and control the model.

A factor that will however be looked at now is order size; while the broad claims discussed in the previous section on VTC and order splitting held successfully based on the simulation conditions set, it is by no means applicable in all situations. The question here is whether splitting up the order and routing to multiple venues will always yield better results, or are there cases where it is in fact better to submit the entire order as one single block. To explore this in the simulation, multiple tests are made with different order sizes; from small, medium to large orders.

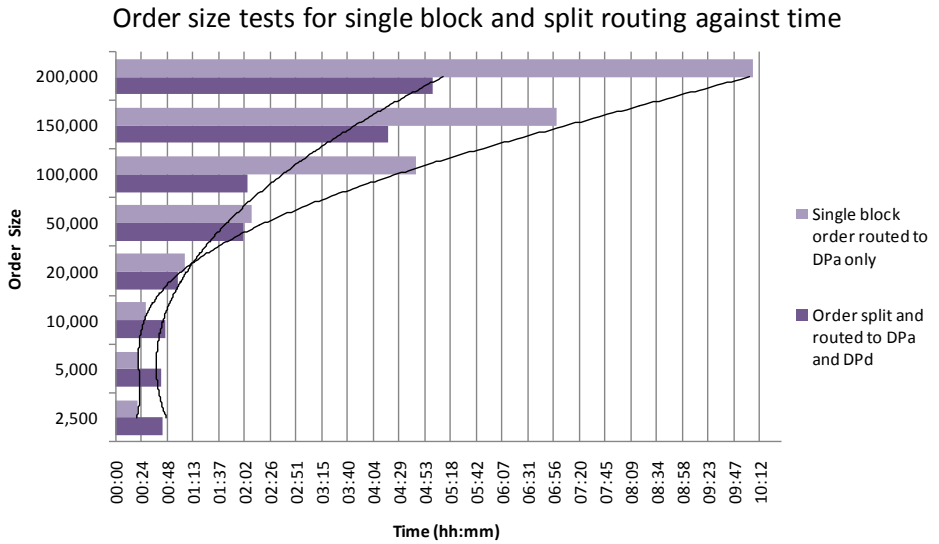


Fig 6.3e: Shows the effects of varying order size against time for single and split order routing

So continuing along the same controlled scenario as before and using the same two dark pools, tests are repeated many times across different order size increments for both single block and split order mode, and the average execution time is recorded. The results are shown in fig 6.3e above and a sample of the raw

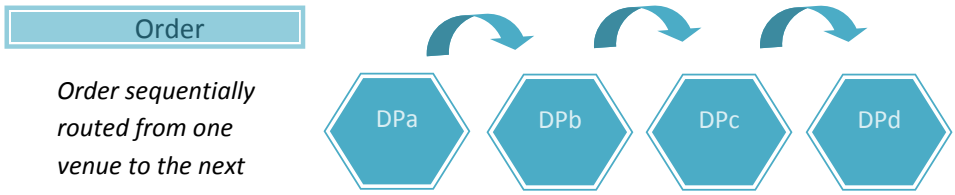
data from the tests can be found in appendix 6.1. What is interesting and clear from the results above is that not all cases of order splitting were effective. In fact, for the smaller order sizes, single block orders (routed to one venue) executed quicker than when they were split across two different destinations. An explanation to this may relate to the timing of incoming orders and their sizes as they enter each of the venues. If a sizable buy order is present in DPa for example, then splitting a small sell order into multiple segments may not always be the best option, especially if DPa could have matched and executed the entire order all in one go. Conversely, when a very large order is routed altogether to a single venue, then it may take longer to execute, in which case splitting it across other venues might be better.

Based on the characteristics of the simulation used, including the fact that no cancellations or re-routings of submitted orders can occur, the overall executed order size average across the venues was found to be 5,000 per trade. The results in fig 6.3f show that routed order sizes of around 15,000 and above were found to benefit better from the two-way order splitting, whereas those lower did not. This however is in no way an established quantitative figure and indeed varies significantly depending on the trading characteristics used. This may include the venue trading rules where some for example exert minimum order size rules while others do not. Dark pools such as the Asian BlocSec [54] do not impose any minimum order size requirements, accepting small size trades even in the low hundreds; while others such as BIDS, Level ATS and Turquoise [24], especially restricted by regulatory requirements, dictate much higher sized thresholds.

Nevertheless, what this test demonstrates is the possible increase in execution probability when an optimal order splitting strategy is determined and maintained. This would factor in the size of the traded order and may be unique for the stock or even the venues it is routed to. Such strategy must be regularly monitored and analysed for its effectiveness. More on order splitting will follow later in the chapter.

6.3.3 Order Cancel and Sequential Re-routing Across Venues

This experiment looks at the effects of cancelling and re-routing of an order from one dark pool to the other and in different market conditions. The previous tests essentially just "parked" the order in one or two venues without any re-routing involved. If we take a large order and simply submit it to the first venue, then after a set interval we cancel and re-route it to the next venue in line (and so on), what results would this give and how does this compare to simply leaving it in one venue? Would we be able to seek out more hidden liquidity?



As with the previous tests, and in order to demonstrate and evaluate this area, similar controlled simulation settings will be used. The market price will remain constant, hence assuming all trades are market orders. Also, with the exception of the SOR trading party being evaluated, no rerouting of orders submitted by any of the other parties will occur; again random "uninformed" routing of orders will be used by all the others. This strict and controlled test may not reflect reality; however it serves the purpose of demonstrating elements of the system which can later be built upon and revaluated with more complex and realistic scenarios.

The simulation was prepared and run repeatedly across a range of different parameters. The buy/sell distribution used in the simulation is again borrowed from the previous tests, and the 200k sell order is submitted by the router to the first dark pool venue; then after a predefined time interval, a cancel message is sent, cancelling what remains of the order and then resubmitting to the next venue (i.e. DPa, then DPb and so on). A continuous cycle of sequential cancelling and rerouting of the order occurs until the entire trade is executed. A selection of time intervals was used and, as with the previous experiments, each type of test was repeated many times with only the average taken. The results are shown in fig 6.3f bellow, along with the distribution of buy/sell used in the simulation. The

same order submitted (but to a single venue) with no cancel/rerouting is also included for comparison. The raw results can be found in appendix 6.2.

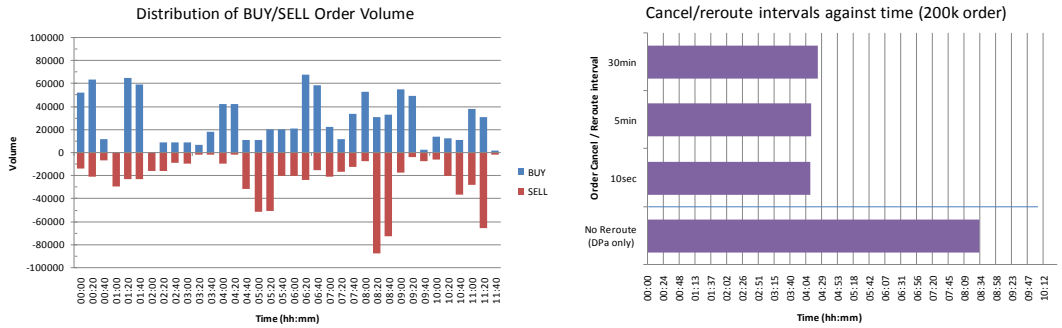


Fig 6.3f: Shows distribution set (left), and results of order completion times using rerouting

What is clear from the above results is that given the controlled conditions used, the continuous movement of the order from one destination to the other does appear to have significantly improved the execution time and captured more liquidity across the venues. This is especially clear when comparing against the single order submission to DPa only, as highlighted in the results above. However, this may only be true for the specific test conducted and against the generally defined buy/sell order distribution used. To study broader effects, the distribution is adjusted so that the other trading parties are gradually shifting more sell orders –in direct competition- early on in the simulation than before.

Fig 6.3g: Shows varied distribution sets alongside effects on order completion using rerouting

The distribution of buy orders in these tests was kept the same; only the sell side was adjusted. The 200k block sell order submitted to the single venue (DPa) continued to execute and complete at similar times of around 8.30hrs throughout the tests (fig 6.3f/g) and didn't seem to be affected. However, what is interesting is when continuous order cancellation/rerouting (across venues) was used, the changes in sell volume distribution significantly affected the order completion times. The improvements recorded in the initial tests from fig 6.3f were no longer being returned, and in fact far less favourable execution times were later being experienced in fig 6.3g. As mentioned earlier, market prices in these tests were constant and all trading parties traded market orders only, hence price should never be a factor in the results. Similarly trading parties (other than the SOR party) routed orders randomly (uninformed) and hence liquidity might be found in any of the dark pools with no particular reason for it to concentrate in one venue over the other. So why does the order rerouting strategy in this experiment yield good results for some market conditions and not others? –as demonstrated.

The answer may be found in the time priority rules of the trading venues, as discussed in chapter 2 and 4 (section 4.2.1). Whilst it may be a good idea to constantly reroute an order from one dark pool to the other in search for liquidity, an important element that is affected is the possible loss of order priority in each of the venues' internal order books.

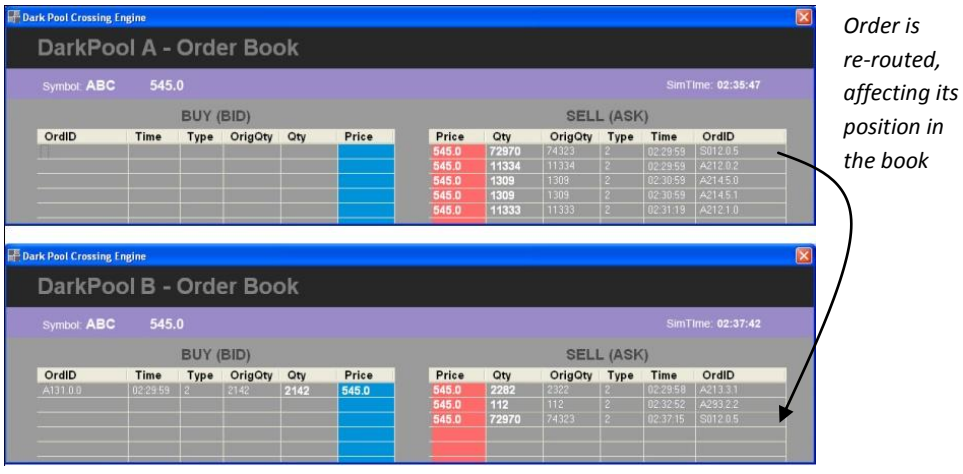


Fig 6.3h: Shows a snapshot of DPa/DPb's order books illustrating positioning in entered orders

When an order is cancelled and then resubmitted to another (or same) venue it loses its positioning and is entered at the end of the order book (unless its price is more favourable than the others). This is illustrated in fig 6.3h above, where the

resubmission of our sell order (remaining 72,910 shares) was entered at the end of DPb's order book after it was cancelled from DPa. This may not be a great issue if the number of other sellers competing were small, and especially if the opposite buy orders was significant; as there will be more of a chance for our order to be at the top position in each of its rerouting intervals. This is not the case however when a significant number of sell orders exist and are all competing for less buyers. Excessive rerouting in this scenario reduces the chances of our order ever being tradable as often enough, and keeps it low in the positioning index of the venues' order books. This results in unfavourable execution times as shown in the tests, and is often discussed and acknowledged in literature e.g. [7]. The behaviours demonstrated in the simulation model reflect well in reality where order priority plays a significant role in execution and order completion times. As mentioned earlier, almost all exchanges and dark pools today follow the principal of price/time priority when executing orders in their books [29].

6.4 Empirical Experiments II

The experiments covered so far touched on a number of basic concepts that may demonstrate, as well as impact, some of the SOR elements presented. The simulation and test conditions used were very much controlled and structured, allowing the results to be better understood. This theme will continue here for the second section of the experiments, where again the scenarios and assumptions are stated, along with the objectives and discussions for each case. These upcoming tests demonstrate more of the SOR elements, combined together and analysed using a range of different simulation settings.

Learning Behaviours with Multiple Orders

So far, all the tests discussed in the first section (experiments I) largely dealt with a single "parent" order throughout the simulation timeframe, which was either submitted as one lot or split across different venues. Whilst this was sufficient for the tests covered, it is not enough to demonstrate some of the other elements as well as the "learning" behaviours that are expected from our SOR system; this will require multiple orders to be submitted through it (at different times) in order to analyse the routing decisions made across the entire simulation for the SOR-enabled trading party. Continuing from the 200k sell order submitted in the previous tests, four smaller "parent" orders are used this time and entered periodically across the simulation; namely 50k order lots every three hours. Other mixed order sizes and timings could have also been considered.

Time	Order ID	Stock	Qty	Price
00:00:00	AO101	ABC	50,000	Market Order
03:00:00	AO102	ABC	50,000	Market Order
06:00:00	AO103	ABC	50,000	Market Order
09:00:00	AO104	ABC	50,000	Market Order

Table 6.4: Details of "parent" orders to be handled by the SOR trading party

The above table lists the exact timings of the orders along with their details. It can be assumed that these orders may have been generated by a separate automated trading algorithm (e.g. TWAP or VWAP as discussed in section 3.5) or submitted directly on behalf of other traders etc. Indeed the breakdown of the orders may vary in size and timing, but the overall idea is the same; i.e. various orders are entered into the SOR across different times, where it must then immediately decide how to handle/route each appropriately (as it receives them).

Volume Distribution in Venues Across Time

Similar buy/sell volume distribution (as with the previous experiments) from fig 6.2a was used. However, what is different here is that the volume will be distributed differently across the dark pools. A proportion of the trading party population was adjusted so that they vary the volume of (opposite) buy orders they route by favouring a particular venue more often than others. The trading parties in the previous tests routed orders completely randomly across all four venues and as a result, it was often the case that each had largely similar volume running through them by the end of the simulation. This was fine for the experiments completed, however they are no longer adequate for the upcoming tests which require further control on the volume dynamics -to analyse some of the more complicated SOR behaviours.

A third of the trading parties or agents (with the exception of the SOR-enabled party) were adjusted so that they are twice more likely to favour routing order volume to DPc (as an example) than others. For all other cases with trading parties, random routing, as well as rerouting[16] was used. The adjustment was made so that an almost guaranteed general increase from ~25% to ~35% of the total traded volume was executed on DPc.

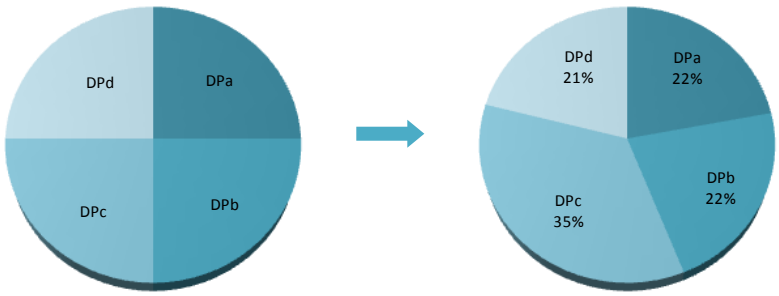


Fig 6.4a: Shows the before/after of overall volume distribution across venues to be used

The above distribution is only an approximate and the actual total traded volume executed in each venue varies from one simulation run to the next; however on average the new adjustments should result in similar levels as above. It can be possible to relate this new variation in volume distribution to the real world,

[16] Active rerouting of unfilled orders was used by half of the trading parties and occurs after a random interval set < 5min. The other half of the population route and leave their orders in a venue until it's fully executed; whilst a simple arrangement, this concept allows the modelling of both active and passive orders in the simulation.

where dark pools are not equal in size; those such as Sigma X, Bats and CrossFinder for example execute far greater volume than some of their smaller counterparts [57].

6.4.1 Equal Order Splitting Strategy Across All Venues

To begin with, applying the conditions and structure discussed so far, the SOR system is configured so that each "parent" order entered is evenly split into smaller orders and routed across the dark pools. In other words, using the scenario outlined earlier, each of the four main 50k sell orders would be split in four equal pieces of 12,500 and routed to DPa, DPb, DPc and DPd respectively. Hence the logic (similar to experiment 6.31) assumes no real difference between the venues in terms of volume or cost and therefore sets out to maximise its chances of capturing liquidity by simply posting to all venues equally. The simulation was run and a sample of the results returned is shown in fig 6.4b below. The full breakdown of each order/sub order executed, along with time and venue, can be found in appendix 6.3.

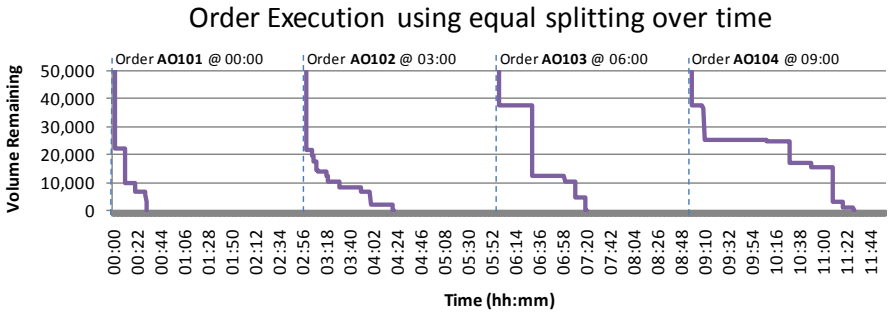


Fig 6.4b: Shows the execution times for each parent order based on equal splitting

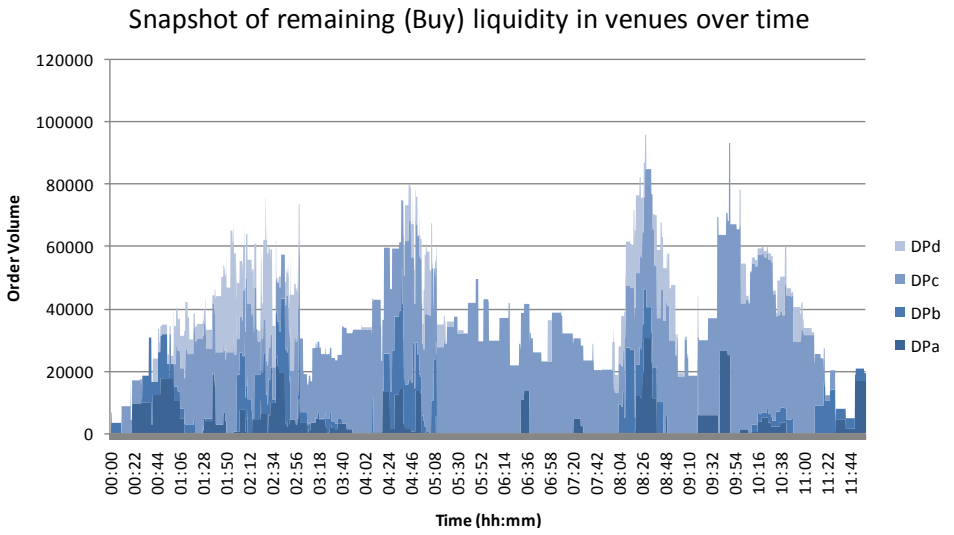


Fig 6.4c: Shows a snapshot of remaining unexecuted buy liquidity in venues over time

Since the size of liquidity across the venues differs, an equal order splitting strategy will not be the most optimal method. In fig 6.4c, the remaining buy volume (taken in 10 sec snapshots) across the venues show a higher presence of liquidity in DPc than the others -as expected based on our scenario. The results in fig 6.4d below show the actual executed volume (on each venue) by all the trading parties in comparison to our SOR-enabled party. It clearly shows that DPc generally executed more orders than the rest; however the SOR system did not recognise this since it always channelled its orders evenly.

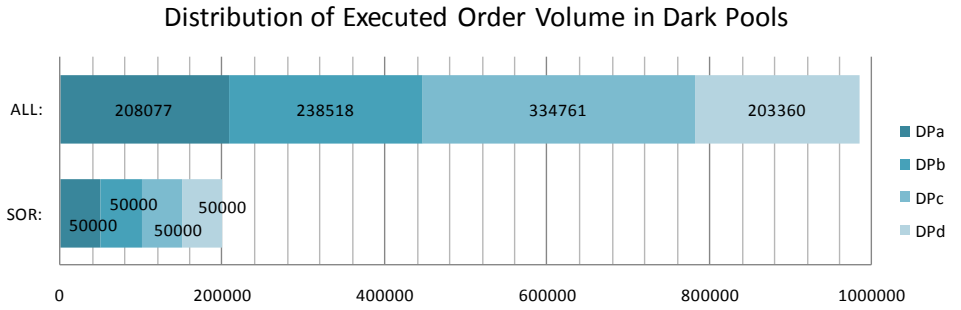


Fig 6.4d: Shows the distribution of executed orders between all and SOR party volume

Hence a dynamic allocation for each parent order across the venues by our SOR system should yield better results. However in this example it does not necessarily always mean more orders to DPc, since other factors such as venue trading cost may come in the picture and could have an effect on the overall routing decisions. Incidentally in this test, the average venue trading fee costs incurred across the orders executed by SOR was 0.45 bps[17].

6.4.2 Historical Trading Volume with VTC

In chapter 2 and 3 the concept of using past execution experience to aid in order routing was discussed. The historical trading volume (HTV) was presented in the form of a simple mean/average based formula; its aim is to calculate some kind of probability or preference metric for SOR to use as part of its routing decisions. In this experiment, the HTV will be used and factored along with the venue trading cost (VTC). Specifically, the HTV here represents the historical activity that the SOR system experiences (based on its own routings) and not from any external sources; this was referred to as rHTV (router HTV) in chapter 3. The tests in this

[17] This is easily calculated since equal amounts of volume were executed across the venues and we know the bps costs for each (DPa 0.3, DPb 0.7, DPc 0.5 and DPd 0.3).

instance will also apply no time-weighted decay factor[18] to the HTV value, although later tests will look at this area in further detail. Again, similar scenario and simulation settings as the previous test will be used; only this time rather than splitting each parent order equally and routing across the venues at once, it will follow the "split and reserve" strategy (discussed in chapter 3 -section 3.4) and use the HTV and VTC to determine the routing decisions. The following formula (from chapter 3) will serve as the main execution probability calculator for this experiment and will hence rank the four dark pool venues accordingly before each routing is made.

$$rank_n = \frac{(VTC_n)^{-1}.\ 0.5}{\sum_{i=1}^{k}(VTC_i)^{-1}} + \frac{(rHTV_n).\ 0.5}{\sum_{i=1}^{k}(rHTV_i)}$$

The SOR splitting mechanism splits the parent order into many smaller orders and submits them based on the -just derived- rank for each venue. The splitting used here ensures that a larger lot is allocated for the higher ranking venues than those lower in the list. The splitting strategy implemented in our SOR system is an adjustable set of pre-configured parameters which for this evaluation is set as follows:

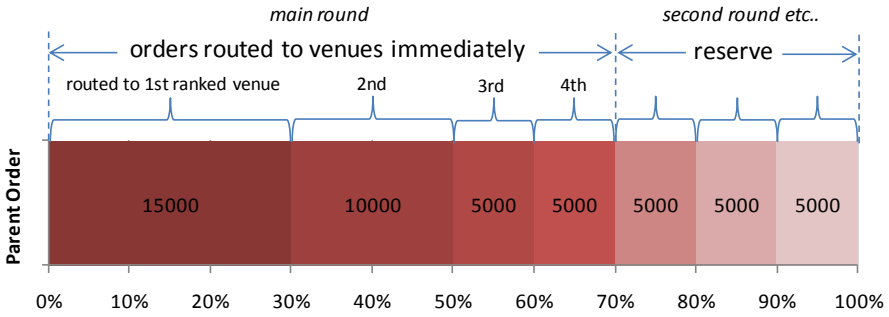


Fig 6.4e Shows the splitting/allocation of each parent 50k sell order by the SOR system

The parent order is split so that 70% is routed immediately across the four venues and 30% is reserved on standby. The reserve orders are "lined up" and gradually routed based on the feedback received from those that were immediately dispatched earlier. Again, there is no absolute best or optimal splitting size that can realistically be derived and applied across all orders routed by SOR. Each order type, volume and market condition dictates different requirements; for example, if a trader wishes to only route their order to a

[18] HTV time-weighted decay factor was discussed at length in chapter 3 (section 3.2.1.2)

particular market(s), then the above splitting strategy would need to be readjusted accordingly[19]. However, generally, the splitting mechanism used here allocates a larger part of the order for immediate routing and a smaller part for routing based on a revised calculation of venue rankings (after execution feedback is received from one or more initial routings).

To recap, this experiment will evaluate the use of HTV with VTC in calculating probability/ranking using the above splitting strategy. The simulation was run, again many times, and a typical result from the tests is shown below in fig 6.4f.

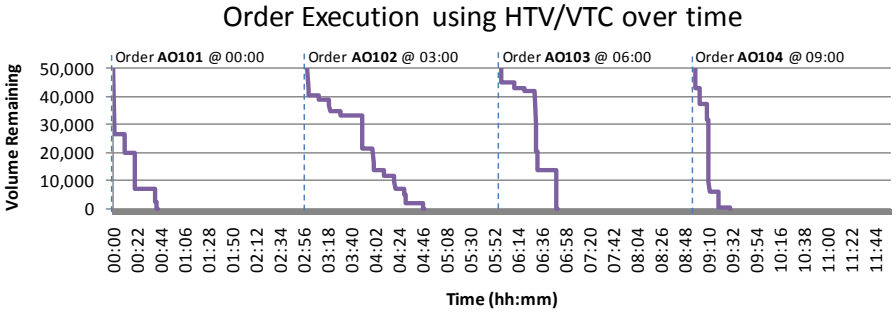


Fig 6.4f: Shows the execution times for each parent order based on HTV/VTC/splitting

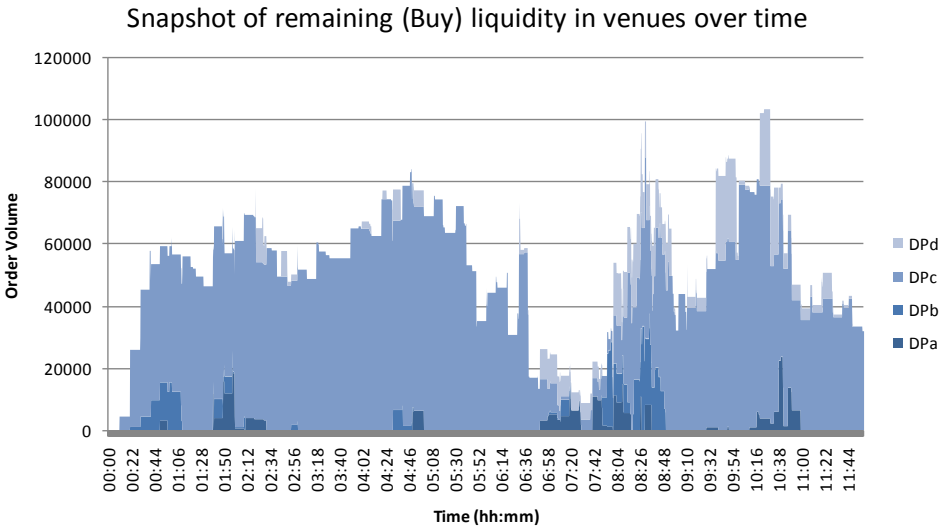


Fig 6.4g: Shows a snapshot of remaining unexecuted buy liquidity in venues over time

The results seem to indicate that the router captured less of the volume in DPc, as evident from fig 6.4g above where a high amount of liquidity in that venue remained throughout the simulation. This suggests that the HTV indicator has

[19] The splitting strategy can be adjusted automatically based on a set of rules, some of which can be customised by the trader to instruct the router to favour certain venues or split the order in a certain way, possibly without holding a reserve etc. But for the purposes of our tests, the splitting mechanism outlined in fig 6.4e will be used.

not made the "right" impact to the overall routing decision; since if it did, then liquidity in DPc would have gradually been further engaged, as we know it contains the highest volume as intended by the scenario. Instead it seems that the VTC indicator has had a much greater impact forcing more routings to go towards the lower cost venues (DPa and DPd). This can be seen more clearly in fig 6.4f below, showing a snapshot (in percentage) of the calculated venue rankings used by SOR for each order routing across the simulation timeline.

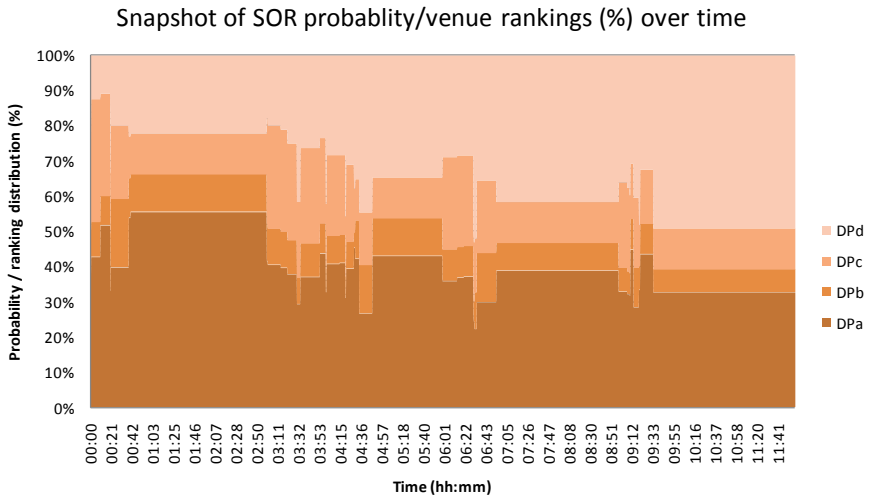


Fig 6.4f: Shows a snapshot of the calculated rankings across all venues over time

The majority of the routing decisions from the start of the simulation to the end favoured DPa and DPd respectively, even though both HTV and VTC in the probability formula outlined earlier had equal weighting factors (0.5 each). However, since the VTC indicator is static (i.e. its ranking of venues is always the same), it has a constant steady influence on the routing. The HTV on the other hand is dynamic and its rank values change depending on past executions. The interesting thing here is that, the HTV indicator can actually magnify the past routing decisions by causing more routings to direct towards the same venues. This is especially the case here where the two venues (DPa and DPd) quickly established their position early on in the simulation and then subsequently continued to dominate as they were helped further by HTV. This is more likely to happen when there is a good coverage of liquidity across the venues, and hence more choice for the router.

Another key point that can explain the results is how reserve orders are routed after an order from the initial main round fully executes on a particular venue. Since the ITA indicator (discussed in chapter 3) is not used in this test, no real

extra significance or weighting is given to those venues that have just (literally) executed and completed an order. Hence each of the reserve order routings that subsequently follow will rely on the same formula, which again, has VTC heavily represented at all times without treating very recent activity differently. The end result is that the orders took longer to execute than what they might have, given the amount of liquidity present (especially in DPc). To balance this, less weighting should be placed on VTC, as well as to consider the use of the ITA indicator to capture liquidity more effectively. Of course, if the intention is to keep the trading costs low at the expense of time, then the above test has certainly achieved this, with an average overall cost of just 0.34bps across all the orders routed. However, the savings in lower fees are not necessarily worthwhile if execution time is important. Certainly the view from many in industry is that (in those cases) access fees or trading costs are often not the top priority for SOR systems [58, 26], and would hence rank low in the routing criteria.

6.4.3 Combining HTV, ITA and VTC in Routing

Following on from the previous test, the experiment is progressed further by combining both historical and immediate trading activity with venue trading cost. The ITA flag was discussed back in chapter 3 as a way to temporarily increase the weighting (and therefore rank) of a venue where execution of an order -routed to it previously- just completes. It works on the assumption that if an order was just found to have fully executed on a particular venue, then there is a chance that further similar liquidity exists on the same destination at that -or near- moment in time[20]. The ITA is combined into the existing formula (from the earlier experiment) and the adjustment factors for each element are reapplied as follows:

$$rank_n = \frac{(VTC_n)^{-1}.\ 0.25}{\sum_{i=1}^{k}(VTC_i)^{-1}} + \frac{(rHTV_n).\ 0.35}{\sum_{i=1}^{k}(rHTV_i)} + ITA_n.0.4$$

The VTC is given a lesser role this time, with an increase to the HTV role. The ITA flag wields the most weighting on the routing decision, but as mentioned earlier, its effects are short-lived and hence returns back to zero after a short period of time. Apart from these changes to the formula above, the experiment

[20] This may not always be the case of course, since time priority rules within the venues' order books may come into play, as demonstrated in the tests earlier in section 6.3.3.

uses the exact same test conditions as the previous simulation in 6.4.2. In other words, the same four-50k orders, timing, splitting strategy and buy/sell volume distribution is used. The simulation was run and the following are typical results obtained: (full breakdown and further tests can be found in appendix 6.4 and 6.5)

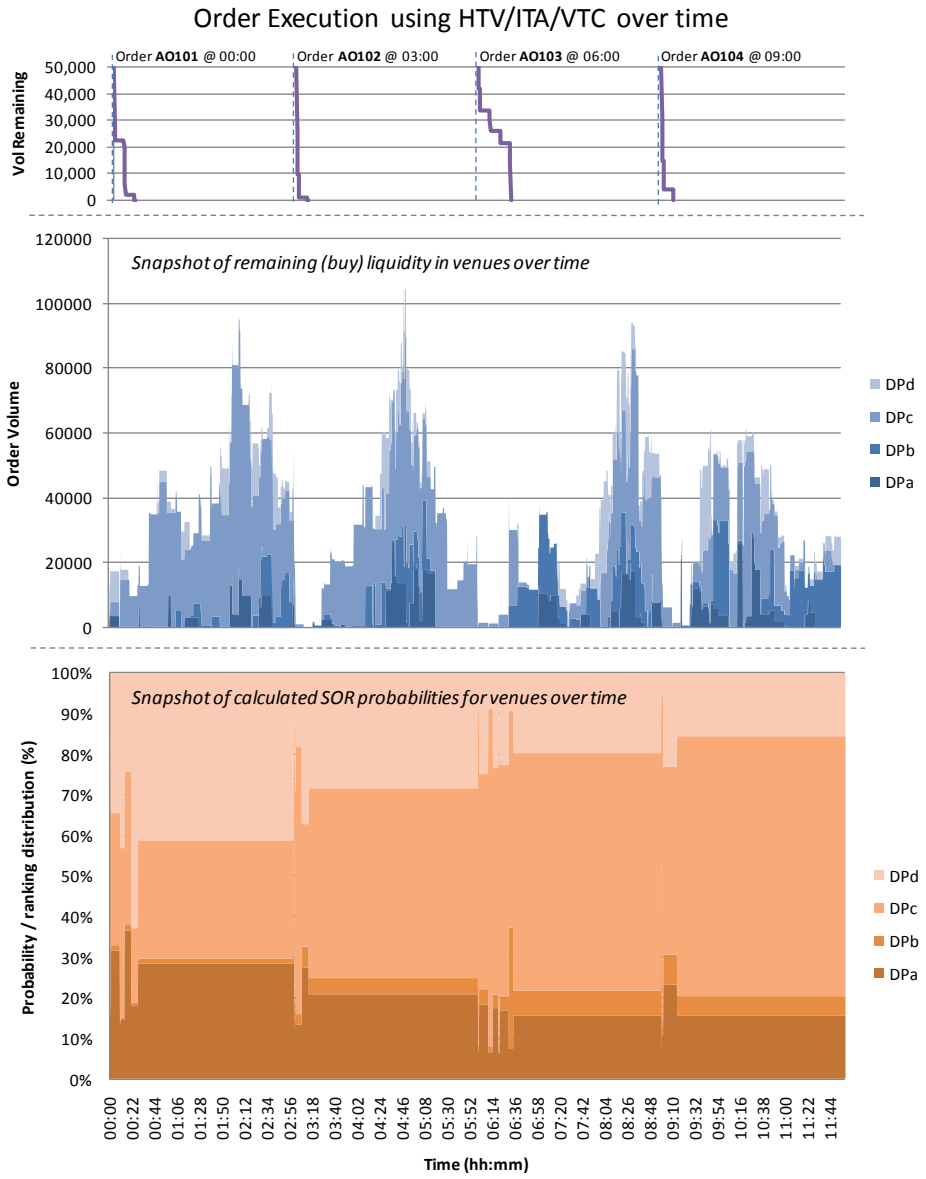


Fig 6.4g: Shows the results when using the discussed HTV, ITA and VTC in routing over time

It is clear from the results above that the orders generally executed quicker than in the previous experiments. The router managed to capture more of the liquidity available in venues such as DPc, which was known to have a higher share of volume than the rest (by design, as discussed earlier). At the beginning of the simulation, and for the initial order, the probability rankings fared well for DPa and DPd, since they offer the lowest trading costs; but as the simulation

progressed, and with more volume being successfully executed at certain venues than others, the SOR system gradually adjusted the venue rankings and probably values accordingly. This is evident for DPc where the probability snapshot (bottom graph in fig 6.4g above) shows a clear preference for this venue emerging over time.

Looking at some examples from the results, we can see that at 3:00 and 9:00, as the sell orders (AO102 and AO104) were about to be routed, there was a significant amount of buy volume building up in DPc than in any other venue. This can be seen in the middle graph above. The SOR system was able to correctly route the bulk of these orders to the best venues at the right stage (which included DPc as its top ranking preference) and therefore was able to execute the orders quicker. The ITA flag also played its role throughout the simulation, directing reserve (or cancelled/rerouted) orders to venues that just executed its initial orders. The effect of ITA can be seen in the form of short "spikes" in the probability snapshot graph, giving a much higher probability for those venues for a short period of time (before resetting back to zero). It is interesting to note that most of the ITA indicator flags across the timeline were set on DPc, again demonstrating the intelligent element of the SOR system in adapting and combining both immediate real time (ITA) and historical data (HTV) to channel order flow effectively[21]. The overall result is that the router was able to dynamically allocate and execute more orders on venues that were more likely to fill and complete them (i.e. DPc), as shown in fig 6.4h below, when compared to the total volume executed by all partied across all venues.

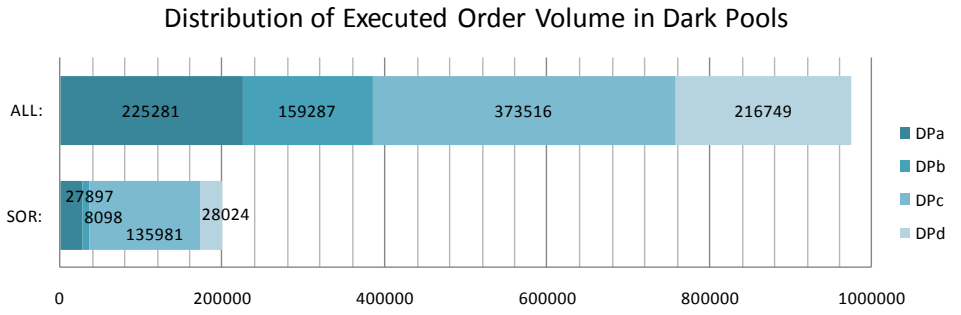


Fig 6.4h: Shows the distribution of executed orders between all and SOR party volume

[21] The Automated Trader published a roundtable industry report around the key areas in successful liquidity sensing in SOR; one such area highlighted is the router's ability in determining how to "marry real time learning with historical experience" [26].

Of course, further tweaks and tests should be made (again on various market conditions) to continue the evaluation of the routing formula deployed. However, what this experiment demonstrates is how the SOR system might utilise and combine different indicators to aid in its routing decision. It is interesting to note that with the VTC indicator having a lower weighting factor, its impact on the routing calculation was reduced as the simulation progressed. Some effects are still seen however; firstly, the initial routings by SOR did favour the lower cost venues (especially DPd), suggesting VTC's influence, which may be explained by the fact that in the early stages of the simulation there was little trading activity experience for SOR to use. Secondly, overall, the second and third highest number of order volume routed by SOR (after DPc) was DPa and DPd, which may be attributed to their low trading costs and hence preferred over DPb. All in all, it is clear that in this experiment, when making its routing decisions the SOR system regarded executed volume and liquidity higher than venue trading costs.

6.4.4 Effects of Decay Factor Applied in HTV Routings

The HTV indicator can be used to assist the router to make current or future routing decisions based on how well orders executed in the past (for each venue). In the previous experiment, this indicator was used in a market setting where high volumes of liquidity was purposely concentrated in a particular venue, and hence through time, the tests showed the routers ability to adjust and capitalise on this better (aided by HTV). However, as with any indicator or calculation that relies heavily on historical data, the past is no guarantee of the future. Therefore, flexibility is required to ensure that the HTV can adapt to different market situations. The time-weighted decay factor (discussed in section 3.2.1.2) was introduced into the HTV calculation to provide for this flexibility need.

To demonstrate this in a simple experiment, the previous tests from section 6.4.3 were repeated again; only this time, the build-up of volume concentration is now shifted mid-way through the simulation from one venue (DPc) to another (DPb). This ensures that on average more buy orders are routed specifically to DPc for the first half of the simulation, and then more over to DPb for the second half.[22]

[22] This can be achieved in many ways, and in this case, was done by simply adjusting the random routing logic in a third of the trading party population to increase the chances of hitting a specific venue over the rest at different stages of the simulation.

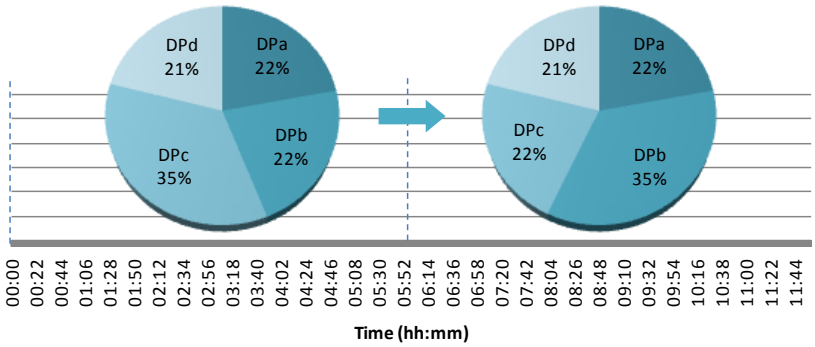


Fig 6.4i: Shows the changes in average (buy) volume distribution from DPc to DPb over time

The simulation was repeated several times, both before and after the decay factor was applied to the HTV indicator. Typical results from the tests are shown in fig 6.4j below. The intended shift in the amount of buy liquidity present in both DPc and DPb can be seen clearly. For the first half of the test DPc is the dominant venue; however this then changes in the second half where DPb takes over.

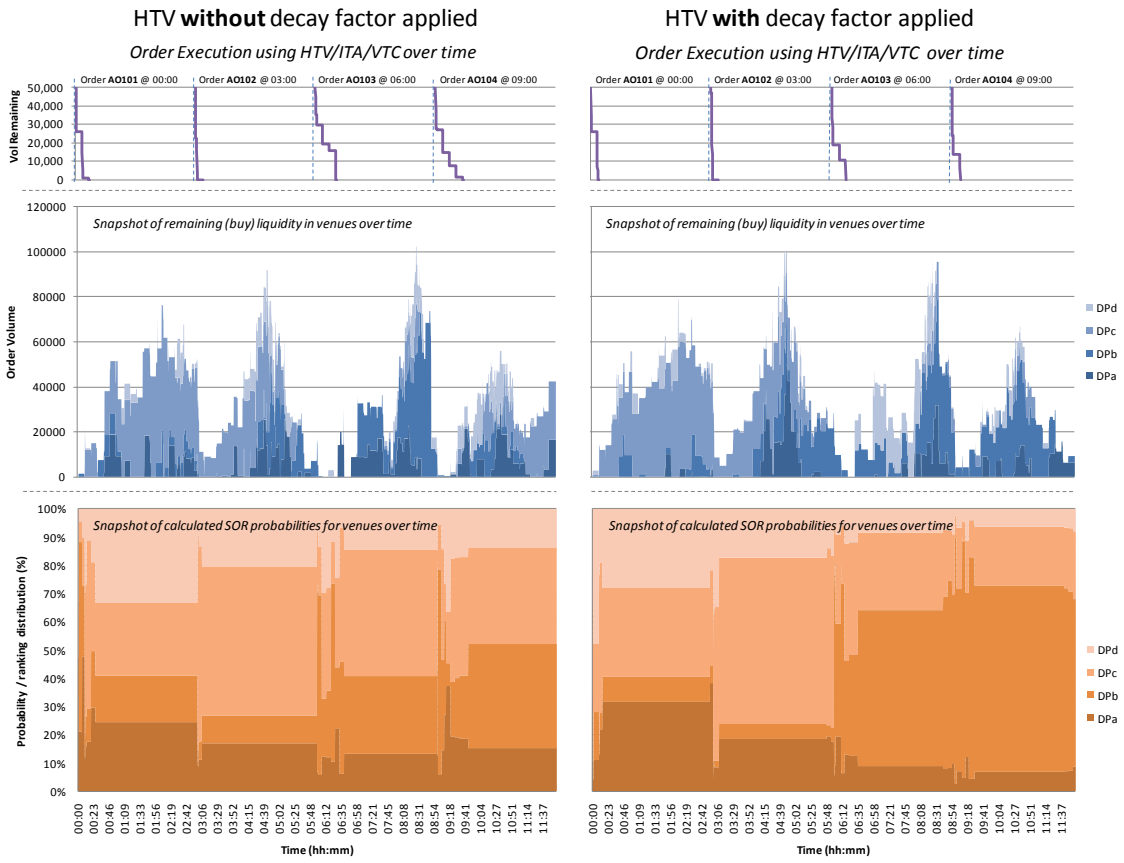


Fig 6.4j Shows the before/after effects of applying the decay factor on HTV in the simulation

The simulation starts normally with the HTV indicator assisting the router by gradually providing historical trading data; this feeds into the overall probability for each venue, and DPc slowly comes out as the clear favourite (0-6hrs from the

bottom graphs in results above). However, as the midpoint of the simulation is reached, the shift in volume distribution starts to occur with more buy liquidity building up in DPb (as intended). The router at this stage begins to experience more and more successful executions from this venue than in DPc. When the decay factor is not applied (bottom-left graph from results above) the HTV indicator is "sluggish" to adapt and its value -which to this point was heavily swayed towards DPc- takes longer to adjust. Hence it continues to significantly influence the overall probability towards DPc, even though this venue may no longer be the best. When the decay factor is applied (bottom-right graph from results), the "older" historical data is damped down -by applying a decay factor- and hence reduces its weighting against HTV data that is more recent. This allows the HTV indicator to be more flexible and sensitive to fluctuations in volume across the venues. This can be seen in the results where the router quickly adjusts its probability rankings in favour of DPb in the second half of the simulation. This in turn improves the routing decisions which can mean quicker more efficient order executions.

The following table details the decay factor parameters configured and deployed in the test above, taken from section 3.2.1.2 but with slight tweaks to suit the simulation timeline used here.

Phase (p)	Time passed	Decay factor
1	Less than 4 hrs	1 (i.e. no decay)
2	Over 4hrs and less than 8hrs	0.5
3	Over 8hrs	0.3

By varying the levels of decay factor values, adjustments can be made to improve the effectiveness of the HTV indicator when used in probability calculations (across various market settings). Further experiments to investigate different scenarios can be looked at; however what this test demonstrates –through a multi agent simulation- is the importance of ensuring that any data utilised in SOR calculations must be adaptable enough to cope with the unpredictable changes in the market.

6.5 Empirical Experiments III

In all the experiments so far, it was assumed that market prices were being published at a constant price level throughout the simulation, with all trades routed and executed as market orders; hence pricing did not really play any role in the results. This was required in order to control the simulation and limit the number of complex variables, allowing particular areas to be looked at and tested easier. So the introduction of the market price element into the simulation will offer a much more dynamic and realistic –but arguably more complex- environment for the SOR system to operate in. The following sections will explore a number of tests utilising this new dimension in the simulation.

Pricing Data for Tradable Stock

As discussed back in chapter 4, the prices published in the simulation (by the Market Data Service) for stock ABC is based on real tick-by-tick data from a comparable stock, which in this case, KGF (Kingfisher plc) prices from early March 2010 were used (appendix 1 and 4) throughout the tests[23]. This generated a total of 1,400 different price updates over the 12 hour period of the simulation.

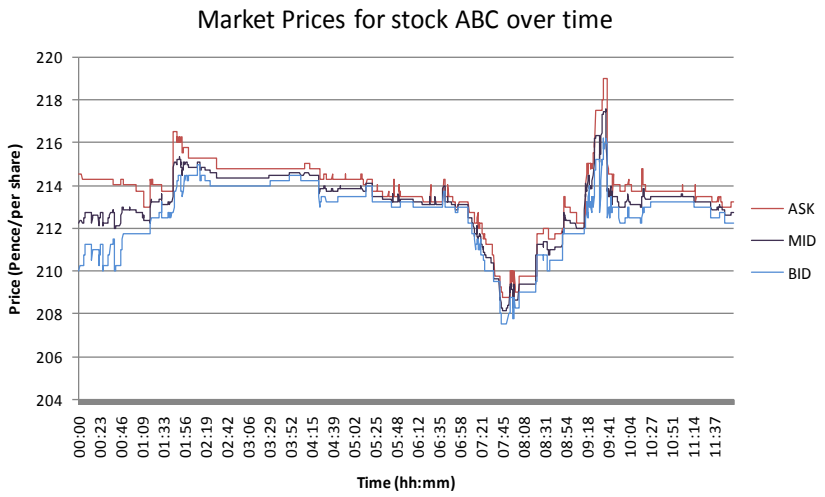


Fig 6.5a: Shows the market pricing for stock ABC used in the simulation

The graph in fig 6.5a above shows the overall price levels for stock ABC over the simulation timeline. All trading parties will use this data in their trading.

[23] Simulated pricing data could have also been modelled and used instead; however since the simulation will not be impacting this (external) market price directly, it is easier and more realistic to use real data; also if simulated pricing was used, it would need to factor in the variation in bid and ask price (spread) as well as general fluctuations, all of which would be readily present in the real data.

6.5.1 Introducing Market Price Updates with Mid Execution

Following on from the experiment in 6.4.4 (which combined VTC, ITA and HTV with decay factor), and using the exact same simulation scenario and settings, the test is repeated again, but this time with the market price element taken into account. This will result in different execution prices depending on when -and how long- an order takes to complete. To simplify and control the initial experiment, all trading parties in the multi agent simulation will trade at exactly the mid-price point, as published by the market data service; hence the dark pool venues will always execute orders at this price –which of course changes through time as shown in fig 6.5a above.

The expectation is that the results and SOR behaviour -in this test- should be similar to 6.4.4; since essentially, although the market price is changing, all trading parties are agreeing on executing at the prevailing mid-price. Each executed order is however given a more quantifiable meaning –with both volume and price now in the picture. The simulation was run and the following (fig 6.5b) shows a typical result of the executed orders and volume from the entire trading party population over time.

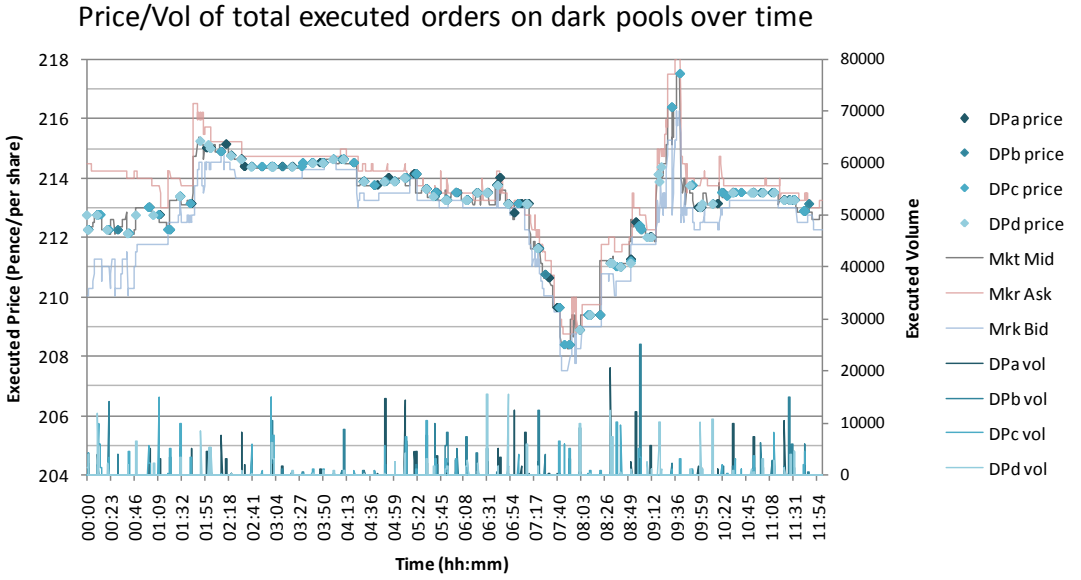


Fig 6.5b: Shows the price/volume of the executed orders (at mid-price) on the venues over time

The graph above shows that all the executed orders (at the price points illustrated) follow very closely along the market mid-price line throughout time. This demonstrates that the simulation behaviour is consistent with our expectation. The four 50k (SOR specific) sell orders were also executed at similar

timelines as in experiment 6.4.4, and the snapshot of SOR venue probability rankings (fig 6.5b) look familiar and identical as well, showing the trend in preferences shifting largely from DPc to DPb over time.

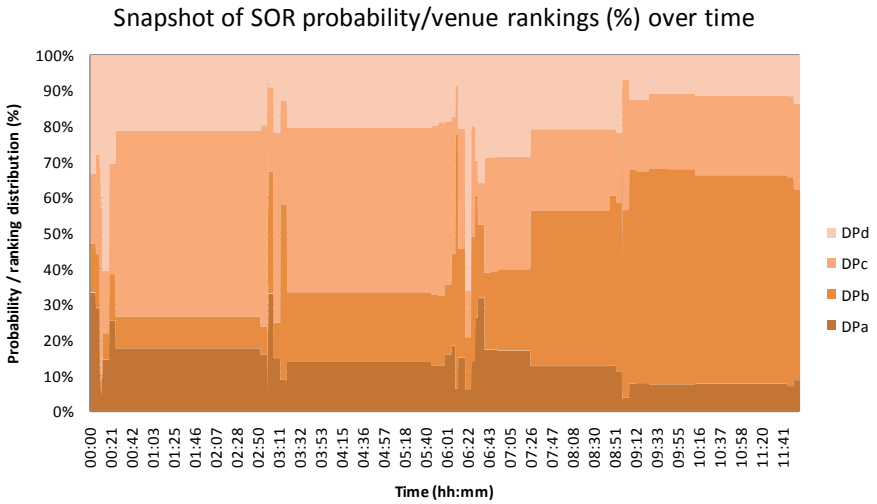


Fig 6.5b Shows a snapshot of SOR's venue probability rankings, similar results to 6.4.4

Order	Start Time (duration)	Volume	Average Price
AO101	00:00 (20.39mins)	50k	212.52p
AO102	03:00 (3.19mins)	50k	214.37p
AO103	06:00 (18.4mins)	50k	213.40p
AO104	09:00 (2.39mins)	50k	212.30p

Table 6.5c: Shows executed prices (for SOR orders) alongside similar time durations as in 6.4.4

The purpose of this experiment was to introduce the market price into the simulation, and hence highlight this in the orders executed. In table 6.5c above, the average price achieved for each parent order routed by SOR demonstrates the price volatility element, which varies depending on the market levels across each execution time and duration.

6.5.2 Dynamic Price Levels within the Market Bid-Ask Spread

Whilst there are cases where trades are executed near -or at- the market mid-price, the trading environment is complex and each party generally has different needs and investment objectives. There will be some that need to actively buy or sell very quickly and may be willing to pay a higher -or receive a lower- price in return; there will also be others that may have a longer term plan and are able to trade "passively", submitting orders with a limit price that is favourable to them. In addition, parties may alter their trading behaviours over time and may be more –or less- aggressive in pricing their orders. This needs to be captured in

some way into the simulation to ensure that a more realistic and competitive element is experienced in the dark pools' order book (beyond that of the simple timing rules discussed in 6.3.3). This will allow for further SOR experiments in a much more complex environment.

A simple model for creating pricing competition in the simulation is used. In this, a third of the order volume is priced around the mid price, i.e. to be executed near the mid-point between the market ask and bid, assuming of course that liquidity is present at the destination venue. A third is aggressively priced, positioned closer to the other end of the spread, i.e. for a buy, the order is priced above the mid-point, near to the ask level. The remainder third of the order volume is "passively" priced, seeking executions far better than the mid-point.

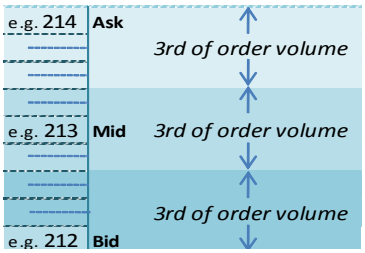


Fig 6.5d: Shows the general price ranges set by the agents in the simulation

Whilst the order price used for each trade routed by the trading agents is set within one of the above price ranges, the exact price will be randomly determined by each trading party (in 1/16th of a pence[24]). Here a variant of the zero-intelligence (ZI) trading strategy is applied, where each agent is assumed to have no intelligence of how to price an order, apart from the general price constrains as illustrated in fig 6.5d above. The use of zero-intelligence strategies in looking at trading behaviour in multi-agent simulations is widely covered in literature, e.g. J. Farmer, P. Patelli et al. [59, 60], where ZI was found to reasonably mimic trading behaviour when compared against samples from real trading data.

The same experiment from 6.5.1 (which was based on experiment 6.4.4) is repeated again, but only this time rather than using a constant mid-price, the pricing of orders is made dynamic. It follows the above model where each trading party (including the SOR enabled party) sets its own random price within a

[24] Tick/quote sizes are the minimum possible price increments that can be applied to the value of a stock in a trading venue or exchange. These generally vary between venues and the type of equity/instrument being traded. This value can be as low as 1/16th of a penny or cent, as is the case for the New York Stock Exchange and NASDAQ [66].

price/volume range (fig 6.5d), inside the bid-ask spread as published by the market service over time (fig 6.5a). This set price is constantly updated each time the market prices move, to ensure it remains within the spread. The simulation was run (multiple times) and a typical result is presented below.

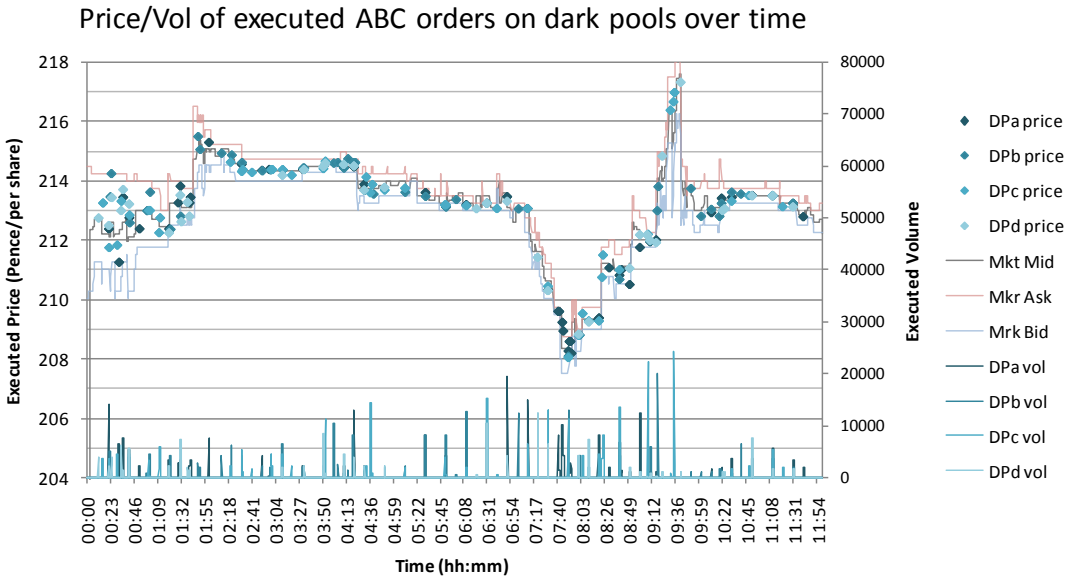


Fig 6.5e: Shows the price/volume of the executed orders on venues over time

What is clear from the results is that orders have now indeed executed at different levels within the market bid-ask spread throughout the simulation period. A sample of the order execution results for the SOR enabled party is shown below in fig 6.5f. With the exception of order pricing, everything in this simulation including the SOR elements used (i.e. HTV, ITA and VTC) were the same as before (experiment 6.4.4). The variation in order pricing -as set by each of the trading parties when being routed- has created further competition in the venues; hence, even with the use of the SOR techniques discussed so far, the execution of the four 50k sell orders take longer than before.

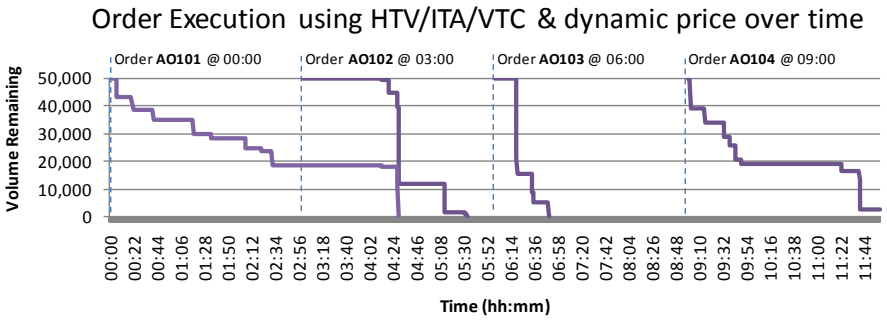


Fig 6.5f Shows the executions for the SOR party, pricing randomly within the bid-ask spread

Price priority in the venues (via limit order book discussed in 2.1.1) thus becomes more important than in the previous tests, impacting how orders are matched and executed. The parent order routed by the SOR system in our experiment can always improve its execution chances by being priced more aggressively and hence guaranteeing a higher price priority in the venues' order books. However, depending on the market and its movement over time, this may be at the expense of more favourable prices.

6.5.3 Execution Price Factors in Order Routing

The previous experiment demonstrated the varying price points that orders executed at along the market bid-ask spread on each of the venues. However on further inspection, what is more interesting is that at various stages of the simulation, and within similar "timeframes", some venues executed orders at better prices than others (for a particular volume). This can be highlighted clearer when looking at a closer section of the results, shown in fig 6.5g below.

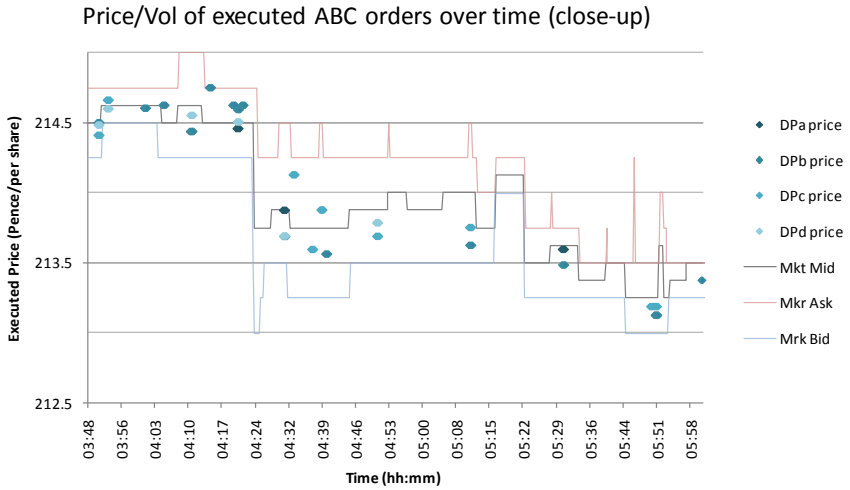


Fig 6.5g: Shows a close-up section plotting price-points of executed orders on venues over time

The close-up view shows many instances where two orders are executed at similar times (on different venues) but completed at different price points. Therefore our simulation model reflects well this very fact; i.e. that execution price for an order may vary from one venue to the other, and hence the price factor should play a role in the routing decisions of SOR. This is further reinforced by MiFID and FSA directives: *"if firms are able to access more than one execution venue and can make a direct and immediate comparison of prices then they should execute at the best price available to the firm."* [17]. Of course, this statement applies more specifically to open venues and exchanges where

price comparisons are possible; but nevertheless this should still be an aim for dark pools, as obtaining the best overall execution for an order –which may include its price- is the ultimate objective.

To demonstrate this further in the simulation, the same test conditions and environment from the previous experiments are repeated; only this time the test focuses on the execution of a particular order, routed at a set time within the simulation. A 75k sell order, randomly priced within the market mid-ask spread, is routed using the same splitting strategy and SOR elements (HTV, ITA and VTC) as before. The other parties continue to share identical behaviour and characteristics as in experiment 6.5.2, and originally in 6.4.4. The simulation was run and a sample of the results is shown in fig 6.5h below.

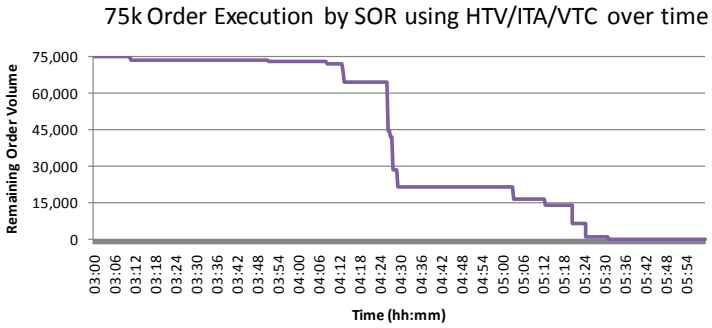


Fig 6.5h: Shows the execution of the 75k sell order over time

The 75k sell order (of stock ABC) begins executing at 3:00 and completes just before 5:30. Here we capture the exact execution price points for each part of the order (as routed by the SOR enabled party) and plot it alongside all the other executions that occurred at the venues.

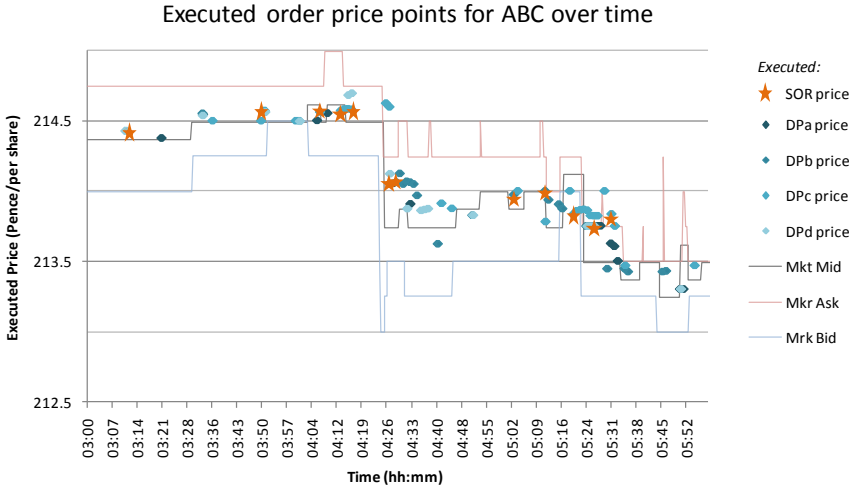


Fig 6.5i: Shows the price-points for all the executed orders on venues over time

From the graph in fig 6.5i, there were a number of instances where executions of stock ABC occurred in more than one venue simultaneously (or in very close proximity) but the SOR system routed and executed at the less favourable priced venue. A clear example is around 4:26, where a couple of executions occurred on venues at better price points (DPc and DPd) than the one SOR routed to (DPa). Therefore, a mechanism to factor execution price in the routing process is required. The Price Improvement Indicator (PII) was introduced and discussed back in chapter 3 to provide this very element as part of the routing algorithm. This created and placed more weighting on those venues that historically provided better execution prices in relation to the market mid-price. The formula representing the PII was factored into the overall routing algorithm as used in the tests so far, with weightings for each of its elements readjusted as follows:

$$rank_n = \frac{(VTC_n)^{-1}.\ 0.1}{\sum_{i=1}^{k}(VTC_i)^{-1}} + \frac{(rHTV_n).\ 0.25}{\sum_{i=1}^{k}(rHTV_i)} + \frac{(PII_n).\ 0.25}{\sum_{i=1}^{k}(PII_i)} + ITA_n.0.4$$

The 0.25 weighting should give the PII a fair amount of emphasis in the formula and provide the router with better price sensitivity. Again, these weighting factors are flexible and would be controlled and adjusted by humans or trading algorithms depending on the execution needs. The same experiment was re-run (multiple times), but this time with the PII element factored into the SOR formula; also a significant amount of the order volume -routed by the trading population- is submitted to a particular venue at a more favourable price than the others[25]. The following is a sample result from the many tests completed; it shows roughly similar execution times for the same 75k sell order as before.

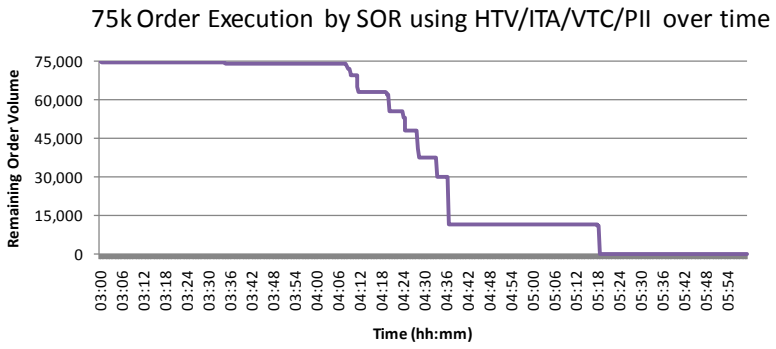


Fig 6.5j: Shows the execution of the 75k sell order over time (PII included)

[25] DPc is chosen here; the adjustment in the trading environment is made such that 50% of the sell orders that execute on DPc by any of the trading parties -including SOR- results in a better price (closer to the ask price) than any other venue at the time.

Examining the actual executions at the venues over time (shown in fig 6.5k below), it does appear that more SOR routed order executions occurred at the better priced venues than before; in this case the venue being DPc, as deliberately conditioned in this controlled experiment.

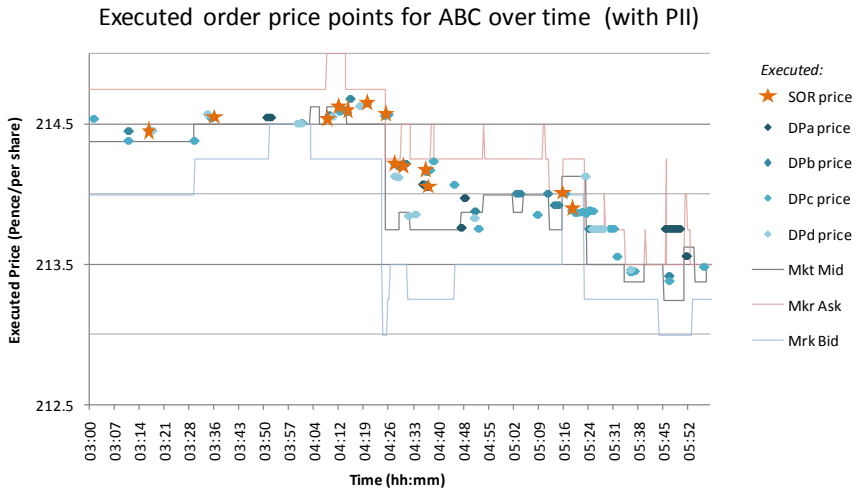


Fig 6.5k: Shows the price-points for all the executed orders on venues over time (with PII)

However, whilst improvements were noticed in some of the tests, further is needed in order to conclude the experiment; this is due to the complexity of the different factors that SOR must deal with here. With the full combination of VTC, HTV, ITA and PII as well as order splitting strategy, it can be difficult to reasonably conclude whether the PII element added value to the router or not. In fact, there were some tests which were found to be the opposite; in those, whilst the PII influence may have resulted in orders being routed to a better priced venue, they may have had little volume and hence would have been more cost effective to leave them. This can also be said with some of the other SOR elements of course, such as VTC, which is why varying the weighting factors are required to ensure that a fine tuned formula is achieved for different markets.

6.5.4 Increase in SOR use across trading party population

Further experiments were made, where half of the trading party population was configured to use the SOR concepts discussed so far. Early results showed that this significantly increased the amount of messaging generated (up to 3 fold) and resulted in far greater movement in liquidity from one venue to the other (as each SOR party is trying to chase and execute volume by splitting and rerouting regularly). Further tests are required to study this area in the future.

Chapter 7

Conclusions and Future Work

The aim of this project and research was to study and look at order routing within non-displayed financial venues (dark pools). A real-time routing algorithm was proposed, based on the research conducted, which could channel order flow across dark pools and take into account multiple factors in its decision making process. The SOR system developed was found to be largely effective in the hundreds of different experiments conducted across the project. The multi-agent simulation used for these tests was designed to incorporate many of the industry standard and real life characteristics found in trading systems today –to present a model as close as possible to reality.

7.1 Research Summary

In the earlier chapters we looked at the use of dark pools in industry and the subsequent market fragmentation, bringing smart order routing to prominence. We defined the research problem by drawing on literature and work around hidden liquidity and some of the challenges facing order routers in this domain. We then progressed with looking at some of the factors and elements that can affect and hence -when used correctly- aid in the routing decisions of our SOR system; this lead towards proposing and defining our dark routing algorithm, in a form similar to a probability formula. We discussed various order splitting strategies as well as other support components such as market data and trade reports to be utilised as part of the SOR system. This followed with a detailed design and implementation of the entire setup of the multi-agent environment. Each entity was designed and developed to play its role effectively as part of the SOR evaluation process, with its functions and characteristics defined. In the main evaluation chapter we conducted controlled experiments which were simple at first, testing small elements of our SOR system, in limited and heavily defined scenarios, and then later progressing towards more complicated areas. The aim and expectation, as mentioned earlier, was not to achieve a golden "one-size-fit-all" formula, but to better understand and demonstrate the SOR concepts proposed, and to show its effectiveness through the use of various simulations.

7.2 Main Findings and Conclusions

Throughout the SOR system's development and evaluation process, a number of observations and findings were made. We summarise and conclude the main ones here, drawing from the experiences gained across the project. Firstly, many of the SOR elements evaluated were generally found to be effective and produced good and clear results, especially under well structured and controlled simulations. For example, although this may be obvious, but the simple splitting of orders across more than one venue (discussed in Experiments I) and with the assumptions made, demonstrated the importance of utilising a splitting strategy in the routing mechanism –as opposed to submitting the whole order to a single venue. Continuous order re-routing between venues was also explored, with the results showing the time priority rules (in the venues' order book) affecting the total execution period of the order –especially in higher market activity (6.3.3). The simulation was able to show this well, which is a factor that routers in industry must consider as part of their decisions.

The introduction of the HTV element, as an indicator for building an analytical picture of historical volumes, as routed and executed by SOR (across each venue and for each traded stock) also fared well in the tests. The combined use of this element with ITA allowed SOR to factor both historical and immediate information when considering subsequent routings. Together with the variable splitting strategy (allocating larger lots of the order to the higher ranking venues based on the calculated probability), we showed that SOR's use of both historical and immediate activity clearly produced more favourable execution results (6.4.3) over the simpler fixed splitting and routing strategies (6.4.1). In fact, even with the introduction of real time market pricing (Experiment III), the HTV and ITA elements played a valued role in the decision making of the SOR system. It is anticipated that in cases where urgency in order execution is sought after, that the order is priced at a competitive point (i.e. market order or better) to ensure a higher level entry in the dark pool order books; and hence, be in a good position to capture liquidity if -and when- it exists.

Due to the nature of these experiments involving the use of historical volume (HTV), the simulation environment had to be adjusted in order to create a suitable scenario where we know for sure that some venues were indeed going to execute more volume than others throughout the tests. We progressed the

experiments by varying the volume executed from one venue to the other, introducing the decay factor to improve the router's ability to assess and adjust to the different market and volume fluctuations that may occur on the dark pools over time. The repeated results in this area were promising (6.4.3/4 and appendix 6.3), and although it is acknowledged that they occurred in a heavily defined multi agent environment, there was still a considerable level of autonomy given to each of the trading parties; this was alongside a certain amount of "uninformed" randomness to improve the realism of the simulations.

The use of a multi agent simulation in this project was important. One of the issues frequently encountered in research and literature is the testing of strategies against pure static and historical data [64]. This approach cannot take into account the effects and behaviours that the system its self (or strategy being tested) has on the data. This is addressed in the multi agent approach where the "cause and effect" is more realistic and any changes in our system can alter the overall outcome. For example, this can be shown in experiments 6.4.1 (fig 6.4d) and 6.4.3 (fig 6.4h) where the distribution of executed orders routed by SOR (across the venues) is compared against the entire executed orders in the simulation. The effect on the total order distribution due to the changes in the SOR routing behaviours (between those two experiments) can be seen clearly. An example of this is discussed further in appendix 6.6.

The later tests involving "real time" market prices demonstrated the different execution price points that can occur (within similar timeframes) across multiple venues (6.5.2/3). The introduction of the price improvement indicator (PII) into the SOR formula was however less convincing and clear. It remains to be seen whether it is actually possible to reach an optimal formula which combines the concepts of HTV, ITA, VTC and PII within its logic. It is more probable that different combinations or weightings of these factors are used based on the trading requirements. As noted earlier for example, where urgency in the order is a priority, then HTV, and more importantly ITA, would be given a higher weighting by the router. However where speculative or passive orders are placed, (i.e. orders seeking favourable prices and with no rush to execute) then VTC and PII could be raised in prominence.

7.3 Technical Challenges

This section is included to mention briefly the challenges experienced with messaging throughput and performance during some of the later demanding experiments. When market pricing data was introduced, a sharp (but expected) increase in messages occurred across the simulation. As trading parties reacted to price changes, more and more messages were exchanged with the venues. The load and pressure this created on the whole system was noticeable, affecting the overall results. An example of this is included in appendix 6.7. The issue was tackled by lower the simulation speed (to a factor of 30x for these type of demanding tests). The entire simulation and hardware infrastructure used for the project was found to have a limit of around 90 messages per second. Anything higher would have resulted in too much latency in the system, which would have affected the quality of the data output.

7.4 Further Development and Experiments

Some of the components of the original system, as outlined in chapter 3 and 4, were not evaluated due to time constraints. This includes the trade reporting service and hence the mHTV routing indicator. Experiments realising this component will add a further dimension to the tests and help show external trade reports (from the market) being consumed and fed back into the routing decisions. Evaluations could be made to analyse and demonstrate its utilisation in the system.

The order splitting used in SOR is another area that could be looked at and evaluated further. The splitting mechanisms discussed in the project could be redesigned to accommodate other more dynamic strategies. As an example, the order splitting could be programmed to be directly proportional to the probability values for each venue. So let's say if the routing algorithm produces probabilities of DPa: 0.2, DPb: 0.1, DPc: 0.4 and DPd: 0.2, then the splitting and routing of the order would be 20%, 10%, 40% and 20% across each venue respectively.

Another area, mentioned briefly in section 6.5.4, is looking at the effects of greater usage of SOR concepts across the trading party population. Experiments would help analyse the effectiveness of the different SOR strategies in an environment where more and more trading parties utilised the same or similar concepts.

Bibliography

[1] J.E. Doran, S. Franklin, N. Jennings and T. Norman. On Cooperation in Multi-Agent Systems. The Knowledge Engineering Review, 12(3). 1997

[2] R. De Winne and C. D'Hondt. Hide-and-seek in the market: Placing and detecting hidden orders. *Rev. Finance*, 11:663-692, 2007.

[3] M. Aitken and H. Berkman. The Use of Undisclosed Limit Orders on the Ausrlian Stock Exchange, *Journal of Banking & Finance*, 25, 2001.

[4] M. Ye. Price Formation, Transaction Cost and Market Share of the Dark Pool, *Department Of Economies, Cornell University*, Sept, 2009.

[5] R. Almgren and B. Harts. A Dynamic Algorithm for Smart Order Routing, *Whitepaper StreamBase Systems*, 2008

[6] A. Pardo and P. Pascual. On the Hidden Side of Liquidity, *Universidad de Las Islas Baleares*, 2003

[7] G. Sofianos. Dark Pools and Algorithmic Trading. *Journal of Trading*, Vol.1 No.4. Fall 2006.

[8] M. Kearns and L. Ortiz. The Penn-Lehman Automated Trading Project. *IEEE Intelligent Systems*, 2003

[9] G. Kendall and Y. Su. A Multi-agent Based Simulated Stock Market. *University of Nottingham*, 2004.

[10] T. Preis, S. Golke and W. Paul. Multi-agent-based Order Book Model of Financial Markets, *EPL Journal,* 75 510-516, Aug 2006.

[11] Trading Agents Competition (TAC/CAT), Market Based Control www.marketbasedcontrol.com

[12] T. Hendershott. Electronic Trading in Financial Markets. *The IT Pro Journal, IEEE Computer Society*. Aug 2003.

[13] L. Morgan, UBS News for Banks, UBS Investment Bank, Direct Execution. Augumn 2007

[14] D. Safarik. Dark Algorithms Solving Fragmentation Issues. *Advanced Trading Journal*, Jul 2007.

[15] J. Downes and J. Goodman, Iceberg / Order Books, Dictionary of Finance and Investment Terms, p45/62, Second Edition, 2006.

[16] M. Craig, Dark pools of liquidity remain untapped. *Financial News*. 29 Jun, 2007

[17] Markets in Financial Instruments Directive (MiFID), Financial Services Authority. FSA Publication DP06/3.

[18] M. Mizen. Trading in the Dark, Tabb Liquidity Matrix, *Tabb Group*, 2009

[19] D. Rawal. Bringing Intelligent Decision-Making to Order Routing, *Journal of Trading*, Vol. 5, No. 1: pp. 30-34, 2010

[20] G. Pujol. Smart Order Routing and Best Execution, *AMCIS Proceedings,* Paper 155, 2009

[21] CAPIS Nocturnal Algorithm, accessed in Nov 2009: www.capis.com/CAPIS_Algos.pdf

[22] GSAT Algorithmic Trading, Sonar/Sigma X, Whitepaper, *Goldman Sachs Electronic Trading*, 27 Nov, 2007

[23] Financial Insights Research Group. www.financial-insights.com

[24] Advanced Trading Journal Dark Pool Directory and Summary, accessed in Mar 2010: http://www.advancedtrading.com/directories/darkpool/

[25] E. Lederman, Turquoise Tariffs for Non-Displayed Orders, *Automated Trader Journal*, 1 October, 2009

[26] R. Balarkas, T. Baylis, B. Hunt, Liquidity Sensing in SOR: Roundtable, *Automated Trader Magazine*, issue 5, April, 2007

[27] Order/Execution Management System Product Round-up, *Advanced Trading*, Sept, 2007.

[28] O. Streltchenko, Y. Yesha, T. Finin. Multi-Agent Simulation of Financial Markets, University of Maryland Baltimore, 2002

[29] The Dark Pools Directory, *Advanced Trading*, accessed in Mar 2010: http://advancedtrading.thewallstreetwiki.com/directories/directory-dark-pools.php

[30] P. Kratzm, T. Schoneborn, Optimal Liquidation in Dark Pools, *Berlin University*, June, 2009.

[31] L. Harris, Does a Large minimum Price Variation Enourage Order Exposure? *University of Southern California*, 1997.

[32] Feature article: Exposing the Identity of Dark Pools in Real Time, *Advanced Trading*, Mar 2010.

[33] J. Białkowski, S. Darolles, Improving VWAP strategies: A dynamical volume approach. *Auckland University of Technology*, Apr 2006.

[34] A. Shleifer, Clarendon Lectures: Inefficient Markets, *Oxford University Press*, 2000.

[35] F. Black, "Noise", *Journal of Finance*, vol. 41, pp. 529–543, 1986

[36] Markit BOAT, MiFID Trade Reporting Service, accessed in Apr 2010: http://www.markit.com/en/products/data/boat/boat.page

[37] Bloomberg Market Data, accessed in Apr 2010: http://www.bloomberg.com/markets

[38] Reuters Market Data Service, accessed in Apr 2010: https://customers.reuters.com/Home/RMDS.aspx

[39] TWAP and VWAP Algorithmic Execution strategies, Steambase, accessed in Mar 2010: http://dev.streambase.com/algorithmic-trading-twap.htm

[40] National Best Bid Offer definition, *Investopaedia*

[41] Repast Agent Simulation Toolkit, accessed in May 2010: http://repast.sourceforge.net

[42] Ascape Agent Simulation, accessed in May 2010: http://ascape.sourceforge.net

[43] Swarm Muli Agent System, accessed in May 2010: http://www.swarm.org

[44] NetLogo Multi Agent Programmable Modelling Environment, accessed in May 2010: http://ccl.northwestern.edu/netlogo

[45] Eclipse IDE, The Eclipse Foundation, http://www.eclipse.org

[46] IBM Websphere MQ, http://www.ibm.com/software/integration/wmq

[47] S. Davies, P. Broadhurst, Websphere MQ Fundamentals, *Published by IBM Redbooks,* 2005

[48] D. Thiben, Integrating Trading and Load Balancing for Efficient Management of Services in Distributed Systems, *Aachen University of Technology, published by Heidelberg, p42-53, volume 1890*, 2000

[49] T. Bourke, Server Load Balancing, *published by O'Reilly*, 2001

[50] The Financial Information Exchange (FIX) Protocol Organisation, http://www.fixprotocol.org

[51] R. Sargent, Verification and Validation of Simulation Models, Winter Simulation Conference / Syracuse University, New York, 1998

[52] K. Carley, Validating Computational Models, Carnegie Mellon University, Sept 1996

[53] K. Troitzsch, Validating Simulation Models, University of Koblenz-Landau, Germany, 2004

[54] BlocSec Dark Pool, accessed in May 2010: http://www.blocsec.com

[55] B. Barber, Terrance Odean, Nig Zhu, Do Noise Traders Move Markets? *Graduate School of Management and Hass School of Business*, 2006

[56] S. Alfarano, T. Lux, A Minimial Noise Trader Model with Realistic Time Series Properties, *Department of Economics, University of Kiel,* 2006

[57] N. Mehta, Why Some Dark Pools Are Increasing Their Volumes, *Traders Magazine*, March 2009

[58] Liquidity Access Fees Not Top Priority for SOR, *Waters Technology*, May 2010.

[59] J. Doyne Farmer, P. Patelli, The Predictive Power of Zero Intelligence in Financial Markets, Santa Fe Instiitute, 2004

[60] 'Zero Intelligence' trading closely mimics stock market, The New Scientist, Feb 2005.

[61] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, first published in 1994.

[62] A. Agarwal, P. Bartlett, M. Dama, Optimal Allocation Strategies for the Dark Pool Problem, University of California, 2010

[63] K. Ganchev, M. Kearns, Y. Nevmyvaka, Censored Exploration and the dark pool problem, *Proceedings of Uncertainty in Artificial Intelligence*, UAI 2009

[64] K. Khunna, M. Smith, D. Wu, T. Zhang, Reconstructing the Order Book, Department of Mathematics, Stanford University, 2009

[65] QuickFIX/J, Fully featured FIX engine, http://www.quickfixengine.org

[66] R. Huang, H. Stoll, Tick Size, Bid-Ask Spreads and Market Structure, *University of Notre Dame*, 2000

[67] P. Sheridan, G. Sofianos, The Cubist Approach to an Execution Strategy, *Financial Times Mandate*, May 2005

APPENDIX 1

Historical volume distribution for a selection of stocks traded across dark pools. Source: Fidessa's Fragulator tool, http://fragmentation.fidessa.com/fragulator

Multiple three-day snapshots were taken across March 2010, showing the volume distribution of trades in lit/dark pools. Both large and mid-cap stocks were sampled.

Stock: HMV	**8-10 March 2010**	**15-17 March 2010**	**22-24 March 2010**
Total volume traded:	20,245,594	15,477,901	18,842,170
Volume traded in dark:	1,528,559 (7.53%)	544,816 (3.52%)	855,285 (4.54%)
Dark Pools:	*Volume:*		
Bats Dark	*54,008*	*13,887*	*2,176*
Nomura NX	*22,834*	*36,218*	*45,263*
Posit	*828,833*	*42,700*	*0*
BlockMatch	*39,858*	*54,208*	*11,668*
Chi-Delta	*399,811*	*120,165*	*449,141*
Turquoise Dark	*183,215*	*277,638*	*347,037*

Stock: BP	**8-10 March 2010**	**15-17 March 2010**	**22-24 March 2010**
Total volume traded:	233,937,856	179,466,565	199,124,515
Volume traded in dark:	2,312,541 (1%)	3,794,302 (2.12%)	2,360,345 (1.2%)

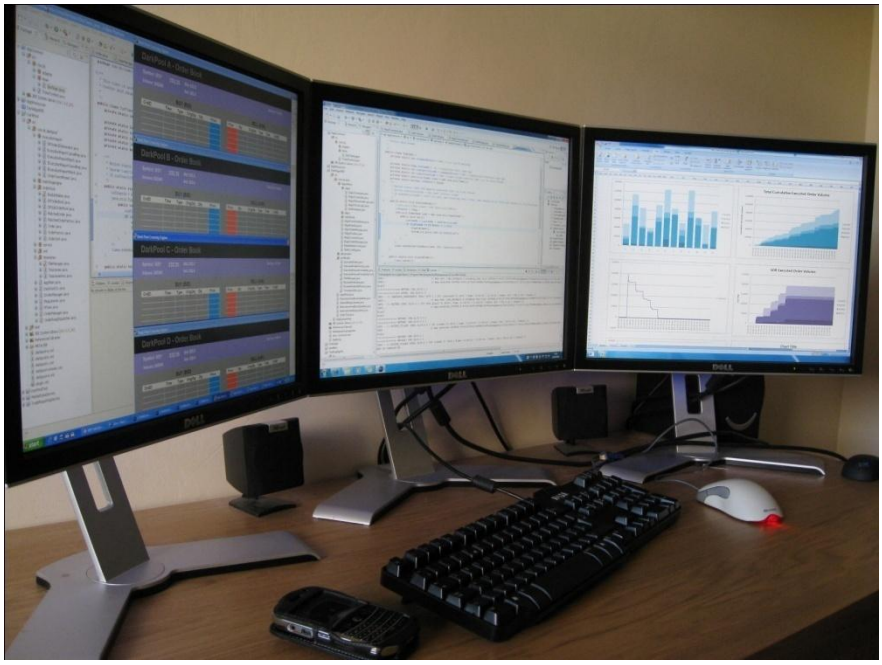
Stock: ARM	**8-10 March 2010**	**15-17 March 2010**	**22-24 March 2010**
Total volume traded:	62,816,289	46,592,532	35,340,818
Volume traded in dark:	1,887,845 (3%)	695,819 (1.5%)	684,686 (2%)

Stock: KGF	**8-10 March 2010**	**15-17 March 2010**	**22-24 March 2010**
Total volume traded:	95,712,008	83,502,735	101,663,395
Volume traded in dark:	6,570,638 (6.81%)	2,702,424 (3.26%)	6,215,222 (6.1%)

APPENDIX 2

The specification for the hardware/server PCs used in the simulation is as follows: *(all connected via Ethernet cable/networking and using Websphere MQ messaging for communication)*

	Server A
Hardware Specification:	Intel Core Quad CPU 2.3 GHz / 4 GB ram
Operating System:	Windows XP
Hosting:	All four Dark Pool Trading Venues
	Server B
Hardware Specification:	Intel Duo CPU 2.4 GHz / 3 GB ram
Operating System:	Windows XP
Hosting:	Trading parties
	Server C
Hardware Specification:	Intel Duo CPU 2.4 GHz / 3 GB ram
Operating System:	Windows XP
Hosting:	Trading parties
	Server D
Hardware Specification:	Intel Pentium CPU 2.8 GHz / 2 GB ram
Operating System:	Windows XP
Hosting:	Market Data and Trade Reporting Service



The system with a sample simulation running (showing 3 of the 4 server monitors)

APPENDIX 3

The following list details the fields definitions used across all the FIX messages adopted in the simulation. The complete list of 1,600+ fields and 100+ messages supported by the FIX protocol can be found on the FIX website: http://www.fixprotocol.org.

FIX Message **Standard Header** fields:

FIX Tag	Field Name	Data Type	Description	Valid Values
8	BeginString	String	Identifies beginning of new message and protocol version. ALWAYS FIRST FIELD IN MESSAGE.	FIX{version}
9	BodyLength	Length	Message length, in bytes, forward to the CheckSum field. ALWAYS SECOND FIELD IN MESSAGE.	{Length}
35	MsgType	Char	Defines message type. ALWAYS THIRD FIELD IN MESSAGE.	8 = ExecutionReport D = NewOrderSingle E = NewOrderList F = OrderCancelRequest G = OrderCancelReplaceRequest H = OrderStatusRequest J = AllocationInstruction K = ListCancelRequest (+ others)
49	SenderCompID	String	Assigned value used to identify firm sending message.	{String}
56	TargetCompID	String	Assigned value used to identify receiving firm.	{String}
52	SendingTime	UTC Timestamp	Time of message transmission, in UTC (Coordinated Universal Time)	{YYYYMMDD-HH:MM:SS:ms}

FIX Message **Body** fields: *(showing required/utilised fields only)*

FIX Tag	Field Name	Data Type	Description	Valid Values
11	ClOrdID	String	Unique identifier for Order as assigned by trading party.	{String}
41	OrigClOrdID	String	ClOrdID (11) of the previous order as assigned by the trading party, used to identify the previous order in cancel/replace requests.	{String}
55	Symbol	String	Ticker symbol of the security to be.	{String}
54	Side	Char	The side of the order, buy,	1 = Buy

			sell etc (many others)	2 = Sell (+others)
38	OrderQty	Qty	Quantity ordered, i.e. represents the number of shares.	{Qty}
40	OrdType	Char	Order type	1 = Market 2 = Limit 3 = Stop / Stop Loss 4 = Stop Limit (+others)
39	OrdStatus	Char	Identifies current status of order (used in Execution Report messages)	0 = New 1 = Partially filled 2 = Filled 3 = Done for day 4 = Cancelled 5 = Replaced 6 = Pending Cancel 7 = Stopped 8 = Rejected (+others)
44	Price	Price	The price willing to buy/sell if set as a limit order.	{Price}
15	Currency	Currency	Identifies currency used for price.	{Currency}
434	CxlRejResponseTo	Char	Identifies the type of request that a Cancel Reject is in response to.	1 = Order cancel request 2 = Order cancel/replace request

FIX Message **Standard Trailer** fields:

FIX Tag	Field Name	Data Type	Description	Valid Values
10	CheckSum	String	Three byte, simple checksum. ALWAYS LAST FIELD IN MESSAGE; i.e. serves, with the trailing <SOH>, as the end-of-message delimiter. Always defined as three characters. (Always unencrypted)	{String}

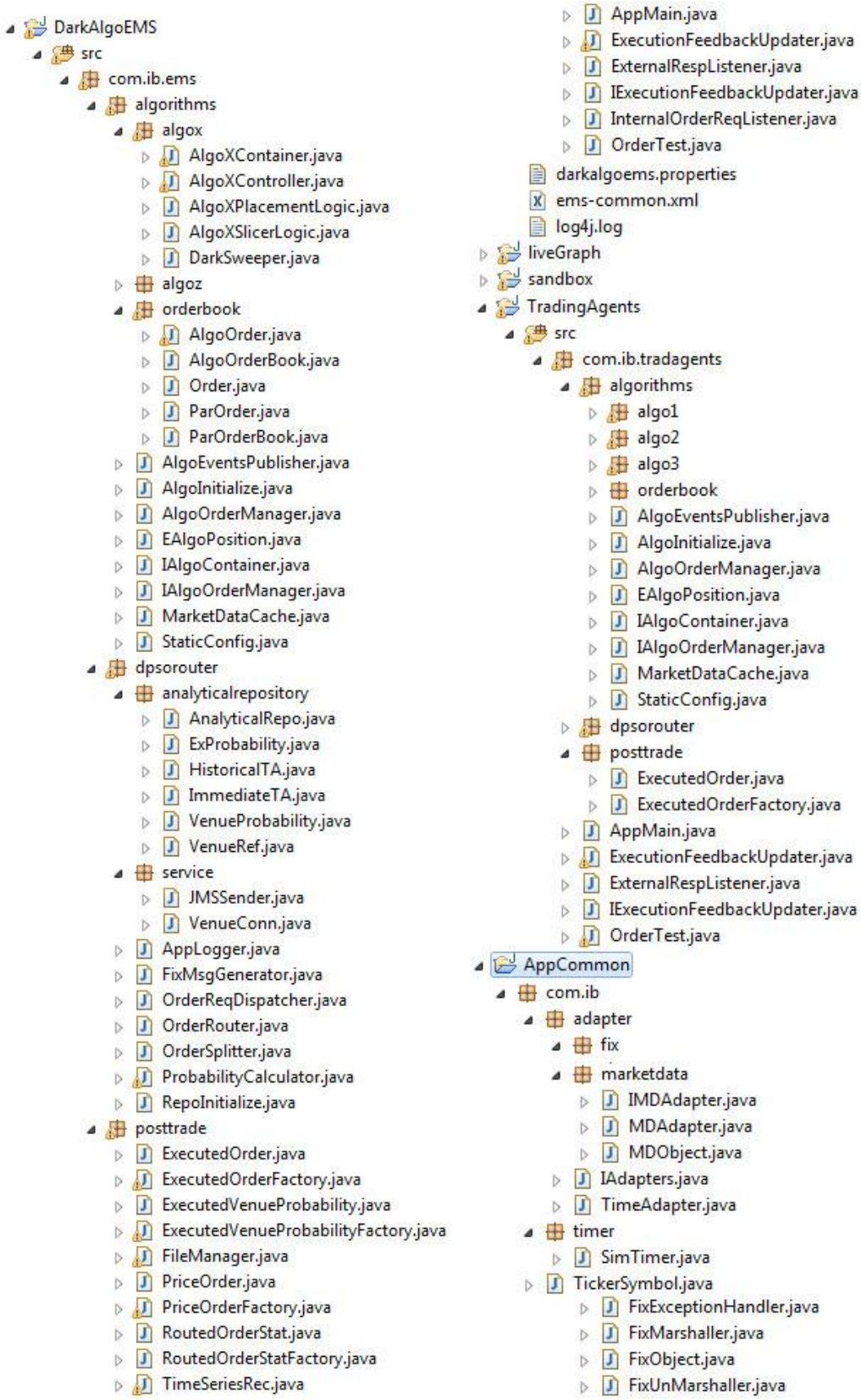
APPENDIX 4

Sample extract from the market data file used in the simulation. This is based on real tick prices for a stock, as discussed in the evaluation chapter.

08:00:09	STOCK:	ABC	ASK:	214.5	BID:	210	MID:	212.25
08:00:42	STOCK:	ABC	ASK:	214.5	BID:	210.25	MID:	212.375
08:03:36	STOCK:	ABC	ASK:	214.5	BID:	210.25	MID:	212.375
08:03:57	STOCK:	ABC	ASK:	214.25	BID:	210.25	MID:	212.25
08:05:59	STOCK:	ABC	ASK:	214.25	BID:	210.5	MID:	212.375
08:06:00	STOCK:	ABC	ASK:	214.25	BID:	210.75	MID:	212.5
08:06:59	STOCK:	ABC	ASK:	214.25	BID:	211	MID:	212.625
08:07:00	STOCK:	ABC	ASK:	214.25	BID:	211.25	MID:	212.75
08:14:30	STOCK:	ABC	ASK:	214.25	BID:	210.25	MID:	212.25
08:14:31	STOCK:	ABC	ASK:	214.25	BID:	210.5	MID:	212.375
08:14:45	STOCK:	ABC	ASK:	214.25	BID:	210.75	MID:	212.5
08:14:46	STOCK:	ABC	ASK:	214.25	BID:	211	MID:	212.625
08:22:17	STOCK:	ABC	ASK:	214.25	BID:	210	MID:	212.125
08:22:18	STOCK:	ABC	ASK:	214.25	BID:	210.25	MID:	212.25
08:22:49	STOCK:	ABC	ASK:	214.25	BID:	210.5	MID:	212.375
08:22:50	STOCK:	ABC	ASK:	214.25	BID:	210.75	MID:	212.5
08:23:44	STOCK:	ABC	ASK:	214.25	BID:	211	MID:	212.625
08:23:45	STOCK:	ABC	ASK:	214.25	BID:	211.25	MID:	212.75
08:26:59	STOCK:	ABC	ASK:	214.25	BID:	210	MID:	212.125
08:27:00	STOCK:	ABC	ASK:	214.25	BID:	210.25	MID:	212.25
08:32:28	STOCK:	ABC	ASK:	214.25	BID:	210.5	MID:	212.375
08:32:29	STOCK:	ABC	ASK:	214.25	BID:	210.75	MID:	212.5
08:32:56	STOCK:	ABC	ASK:	214.25	BID:	210.75	MID:	212.5
08:34:28	STOCK:	ABC	ASK:	214.25	BID:	211	MID:	212.625
08:34:30	STOCK:	ABC	ASK:	214.25	BID:	211.25	MID:	212.75
08:38:33	STOCK:	ABC	ASK:	214.25	BID:	211.5	MID:	212.875
08:38:34	STOCK:	ABC	ASK:	214	BID:	211.5	MID:	212.75
08:39:32	STOCK:	ABC	ASK:	214	BID:	210	MID:	212
08:40:32	STOCK:	ABC	ASK:	214	BID:	210.25	MID:	212.125
08:46:04	STOCK:	ABC	ASK:	214.25	BID:	210.25	MID:	212.25
08:46:05	STOCK:	ABC	ASK:	214.25	BID:	210.5	MID:	212.375
08:46:35	STOCK:	ABC	ASK:	214.25	BID:	210.75	MID:	212.5
08:46:36	STOCK:	ABC	ASK:	214.25	BID:	211	MID:	212.625
08:47:31	STOCK:	ABC	ASK:	214.25	BID:	211.25	MID:	212.75
08:47:32	STOCK:	ABC	ASK:	214.25	BID:	211.5	MID:	212.875
08:48:32	STOCK:	ABC	ASK:	214.25	BID:	211.75	MID:	213
08:48:33	STOCK:	ABC	ASK:	214	BID:	211.75	MID:	212.875
08:56:04	STOCK:	ABC	ASK:	214.25	BID:	211.75	MID:	213
08:56:05	STOCK:	ABC	ASK:	214	BID:	211.75	MID:	212.875
08:56:50	STOCK:	ABC	ASK:	214	BID:	211.75	MID:	212.875
09:03:37	STOCK:	ABC	ASK:	213.75	BID:	211.75	MID:	212.75
09:11:09	STOCK:	ABC	ASK:	213.25	BID:	211.75	MID:	212.5
09:11:10	STOCK:	ABC	ASK:	213.5	BID:	211.75	MID:	212.625
09:11:25	STOCK:	ABC	ASK:	213.5	BID:	211.75	MID:	212.625
09:11:35	STOCK:	ABC	ASK:	213.25	BID:	211.75	MID:	212.5
09:11:41	STOCK:	ABC	ASK:	213.25	BID:	211.75	MID:	212.5
09:11:51	STOCK:	ABC	ASK:	213	BID:	211.75	MID:	212.375
09:11:56	STOCK:	ABC	ASK:	213	BID:	211.75	MID:	212.375
09:18:02	STOCK:	ABC	ASK:	213	BID:	211.75	MID:	212.375
09:19:05	STOCK:	ABC	ASK:	212.75	BID:	211.75	MID:	212.25
09:19:15	STOCK:	ABC	ASK:	214.25	BID:	211.75	MID:	213
09:19:31	STOCK:	ABC	ASK:	214.25	BID:	212.25	MID:	213.25
09:19:32	STOCK:	ABC	ASK:	214	BID:	212	MID:	213
09:19:52	STOCK:	ABC	ASK:	214.25	BID:	212.5	MID:	213.375
09:20:53	STOCK:	ABC	ASK:	214	BID:	212.5	MID:	213.25
09:28:23	STOCK:	ABC	ASK:	214.25	BID:	212.5	MID:	213.375
09:28:24	STOCK:	ABC	ASK:	214	BID:	212.5	MID:	213.25
09:28:53	STOCK:	ABC	ASK:	214	BID:	212.75	MID:	213.375

APPENDIX 5.1

The following shows parts of the class and package structure of the SOR system and trading party developed.



APPENDIX 5.2

The following shows parts of the class and package structure of the dark pool venue system developed. It also shows the market data service.

APPENDIX 6.1

Following tables show a sample of the raw results from experiment 6.3.2 for the different order sizes and volume using both single and split mode routing (across two venues), along with completion times for each repeated test. The average time is highlighted at the bottom of each column/test category.

Tests for order volume: 2,500 to 20,000

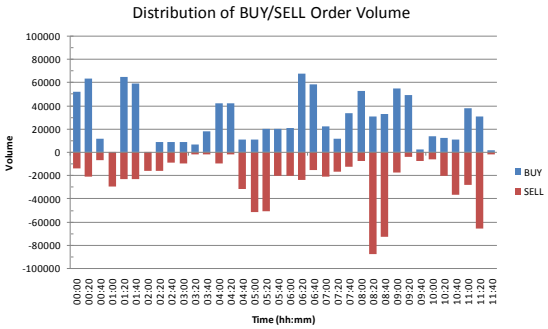
Routing:	Single	Split	Single	Split	Single	Split	Single	Split
Volume:	2,500	2,500	5,000	5,000	10,000	10,000	20,000	20,000
Repeated Tests:	01:30	01:20	00:01	01:30	00:01	00:06	00:06	00:06
	00:15	00:30	00:01	00:30	00:06	01:20	04:00	03:00
	00:06	00:30	01:30	00:15	00:30	00:15	00:06	00:06
	00:06	00:30	00:15	00:06	00:06	00:06	01:30	01:30
	00:06	01:35	00:06	00:06	00:06	00:06	01:20	00:06
	00:06	01:20	00:06	01:20	00:06	01:20	01:30	00:06
	00:01	00:06	00:30	00:30	01:30	03:00	00:06	02:30
	00:35	00:06	00:30	01:30	01:30	00:06	00:06	00:30
	01:30	01:20	00:01	01:30	00:01	00:06	00:06	00:06
	00:15	00:30	00:01	00:30	00:06	01:20	04:00	03:00
	00:06	00:30	01:30	00:15	00:30	00:15	00:06	00:06
	00:06	00:30	00:15	00:06	00:06	00:06	01:30	01:30
	00:06	01:35	00:06	00:06	00:06	00:06	01:20	00:06
	00:06	01:20	00:06	01:20	00:06	01:20	01:30	00:06
	00:01	00:06	00:30	00:30	01:30	03:00	00:06	02:30
	00:35	00:06	00:30	01:30	01:30	00:06	00:06	00:30
Average:	00:20	**00:44**	00:22	**00:43**	00:29	**00:47**	01:05	**00:59**

Tests for order volume: 50,000 to 200,000

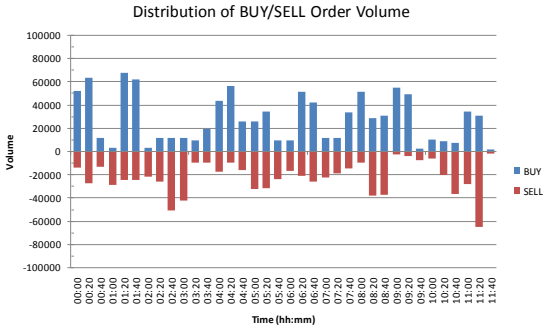
Single	Split	Single	Split	Single	Split	Single	Split
50,000	50,000	100,000	100,000	150,000	150,000	200,000	200,000
03:05	04:10	06:35	01:30	06:30	06:30	09:10	06:30
04:05	01:30	06:31	01:30	06:30	04:30	09:10	05:10
00:07	04:10	04:10	04:05	05:10	03:50	11:59	04:30
01:30	01:30	01:30	01:30	04:10	04:05	10:10	04:05
01:03	01:30	07:30	03:50	05:30	05:30	11:59	06:00
04:40	00:30	06:04	01:30	09:10	03:30	11:59	04:30
01:20	01:30	04:13	01:20	08:45	05:10	07:45	04:40
01:30	01:20	01:30	01:30	10:11	01:30	08:30	04:50
03:05	04:10	06:35	01:30	06:30	06:30	09:10	06:30
04:05	01:30	06:31	01:30	06:30	04:30	09:10	05:10
00:07	04:10	04:10	04:05	05:10	03:50	11:59	04:30
01:30	01:30	01:30	01:30	04:10	04:05	10:10	04:05
01:03	01:30	07:30	03:50	05:30	05:30	11:59	06:00
04:40	00:30	06:04	01:30	09:10	03:30	11:59	04:30
01:20	01:30	04:13	01:20	08:45	05:10	07:45	04:40
01:30	01:20	01:30	01:30	10:11	01:30	08:30	04:50
02:10	**02:01**	04:45	**02:05**	06:59	04:19	10:05	05:01

APPENDIX 6.2

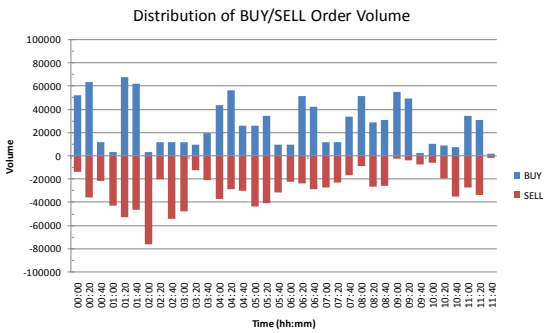
Following tables show a sample of the raw results from experiment 6.3.3 for the 200k sell order when routed to DPa only, or cancelled and re-routed sequentially from one venue to the next (in 10 sec, 5min and 30min intervals). This is repeated across different market conditions as explained in then experiment.



	Routing intervals			
	No Reroute	10sec	5min	30min
	07:30	04:11	04:15	04:30
	09:15	04:11	04:11	03:04
	11:10	04:10	04:11	04:50
	07:11	04:20	04:15	04:40
	08:50	04:20	04:20	04:30
	07:30	04:11	04:10	04:20
	08:40	04:11	04:11	04:50
	08:25	04:05	04:15	04:30
	11:10	04:11	04:11	03:04
	07:11	04:10	04:15	04:50
	08:50	04:20	04:20	04:40
	07:30	04:20	04:10	04:30
	08:40	04:11	04:11	04:20
	08:25	04:11	04:15	04:50
	11:10	04:05	04:11	04:30
Average:	08:33	04:12	04:13	04:24



No Reroute	10sec	5min	30min
11:30	07:10	07:55	08:20
07:40	06:51	08:20	06:34
08:10	07:30	07:30	07:10
08:30	07:40	07:01	07:39
11:10	06:50	06:35	07:45
08:20	07:10	07:05	06:50
07:10	07:50	08:10	08:10
08:30	07:30	08:35	07:40
07:40	07:10	07:55	08:20
08:10	06:51	08:20	06:34
08:30	07:30	07:30	07:10
11:10	07:40	07:01	07:39
08:20	06:50	06:35	07:45
07:10	07:10	07:05	06:50
08:30	07:50	08:10	08:10
08:52	07:18	07:38	07:31



No Reroute	10sec	5min	30min
10:10	12:00	11:10	12:00
09:30	12:00	11:30	12:00
08:10	12:00	11:30	12:00
08:50	12:00	11:50	12:00
08:50	12:00	11:00	12:00
09:05	12:00	10:50	12:00
08:40	12:00	12:00	12:00
08:10	12:00	11:30	12:00
10:10	12:00	11:10	12:00
09:30	12:00	11:30	12:00
08:10	12:00	11:30	12:00
08:50	12:00	11:50	12:00
08:50	12:00	11:00	12:00
09:05	12:00	10:50	12:00
08:40	12:00	12:00	12:00
08:55	12:00	11:25	12:00

APPENDIX 6.3

Following table shows a breakdown of all four 50k orders as they were executed across the different venues. The SOR system split each order into four smaller and equal lots and routed across all venues. The results are based on the experiment from section 6.41.

EXECUTED TIME	ORDER	QTY	VENUE	VENUE COST	EXECUTED TIME	ORDER	QTY	VENUE	VENUE COST
ORDER **AO101.0** @ 00:00					**ORDER AO103.0** @ 06:00				
00:01:24	AO101.0	3233	DPa	0.3	**06:00:01**	AO103.2	1938	DPc	0.5
00:01:24	AO101.1	2766	DPb	0.7	**06:00:01**	AO103.2	5166	DPc	0.5
00:01:24	AO101.2	3353	DPc	0.5	**06:00:01**	AO103.2	5166	DPc	0.5
00:01:24	AO101.3	9600	DPd	0.3	**06:00:02**	AO103.2	230	DPc	0.5
00:01:56	AO101.2	2418	DPc	0.5	**06:30:39**	AO103.1	12500	DPb	0.7
00:01:59	AO101.2	3562	DPc	0.5	**06:30:59**	AO103.3	6772	DPd	0.3
00:01:59	AO101.3	2968	DPd	0.3	**06:30:57**	AO103.3	2916	DPd	0.3
00:10:27	AO101.2	2266	DPc	0.5	**06:30:57**	AO103.3	2812	DPd	0.3
00:10:39	AO101.3	332	DPd	0.3	**07:00:39**	AO103.0	2165	DPa	0.3
00:10:39	AO101.0	2668	DPa	0.3	**07:10:39**	AO103.0	5766	DPa	0.3
00:10:39	AO101.2	501	DPc	0.5	**07:20:39**	AO103.0	4569	DPa	0.3
00:10:39	AO101.0	6499	DPa	0.3					
00:20:39	AO101.1	3333	DPb	0.7					
00:29:39	AO101.1	701	DPb	0.7					
00:30:41	AO101.1	2666	DPb	0.7					
00:30:41	AO101.1	3134	DPb	0.7					
ORDER **AO102.0** @ 03:00					**ORDER AO104.0** @ 09:00				
03:00:19	AO102.0	2142	DPa	0.3	**09:00:39**	AO104.2	12500	DPc	0.5
03:00:19	AO102.0	7617	DPa	0.3	**09:09:19**	AO104.3	1071	DPd	0.3
03:00:19	AO102.1	497	DPb	0.7	**09:10:59**	AO104.3	11429	DPd	0.3
03:00:19	AO102.2	5210	DPc	0.5	**10:08:59**	AO104.0	375	DPa	0.3
03:00:19	AO102.3	666	DPd	0.3	**10:30:19**	AO104.0	3750	DPa	0.3
03:00:19	AO102.1	2298	DPb	0.7	**10:30:39**	AO104.0	3750	DPa	0.3
03:00:19	AO102.0	2727	DPa	0.3	**10:50:39**	AO104.1	1514	DPb	0.7
03:00:19	AO102.2	5857	DPc	0.5	**11:10:34**	AO104.0	4459	DPa	0.3
03:00:19	AO102.2	1433	DPc	0.5	**11:10:34**	AO104.1	4458	DPb	0.7
03:00:39	AO102.0	14	DPa	0.3	**11:10:34**	AO104.1	3750	DPb	0.7
03:04:59	AO102.3	2142	DPd	0.3	**11:20:19**	AO104.0	166	DPa	0.3
03:06:19	AO102.3	1762	DPd	0.3	**11:20:39**	AO104.1	2000	DPb	0.7
03:09:39	AO102.1	3119	DPb	0.7	**11:30:19**	AO104.1	778	DPb	0.7
03:10:39	AO102.3	666	DPd	0.3	**08:59:39**	AO104.2	12500	DPc	0.5
03:18:59	AO102.3	1560	DPd	0.3	**09:09:19**	AO104.3	1071	DPd	0.3
03:20:39	AO102.3	2148	DPd	0.3	**09:10:59**	AO104.3	11429	DPd	0.3
03:30:39	AO102.3	666	DPd	0.3	**10:08:59**	AO104.0	375	DPa	0.3
03:30:39	AO102.1	666	DPb	0.7	**10:30:13**	AO104.0	3750	DPa	0.3
03:30:39	AO102.3	668	DPd	0.3	**10:30:33**	AO104.0	3750	DPa	0.3
03:50:39	AO102.1	666	DPb	0.7	**10:50:39**	AO104.1	1514	DPb	0.7
03:50:39	AO102.1	668	DPb	0.7	**11:10:39**	AO104.0	4459	DPa	0.3
03:59:39	AO102.1	4586	DPb	0.7	**11:10:39**	AO104.1	4458	DPb	0.7
04:08:19	AO102.3	113	DPd	0.3	**11:20:19**	AO104.0	166	DPa	0.3
04:20:39	AO102.3	2109	DPd	0.3	**11:20:39**	AO104.1	2000	DPb	0.7
					11:30:19	AO104.1	778	DPb	0.7

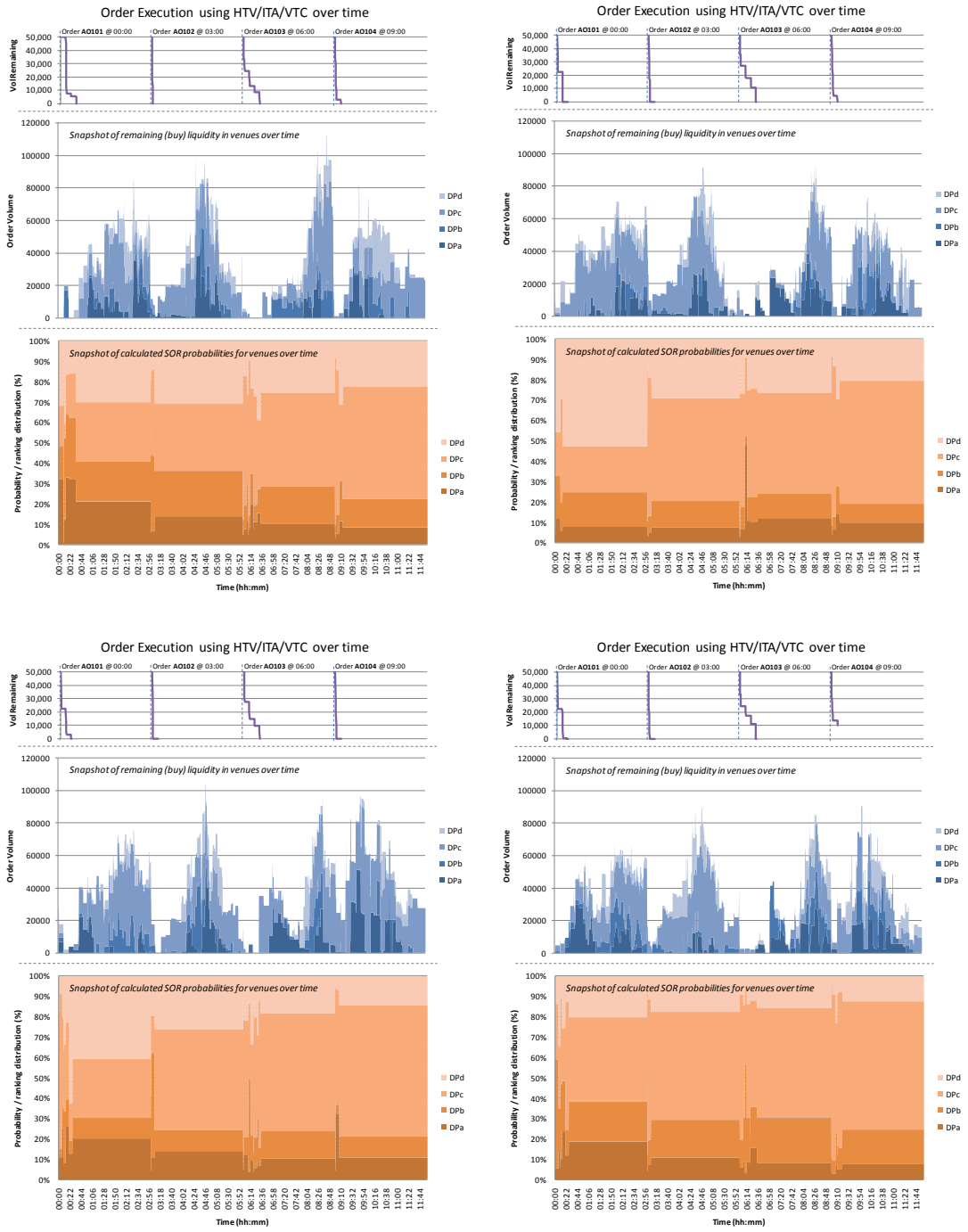
APPENDIX 6.4

Following table shows a breakdown (from a typical test) of all four 50k orders as they were split and routed based on HTV, ITA and VTC as discussed in experiment 6.4.3.

EXECUTED TIME	ORDER	QTY	VENUE	VENUE COST	EXECUTED TIME	ORDER	QTY	VENUE	VENUE COST
ORDER **AO101.0** @ 00:00					**ORDER AO103.0** @ 06:00				
00:00:52	A101.1	DPd	9600	0.3	**06:00:00**	A103.0	DPc	5098	0.5
00:00:52	A101.0	DPa	3153	0.3	**06:00:00**	A103.0	DPc	2918	0.5
00:00:52	A101.0	DPa	2646	0.3	**06:00:39**	A103.0	DPc	2500	0.5
00:00:52	A101.2	DPc	3533	0.5	**06:00:39**	A103.0	DPc	4484	0.5
00:00:52	A101.0	DPa	2666	0.3	**06:00:59**	A103.5	DPc	682	0.5
00:00:59	A101.2	DPc	1667	0.5	**06:00:59**	A103.5	DPc	583	0.5
00:01:39	A101.5	DPc	1667	0.5	**06:10:39**	A103.5	DPc	3735	0.5
00:01:39	A101.5	DPc	2678	0.5	**06:10:59**	A103.6	DPc	1053	0.5
00:10:19	A101.0	DPa	2686	0.3	**06:10:59**	A103.6	DPc	2916	0.5
00:10:39	A101.5	DPc	665	0.5	**06:20:39**	A103.6	DPc	1031	0.5
00:10:39	A101.0	DPa	2136	0.3	**06:20:59**	A103.4	DPc	3757	0.5
00:10:39	A101.1	DPd	400	0.3	**06:30:19**	A103.4	DPc	1243	0.5
00:10:59	A101.6	DPd	4900	0.3	**06:30:35**	A103.3	DPb	5000	0.7
00:10:59	A101.4	DPc	5600	0.5	**06:30:35**	A103.2	DPa	1205	0.3
00:11:39	A101.0	DPd	1003	0.3	**06:30:58**	A103.1	DPc	9499	0.5
00:11:39	A101.3	DPd	3197	0.3	**06:30:58**	A103.1	DPc	501	0.5
00:20:39	A101.3	DPd	1803	0.3	**06:30:58**	A103.2	DPc	3795	0.5
ORDER **AO102.0** @ 03:00					**ORDER AO104.0** @ 09:00				
03:00:17	A102.0	DPd	2737	0.3	**09:00:00**	A104.0	DPc	8827	0.5
03:00:17	A102.3	DPb	288	0.7	**09:00: 00**	A104.0	DPc	6173	0.5
03:00:17	A102.0	DPd	2142	0.3	**09:00: 00**	A104.2	DPa	5000	0.3
03:00:17	A102.1	DPc	1339	0.5	**09:00: 00**	A104.6	DPc	3410	0.5
03:00:17	A102.2	DPa	288	0.3	**09:00: 00**	A104.6	DPc	1590	0.5
03:00:17	A102.3	DPb	668	0.7	**09:00: 01**	A104.5	DPa	2022	0.3
03:00:17	A102.1	DPc	5857	0.5	**09:00:19**	A104.4	DPc	5000	0.5
03:00:19	A102.2	DPa	4712	0.3	**09:00:39**	A104.5	DPc	2978	0.5
03:00:19	A102.3	DPb	2142	0.7	**09:01:19**	A104.3	DPc	17	0.5
03:00:19	A102.1	DPc	2804	0.5	**09:01:19**	A104.3	DPc	4983	0.5
03:00:19	A102.0	DPd	2142	0.3	**09:01:37**	A104.1	DPc	6000	0.5
03:00:39	A102.4	DPc	53	0.5	**09:01:37**	A104.1	DPc	27	0.5
03:00:59	A102.5	DPa	673	0.3	**09:10:37**	A104.1	DPc	3973	0.5
03:00:59	A102.4	DPc	4947	0.5					
03:01:19	A102.6	DPc	910	0.5					
03:01:19	A102.6	DPc	4090	0.5					
03:01:39	A102.5	DPc	1768	0.5					
03:01:39	A102.5	DPc	2559	0.5					
03:02:19	A102.0	DPc	7441	0.5					
03:02:19	A102.0	DPc	538	0.5					
03:02:34	A102.3	DPc	128	0.5					
03:02:34	A102.3	DPc	668	0.5					
03:10:34	A102.3	DPc	1106	0.5					

APPENDIX 6.5

The following presents further sample results from the many repeated tests based on experiment 6.4.3. The bottom graphs (showing snapshots of how SOR calculated venue probabilities over time) present a largely similar pattern throughout the repeated tests in the multi agent simulation.



APPENDIX 6.6

In this appendix, an example from the evaluation chapter is discussed to show the effect on the total distribution due to some changes in SOR routing behaviours. This highlights the workings of the multi agent simulation.

The first experiment (fig 6.4f), the SOR's equal splitting strategy shows the total executed volume across each venue in comparison to the entire volume.

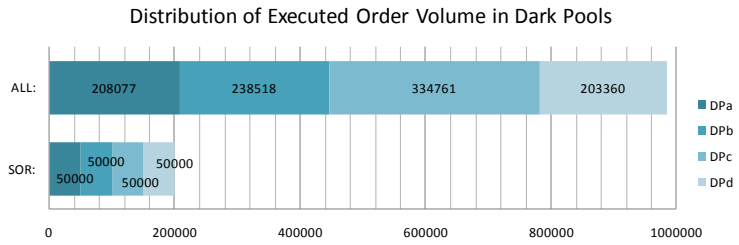


Fig 6.4d: Shows the distribution of executed orders (with SOR equal splitting strategy)

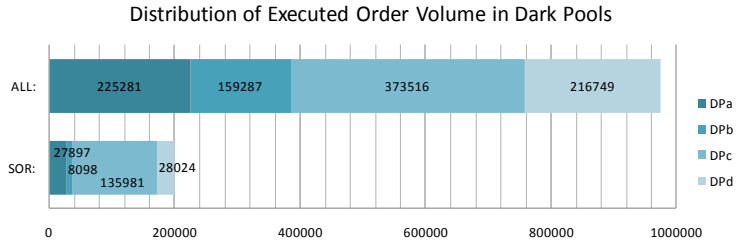
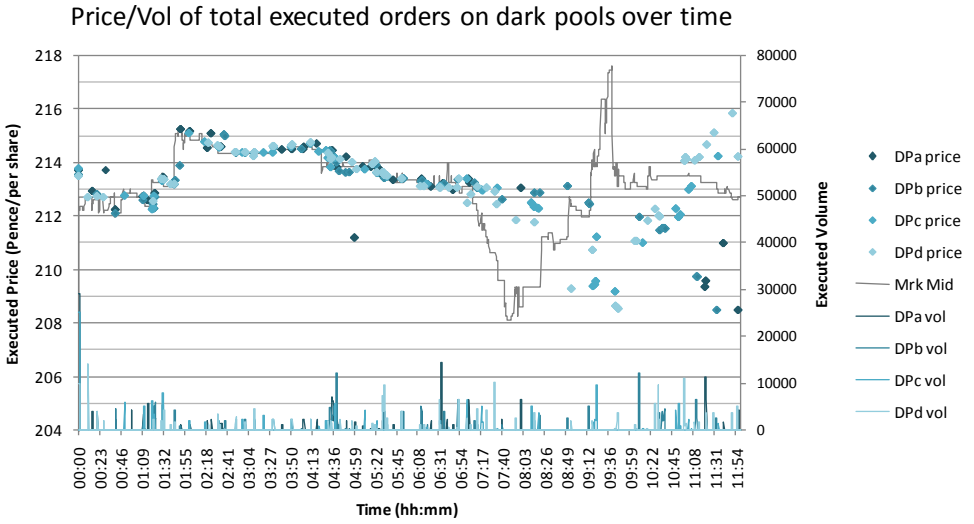


Fig 6.4h: Shows the distribution of executed orders (with SOR HTV/ITA + variable splitting)

In the second experiment (fig 6.4h), with the use of HTV and ITA, SOR was able to detect more volume on DPc and therefore directed more of its order flow there (early on) over time –away from the other venues. The total amount of executed volume at DPb for example (by all parties) in the second experiment was significantly lower, indicating the movement of liquidity as a result of the surge in SOR routings to DPc specifically. This also influenced some of the other trading parties, causing even less order volume to hit DPb; this is shown by the fact that the decrease in executed volume on that venue was much greater than that caused by the SOR router its self. Such revelations and dynamics in the results would not be possible without a multi agent setup.

APPENDIX 6.7

The load on the messaging and server systems due to some of the more demanding experiments was significantly high, affecting the results and outputting inaccurate data. Following are samples showing executions that are significantly outside the market ask/bid spread, all due to high latency:



The simulation speed was lowered to 50x, reducing the number of messages per second flowing, and hence improving the output. However this was still not enough and was only resolved when the speed was lowered down to 30x time factor (for these types of tests).