Simulating The Order Book

A Tool To Discover Trading Strategies

Master Thesis

Department of Computer Science

Erik Eiesland

May 16, 2011

Halden, Norway

Abstract

Keywords: Stock Market, Algorithmic Trading, Machine Learning, Simulation, Order Book

In this paper a system for testing stock market trading strategies involving limit orders are discussed. The system, which consists of an order book simulator and a component for collecting data, is implemented using free and readily available intra-day order book data and trade logs. A order book simulator is a requirement for being able to test trading strategies without the need for a fill model that can probabilistically tell if a given limit order would be filled in the course of the day the order is issued. The main functionality of this order book simulator is to determine when hypothetical orders had been executed, had they been issued to the given order book. This is done by merging these hypothetical orders with real historical data. The data set collected consists of a little more than a hundred days of data for 78 companies listed on NASDAQ OMX Stockholm, collected between November 2010 and May 2011. Due to the nature of this data set, testing of the accuracy of the order book simulator has proven to be quite difficult. Accuracy in this context is the ability to tell when a hypothetical order is filled with the highest level of certainty. Testing of the order book simulator is done by replacing real historical orders with hypothetical ones. If the order book simulator works as intended the replacement orders should be filled at the same time as the historical orders did. Finding the orders to replace has been demanding since the data set does not contain information about when orders get filled. But even so the order book simulator has shown to be accurate for orders happening in times of limited market activity. Further testing needs to be done to be able to tell if the simulator works well for orders issued in times of high market activity.

Table of Contents

Chapter 1

Introduction

Following the advances in communications technology in the last forty years, the way trading is carried out on stock markets has changed dramatically. Stock markets have shifted from being physical locations where buyers and sellers met and negotiated, to public markets traders can access from remote locations to carry out their transactions. The first electronic stock market was NASDAQ, established in 1971. Since the introduction of electronic trading, or e-trading, the number of people working on the exchange floor has declined and today most retail trading is done via the Internet.

The communications technology revolution mentioned above has also resulted in the emergence of algorithmic trading. With algorithmic trading the trades are not planned or executed by human traders, but are instead done by possibly complex algorithms. The computer programs containing these algorithms can do the trading entirely on their own without human intervention. A special kind of algorithmic trading is called high-frequency trading or HFT where computers make decisions and execute orders in the blink of an eye, and can complete a large number of trades before a human trader can process and analyze the information. In 2006 a third of all EU and US stock trades were driven by automatic programs and in 2009 HFT firms accounted for 73% of the US equity trading volume [30].

This thesis will focus on the implementation of an order book simulator where different well-known trading algorithms can be tested and evaluated, and compared to results with algorithms generated by the automatic programming system ADATE. Historical market data for the large cap companies at the Stockholm Stock Exchange (NASDAQ OMX Stockholm) is used in the experiments.

The remainder of this paper is divided into five chapters. Chapter 2 gives an overview of topics related to algorithmic trading and quantitative analysis, as well as an explanation of the methods

used in the implementation of the order book simulator. This implementation is explained in detail in chapter 3. Chapter 4 gives deeper insight into how the order book simulator was tested and presents the results of this testing. Chapter 5 gives ideas on the possible road ahead and future work that can spawn out of this project. The last chapter, chapter 6 wraps up the paper with a short conclusion. Appendices are located at the end of the paper.

Chapter 2

Background

This chapter will give a background on the topics covered in this thesis. The first couple of sections gives a very brief introduction to some fields related to quantitative analysis and algorithmic trading including the efficient market hypothesis. The next sections introduces quantitative trading, some of the common methods and strategies used, and the tools used to measure risk. The chapter ends with examples of related work and an overview of the methods used in the implementation of the order book simulator and in the suggested trading strategies.

It is easy to get the terms "algorithmic trading" and "quantitative trading" mixed up. Hendershott [30] defines algorithmic trading as order execution done entirely by computer algorithms. Chaboud [11] has a similar definition. The definition below is taken from one of the many finance oriented dictionaries found on-line:

> **Algorithmic trading:** A form of automated trading in which computers execute trade orders based on a series of parameters, such as time, price and volume.
> Source: `http://www.investorwords.com/6399/algorithmic_trading.html`

Quantitative trading on the other hand, is trading based on results from quantitative analysis, as seen in the definition below. This implies nothing about who is performing the actual trading, it could be human traders or it could be computers acting on the information gathered from doing quantitative analysis. Quantitative analysis is explained further in section 2.1.3.

> **Quantitative trading:** Investing based on the findings from performing quantitative analysis.

Source: http://www.investorwords.com/8313/quantitative_trading.html

2.1 Related Fields in Finance

This section will give an short introduction to some of the fields in finance that algorithmic and quantitative trading stems from.

2.1.1 Mathematical Finance

Mathematical finance is a field of applied mathematics, concerned with financial markets. According to Jarrow and Protter [34], the French mathematician Louis Bachelier (1870-1946) is seen by many as the founder of modern mathematical finance. He created a model of Brownian motion while deriving the dynamic behavior of the Paris stock market in 1900. Albert Einstein and T. N. Thiele of Copenhagen also created models of Brownian motion around the same time (1905 and 1880 respectively) as Bachelier and it was Einstein's model that immediately became influential. The field of mathematical finance has grown to encompass quite a bit of theory since its beginnings. There are introductory books available for free on-line, such as that of Ross called "An elementary introduction to mathematical finance", a book from from 2003 [48].

2.1.2 Computational Finance

The power of modern computers have transformed the whole of society, including the world of finance. The computational powers of these machines enable analyses on a different scale than previously possible. Computational finance is field of research that encompass this computational intelligence together with mathematical finance, numerical methods and computer simulations [56]. There are many approaches to problem solving in the field of computational finance. One approach is to model entire markets by employing computer programs that model behaviors and players in the market. These computer programs are called agents. Other computer techniques such as artificial neural nets and evolutionary computation are also in use in computational finance.

2.1.3 Quantitative Analysis

Quantitative analysis is concerned with determining the value of securities by examining numerical, measurable characteristics such as revenues, earnings, margins and market share [13]. The findings

of these quantitative analyses can be used in computer algorithms that are designed by quantitative analysts, who may be traders themselves, based on the historical performance of the encoded strategy tested against historical financial data. Quantitative analysis is not the same as technical analysis because all the methods used by technical analysts may not be quantifiable and therefore possible to include in a quantitative trading strategy. Quantitative analysis can use all sort of numeric inputs and fundamental data, and some algorithms even incorporate news events. Many of the quantitative analysts, often called "quants" come from other fields of science such as mathematics and physics. Because of the "quants" background in these fields they often draw from experience in mathematics used in other fields such as statistics and probability, calculus centered around partial differential equations and econometrics. The most commonly used numerical methods are finite difference methods and Monte Carlo methods used to solve the partial differential equations.

2.2 Efficient Market Hypothesis

You cannot read many articles related to algorithmic trading and quantitative analysis before you stumble upon the term "efficient market hypothesis". Thus it seem only natural to explain the fundamentals of the hypothesis. The term "efficient market" was first coined by Fama et al. in 1969 [21]. An efficient market was defined as a market which adjusts rapidly to new information. This was later modified to a more modern definition, which says that asset prices in an efficient market fully reflect all available information. Asset prices should as a consequence always be consistent with fundamental data [6]. The above definition is called the "strong" version of the hypothesis and this version could only be literally true if "all available information" was costless to obtain. If information was instead costly to obtain, there must be a financial incentive to obtain it, but no such incentive would exist if all the information was reflected in the asset prices. A weaker hypothesis has been put forward, which can be seen as more economically realistic. In this hypothesis the prices reflect all information up to a point where the marginal benefits of acting on the information do not exceed the marginal costs of collecting it [35]. There is also an intermediate version of the hypothesis called the "semi-strong" version where information is taken to mean all publicly available information.

Empirical testing into the hypothesis that financial markets are efficient depend on models that provide a link between asset prices and fundamental data, or else one cannot say that prices are consistent with these economic fundamentals [6]. And if the evidence show signs that contradicts the efficient market hypothesis, the problem might not be in the efficient market hypothesis, but

rather in the asset-price model. This is called the "bad model" problem [21].

The efficient market hypothesis makes a number of predictions that can be tested empirically. Going through all the literature and making the case against or for efficient market hypothesis could be several theses in itself. One of the predictions of efficient market hypothesis is that asset prices move as random walks over time. The term "random walk", in financial literature, is a term to characterize a price series where all subsequent price changes represent random departures from previous prices [41]. According to Samuelson [49], asset prices in an efficient market should fluctuate randomly through time in response to the unanticipated component of news, i.e. tomorrow's price changes will reflect only tomorrow's news and will be independent of the price change today. There may be a trend in asset prices that manifests itself over time, so that the total return on a financial asset exceeds the return on a risk-free asset by an amount that is proportionate to the level of risk undertaken in holding the asset. But even so, deviations away from this trend should be unpredictable [6]. On balance, there is evidence that the "random walk hypothesis" is approximately true, at least according to Beechey et al. [6]. The stock returns are somewhat predictable, but the degree of predictability is generally small compared to the high variability of returns. Beechey et al. provides more empirical evidence regarding the efficient market hypothesis, which will not be explained here, interested readers are directed to their well referenced survey [6].

The efficient market hypothesis is important when it comes to algorithmic trading and quantitative analysis (and all other analyses used for prediction of the market). It follows logically that if a strong efficient market hypothesis is true, there should be no point in doing algorithmic trading derived from analysis of the market, and it would be no helpful information to gain by doing quantitative analysis. But at the same time, large financial institutions claim to have made billions of US dollars by employing algorithmic trading schemes[1].

2.3 What is a trading strategy?

According to the many finance oriented dictionaries to be found on-line, a trading strategy is a set of rules and parameters that investors use when making securities transactions[2]. By adhering to these rules the traders can avoid excessive influence by their personal emotions, emotions that can produce hasty decisions.

[1] http://www.opalesque.com/53867/Highfrequencytrading/frequency_under261.html
[2] http://www.investorwords.com/6708/trading_strategy.html

2.4 The Limit Order Book

There are two fundamentally different types of orders that can be issued in a stock market. They are the market order and the limit order. In a market order you only specify the desired volume (e.g. 500 shares) and the the order is filled immediately after it is issued. The order is filled with the most competitive orders on the opposing book. If we use the snapshot of the ERIC-B order book in figure 2.1 as the basis for a hypothetical buy market order for 40 000 shares, the result would be as follows. This order would be filled by the ten sell orders totaling 30 112 shares at 71.55 SEK and 9 888 of the 61 257 shares selling at 71.60 SEK. The remaining 51 369 shares at 71.60 SEK would remain in the order book as the new top of the sell side of the book, i.e. 71.60 SEK would be the new asking price.

The limit order works a little differently. In a limit order the desired price is part of the order in addition to the desired volume. This means that instead of getting executed immediately it is placed in the order book queue. The order book is sorted with the highest unexecuted buy price at the the top and the topmost buy order is often referred to as the bid price. On the opposing side of the book the sell prices are listed with the lowest unexecuted sell price on top and here the topmost order is called the ask price. The bid price and sell price are sometimes referred to as the inside market, and the difference between them as the spread [37]. It is often these prices, the bid and ask price, that are mentioned when stock prices are discussed in the media and other fora, together with the price of the last traded stock. If we take another look at figure 2.1 and place a hypothetical limit order to buy 1 500 shares of the ERIC-B stock at 71.20 SEK it would be placed on level six of the order book, the number of orders on this level would rise from four to five and the bid depth (total number of shares) would increase from 22 745 to 24245. This order would be executed if the price moves in this direction, if it does not the order will remain unexecuted. The order can also be withdrawn or changed before it is executed. Within each price level orders are sorted by time of arrival and they are served on a first in, first served basis. It is impossible for two or more orders to arrive at the same time.

A limit buy order with a price much higher than the current ask price, or a limit sell order with a much lower price than the current bid is effectively a market order and will get executed right away. Because of this every order can be viewed as a limit order with the important distinction that limit orders are guaranteed price (if executed), but not execution, while a market order is guaranteed execution, but not price [38].

The order book mechanisms described above has been around more or less unchanged since

Table 2.1: Tick sizes for NASDAQ OMX Stockholm 30 and Large Cap in SEK. Source: NASDAQ OMX Market Model [43]
.

From	To	Tick Size
0.0000	0.4999	0.0001
0.5000	0.9995	0.0005
1.0000	4.9990	0.001
5.0000	9.9950	0.005
10.0000	49.990	0.01
50.0000	99.9500	0.05
100.0000	499.9000	0.1
500.0000	999.5000	0.5
1,000.0000	4,999.0000	1.00
5,000.0000	9,995.0000	5.00
10,000.0000	49,990.0000	10.00
50,000.0000	-	50.00

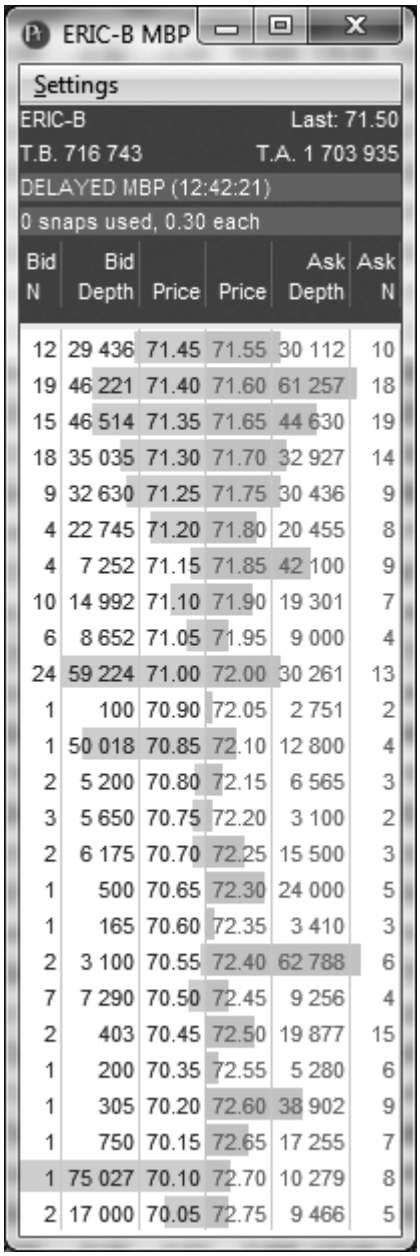
the dawn of financial markets. The automation and computerizing of the process is what makes electronic stock trading using algorithms possible, and this is a more recent change. The general publics opportunity to view these order books is an even more recent event [38].

The limit orders issued by traders or algorithms must have a limit price corresponding to a system of price intervals called “ticks”. Each stock exchange has its own rules with regards to tick sizes. The rules currently in force on NASDAQ OMX Stockholm can be seen in table 2.1. If a limit order is submitted with a price that does not correspond with these tick sizes it is automatically rounded to the nearest tick when added to the order book.

2.5 Strategies and Methods used in Quantitative Trading

There are a lot of different trading strategies that have been proposed or implemented since electronic trading first begun, and they are evolving all the time. The strategies discussed below may not be used exclusively in algorithmic trading, or even in electronic trading, but they can use the fast decision making of algorithms to their advantage. The strategies are also not limited to strategies that use the information of the open order book as part of their decision making foundation. The methods used as part of these strategies may not be exclusive to the particular strategy in question, they may be common with other strategies and they may hail from very different scientific fields. The

Figure 2.1: A snapshot of the order book of the ERIC-B stock at 12:42:21 from PrimeTrader.



ERIC-B MBP

Settings

ERIC-B Last: 71.50

T.B. 716 743 T.A. 1 703 935

DELAYED MBP (12:42:21)

0 snaps used, 0.30 each

Bid N	Bid Depth	Price	Price	Ask Depth	Ask N
12	29 436	71.45	71.55	30 112	10
19	46 221	71.40	71.60	61 257	18
15	46 514	71.35	71.65	44 630	19
18	35 035	71.30	71.70	32 927	14
9	32 630	71.25	71.75	30 436	9
4	22 745	71.20	71.80	20 455	8
4	7 252	71.15	71.85	42 100	9
10	14 992	71.10	71.90	19 301	7
6	8 652	71.05	71.95	9 000	4
24	59 224	71.00	72.00	30 261	13
1	100	70.90	72.05	2 751	2
1	50 018	70.85	72.10	12 800	4
2	5 200	70.80	72.15	6 565	3
3	5 650	70.75	72.20	3 100	2
2	6 175	70.70	72.25	15 500	3
1	500	70.65	72.30	24 000	5
1	165	70.60	72.35	3 410	3
2	3 100	70.55	72.40	62 788	6
7	7 290	70.50	72.45	9 256	4
2	403	70.45	72.50	19 877	15
1	200	70.35	72.55	5 280	6
1	305	70.20	72.60	38 902	9
1	750	70.15	72.65	17 255	7
1	75 027	70.10	72.70	10 279	8
2	17 000	70.05	72.75	9 466	5

quantitative trading strategies can be separated into strategies that look for correlations among liquid assets, such as arbitrage and pairs trading, and strategies that follow patterns in price-movement such as trend following and mean reversion.

2.5.1 Volume-Weighted Average Price

According to a paper from 2004 called "Competitive Algorithms for VWAP and Limit Order Trading" by Kakade, Kearns, Mansour and Ortiz [37], Volume-Weighted Average Price or VWAP-trading is one of the most common activities in modern financial markets. The VWAP of a stock can simply be explained as the average price paid per share during a specified time, usually a day. This means that the price of each transaction in the market is weighted by its volume. In VWAP-trading the goal is to buy or sell a fixed number of shares at price that closely tracks the VWAP. VWAP is especially common in automatic trading algorithms, especially in optimal trading execution strategies [15]. The formula for calculating VWAP is as follows:

$$P_{VWAP} = \frac{\sum_j P_j \cdot Q_j}{\sum_j Q_j} \tag{2.1}$$

where:

$P_{VWAP} =$ Volume-Weighted Average Price
$P_j =$ price of trade j
$Q_j =$ quantity of trade j
$j =$ each individual trade that takes place over the defined period of time, excluding cross trades[3] and basket cross trades.

2.5.2 Arbitrage

There are a lot of trading strategies that use various forms of arbitrage as input e.g. the pairs trading scheme mentioned below. Arbitrage is defined as a activity that profits by exploiting price differences of identical or similar financial instruments, on different markets or different forms[4]. Arbitrage exists as a result of market inefficiencies (see section 2.2), and it provides a mechanism to ensure that prices do not deviate substantially from fair value for long periods of time. There are

[3] A transaction that is not exposed to the public by outcry or usual trading practices - `http://www.bizterms.net/term/X-or-Cross-Trade.html`

[4] Definition found at: `http://www.investorwords.com/245/arbitrage.html`

various forms of arbitrage. The short-term mispricing that is exploited in pairs trading is often called statistical arbitrage. Time series, and methods applied to these time series, are employed to find this statistical arbitrage. Another form of arbitrage called risk arbitrage, on the other hand, relates to strategies involving stocks of merging companies [18]. For this reason risk arbitrage is sometimes called merger arbitrage. Another type of arbitrage is called event arbitrage. Event arbitrage relates to a specific event, such as a contract signing, regulatory approval or judicial decision etc. that can be taken advantage of by the traders.

2.5.3 Pairs Trading

According to E. Gatev et al. [22] pairs trading is a popular and *disarmingly* simple trading strategy with a 20-year history on Wall Street (2006) and it is used by e.g. hedge funds and investment banks and also by independent traders. The idea of the strategy is to find two stocks whose prices have moved together historically. If there is a widening of the spread between them, short sell the winner and buy the loser. If history repeats itself, prices will converge and a profit can be made. Pairs trading is a strategy in the family of investing approaches that is based on going long (buying) on what are perceived as under-priced assets and going short (short selling) assets that are seen as over-priced [18].There are many different methods used to single out these pairs, finding out when they diverge enough to establish a position, and when they converge again and it is time to unwind the position.

The pairs trade strategy fall into the category of strategies using statistical arbitrage [22]. This strategy, like most other trading schemes, should not be able to provide risk-adjusted returns under the predictions of strong efficient market hypothesis (see section 2.2).

Do et al. [18] categorizes pairs trading implementations into three main methods, which they call the distance method, the co-integration method and the stochastic spread method. The distance methods is discussed by Gatev et al. [22] and Nath [44], the co-integration is discussed by Vidyamurthy [59] and the stochastic spread approach is discussed by Elliot et al. [19].

In Gatev et al. [22] a pairs trading strategy is tested on all the stocks in the CRSP[5] stock database daily files that were considered liquid. Liquidity was assumed for stocks that have been traded every day in the testing period. The testing was done in two stages. In the first 12-month stage the pairs where formed and in the last 6-months the trading took place. The pairs formation period involved calculating and creating a cumulative total returns index for each stock based on the data from the

[5]Center for Research in Security Prices, a provider of historical stock market data and part of the Booth School of Business at the University of Chicago. URL: `http://www.crsp.com/`

12-month period. They then found a "matching partner" for each stock by finding the security that minimized the sum of squared deviations between the two normalized price series, minimizing what is otherwise known as the distance between price series [18]. This approach to choosing pairs was favored because it has, according to Gatev et al., similarities with how traders themselves chose pairs, looking for stocks whose prices "move together". Pairs restricted to certain industry sectors, as defined by S&P 500, were also part of the experiments.

The top 5 and 20 pairs found in the liquid stocks were studied further, as well as the 20 pairs following the top 100 pairs. These 20 pairs were selected because they did not share some characteristics shared by the top 20 pairs. Figure 2.2, taken from Gatev et al., shows one of the pairs in their experiments and illustrates how positions were opened and closed following the divergences and convergences of the normalized price paths. Both stocks are "leaders" at some points in time. The rules for opening and closing a position were based on a standard deviation metric. A long-short position would be taken in a pair when the prices diverged more than two historical standard deviations, as estimated during the pairs formation. The position would be closed when the prices cross each other. If prices would not cross the position were closed on the last day of the trading period.

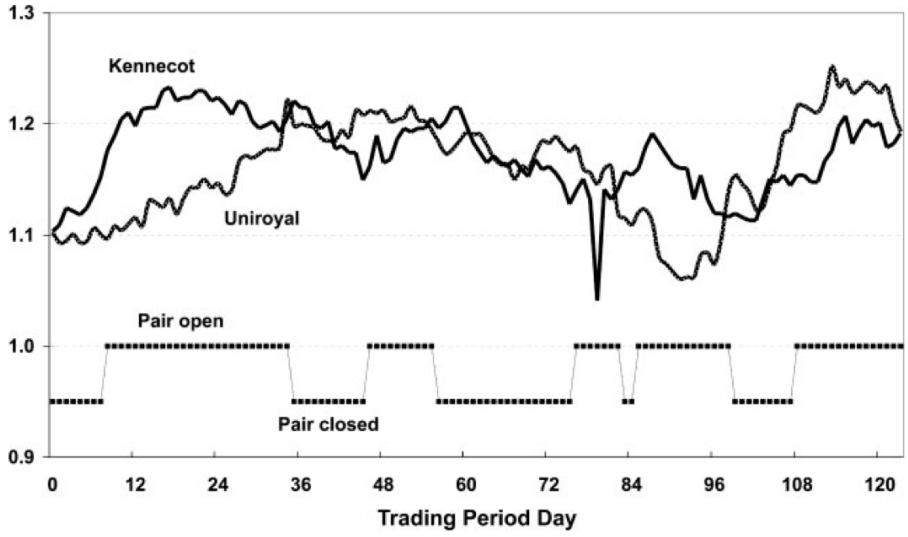
Trading gains and losses in Gatev et al. were computed over long-short positions of one dollar ($1). The excess return on a pair during a trading interval of 6 months were computed as the reinvested payoffs during the trading interval. Experiments were run with positions being opened and closed at the end of the day. The fully invested portfolio of the five best pairs earned an average excess monthly return of 1.31%, and the portfolio of the 20 best pairs 1.44% per month. The results in Gatev et al. are extensive and contain many statistical measurements, risk-adjustments and results for different industrial sectors which can be seen in a large number of tables in Gatev et al. [22].

Vidyamurthy [59] utilized a phenomenon known as co-integration introduced by Engle and Granger in 1987[20] for the pair formation. Co-integration is the phenomenon that two time series that are both integrated of order d, can be linearly combined to produce a single time series that is integrated of order $d - b, b > 0$, the most simple case of which is when $d = b = 1$. This combined time series is stationary, i.e. the joint probability distribution does not change when the time series is shifted in time or space [18].

2.5.4 Trend Following

According to Silaghi and Robu [53], trend follwing is a "classical" strategy, which only use information from price movements. These strategies tries to take advantage of long-term moves, both

Figure 2.2: Daily normalized prices for Kennecott and Uniroyal between August 1963 and January 1964. Source: Gatev et al. [22].



from periods of prosperity and periods of decline, that takes place in the market. Seeing how these trends typically manifests themselves over longer time periods, they may be less interesting from an algorithmic trading point of view, an most certainly for intra-day trading.

2.5.5 Mean Reversion

Mean reversion is a mathematical concept that is sometimes used in stock investing and algorithmic trading. The essence of mean reversion is the assumption that a stock's high and low prices are temporary and that a stock's price will tend to move to the average price over time. If stock prices follow a mean reverting process, and there exists a tendency for the price level to return to its trend path, then investors can use the information of past returns to predict future returns [14]. If the stock prices instead followed a random walk pattern, it would not be possible to make such predictions. The pairs trading strategy discussed above depends on the concept of mean reversion.

2.5.6 Trade Execution Strategies

Trade execution is the process of trading a particular instrument of interest [15]. A central, practical issue in trade execution strategies is how to efficiently trade large orders, compared to e.g. the average daily volume. The objective is to carry out the execution without running the risk of a unnecessary increased market impact cost, which is probable since trading large volumes has the

potential to change asset prices. A standard approach to minimizing the total trade cost is to divide the large order into smaller child orders and trade them over time. This is not without risk however, opportunities may be missed during the course of the trading execution. Trade execution strategies seeks to balance between this risk and the total trade cost [15]. The trade execution strategies are related to transaction cost reduction mentioned above.

2.5.7 Market Neutral Strategies

Market neutral strategies are strategies that aims to achieve negligible exposure to systematic risks, while at same time capitalizing on the active returns from the long position on the winners and the short position on the losers [18]. The pairs trading strategy discussed above has sometimes been categorized as a market neutral trading strategy [31].

2.5.8 Market Making Strategies

According to Hendershott [29], not all algorithmic trading is done by institutional buy-side investors, making profit directly, as many observers think. There are other uses for algorithms, such as market making. These algorithms could indirectly influence algorithms whose main purpose is making a profit. A definition of market making can be seen in appendix A.

2.5.9 Static Order Book Imbalance

Static Order Book Imbalance or SOBI is a stock trading strategy that uses Volume Weighted Average Price (VWAP) [38]. An explanation of VWAP can be seen in section 2.5.1. This strategy use the information content of the open order books and computes VWAP for sections of the buy and order books and compares the result with the last price. The strategy asserts that if the VWAP of the buy book is much further from the last price than the VWAP of the sell book, it is a sign that buyers are less supportive of this price than are sellers, as indicated by their limit orders (statistically) standing further off. Should this occur the SOBI strategy will place an order to sell shares, on the theory that the weaker buy-side support will cause the price to fall. The strategy can be adjusted by tuning different variables such as how large a section of the order books to use for VWAP calculations, how strong the VWAP imbalance must be in order to trade, the volume of shares to trade, whether to place market or limit orders and at what price. The SOBI strategy is an example of a strategy that uses and interprets order book data as an expression of market sentiment.

2.5.10 High-frequency Trading

High-frequency trading is a type of investment strategy whereby profits are attempted to be made by rapidly buying and selling stocks, with typical holding periods in terms of seconds or milliseconds. These strategies typically try to take advantage of different kinds of arbitrage that exists for very brief periods of time. High-frequency trading and algorithmic trading both use computers to make fast decisions, but algorithmic trading algorithms may incorporate holding periods that are much longer than that of high-frequency trading, i.e. high-frequency trading can be categorized as a type of algorithmic trading, but algorithmic trading does not have to be high-frequency trading [8].

2.5.11 Low-latency Trading

Low-latency trading is a special class of high-frequency trading that relies on the ultra-low-latency direct market access technology, that enables traders to communicate with the market at very high speeds. Strategies involved in low-latency trading respond do market events in milliseconds, which compared to human traders can be seen as almost instantaneously and thus these strategies can take advantage of imbalances that are present for extremely short periods of time, even shorter periods than high-frequency trading [28].

2.5.12 Markets

Algorithmic and trading based on quantitative analysis is popular not only in stock market trading, including different kinds of securities, but also the foreign exchange market and other markets. Chaboud's book on algorithmic trading focuses on trading in the foreign exchange market [11] and Neely et al. [45] has found profitable technical trading rules using genetic algorithms in the same market. Methods exclusive to these other markets are not in the scope of this paper.

2.6 Measuring Performance and Managing Risk

Handling risk is an important factor when employing quantitative trading schemes and risk should be an integral part of performance measurements. Managing risk is also a big field of research in finance in general. Measuring how an quantitative trading scheme's performance would be in the real world is often a non-trivial issue. A lot of factors need to be considered in order to make good performance estimations, not only risk factors, but also e.g. accurate backtesting (see section 2.9.1) etc.

2.6.1 Risk-adjusted Return

The risk-adjusted return is a measure of how much an investment returned in relation to the amount of risk it took on. Risk-adjusted return is often used to compare a high-risk, potentially high-return investment with a low-risk, lower-return investment [42].

2.6.2 Sharpe Ratio

According to Modigliani [42], the risk-adjusted return measure "Sharpe ratio" is the most common measure of risk-adjusted return in use today. The Sharpe ratio is defined as follows:

$$S = \frac{R - R_f}{\sigma} = \frac{E[R - R_f]}{\sqrt{var[R - R_f]}} \tag{2.2}$$

This risk-adjusted measure was developed by William F. Sharpe [52], and it is calculated using standard deviation and excess return to determine reward per unit of risk. The higher the Sharpe ratio, the better the fund's historical risk-adjusted performance.

2.7 Related Work

This section will give an outline of some of the related academic work on using machine learning in stock trading. The section about the Penn-Lehman project describe a system with similarities with the order book simulator discussed in this paper. It has been difficult to find good (more than just a few percent return) articles in the field of machine learning in relation to securities trading. The absence of published research showing good results in this field is no surprise however. Publishing hugely profitable trading strategies and the methods used to acquire them would most certainly make them obsolete and as a consequence unable to make their creators rich. The above is especially true for strategies involving limit orders and the order book. Another point to be made is that the performance calculations vary considerably, both in terms of model selection and factors included (such as e.g. interest, risks and brokerage fees). This makes it difficult to compare the results.

Using Genetic Algorithms To Find Technical Trading Rules

Allen and Karjalainen [1] wrote a paper in 1993 about their attempt at finding technical trading rules by using a genetic algorithm. They used data from the Standard and Poor's Composite Stock Index which contained data from 1963 to 1989. With their algorithm they could find trading rules that

gave them excess returns in the out of sample period 1970 to 1989. This was with transaction costs included. The results were compared to benchmark models of a random walk, an autoregressive model and a GARCH-AR model. In the 100 trials they ran, 82 different rules were found. Trading with these rules in the 20 year test period would give an excess return of 4.51% above the annual buy-and-hold return of 6.72%. This is including the stock market crash of October 1987.

Evolving Short-term Trading Strategies Using Genetic Programming

In a paper from 2002, Svangård, Nordin, Lloyd and Wihlborg [54] discuss using genetic programming to find short-term trading strategies. They used their GP-engine (Genetic Programming Engine) to analyze financial time series at one minute intervals for the purpose of discovering a successful trading strategy for a single stock, which in this case was Nokia. A trading strategy was deemed successful if it could outperform a simple buy-and-hold strategy. The data used in the experiments were collected between April 5$^{\text{th}}$ and May 9$^{\text{th}}$ in 2001 and contained values from 55 financial instruments including the ask and bid price of the Nokia stock. The data used in the training, validation and out of sample data (test data) were normalized to values between 0 and 1. All the values, except the time value were converted into relative values using the following equation:

$$v_t/v_{t-1} - 1 \tag{2.3}$$

A positive observation in $t + 1$ indicated an increase in value, while a negative observation indicated a decrease in value. An observation of exactly zero meant no change. The input data also contained two feedback variables. The first was the previous output, making the agent aware of its actions one minute ago. The second variable showed the relative performance of the share price since the last transaction was executed. A relatively strict set of conditions was put in place to guide the trading agent's actions. The agent could reinvest the proceeds from successful transactions, but only blocks of shares could be purchased. Interest earned for inactive cash in the account was not taken into account for simplicity. The agent had to buy at the ask price and sell at the bid price. The agent's fitness was based on its ability to earn money and was defined by a simple formula:

$$v_0/v_n \tag{2.4}$$

v_n was the value of the portfolio at the end of the dataset, and v_0 was the initial value of the portfolio, i.e. a fitness below 1.0 indicated a surplus return while a fitness higher than 1.0 meant that money was lost.

The results were relatively good in that the agent gave excess returns compared to a buy-and-hold strategy in all three sections of the data set. In the training set the agent had a risk-adjusted return of 3.06% while the buy-and-hold had 2.4%. In the validation data the results were 2.13% and 1.57% for the agent and the buy-and-hold strategy respectively. The numbers for out of sample data were 1.89% and -2.93%.

The Penn-Lehman Automated Trading Project

In a paper from 2005 by Kearns and Ortiz, the Penn-Lehman Automated Trading project [38] is described. This project was centered around the Penn Exchange Simulator (PXS) as a tool to find algorithms and strategies for automated trading in financial markets. PXS is a software simulator for automated stock trading that merges automated client limit orders with real-world, real-time order data that is available via modern Electronic Crossing Networks (ECN). The Penn-Lehman Project is currently inactive, but it may be continued in the future, at least that is what its homepage says[6]. Lehman of Penn-Lehman refers to Lehman Brothers who famously declared bankruptcy in 2008.

The PXS uses data from Island's live web-based order book viewer. This order book contains the top 15 limit orders with price and volume for buy and sell orders. Island (`http:\www.island.com`) does not exist anymore, in its old form, as it was aquired by Instinet, rebranded to Inet and later aquired by NASDAQ in 2005. The PSX simulation can be run in either live mode or historical mode. In live mode the order books are updated from Island order books every 3 seconds. If historical data from Island is available and archived for a specific day, historical mode may be used. Running historical simulations is said to be considerably faster since it does not have to wait for live data and therefore works by its internal processing rate.

The PXS has an API for clients (trading strategies) that communicate with the simulator. The API's primary functions are buyOrder(p , v), sellOrder (p, v) and withdraw(o). The first two functions take price and volume as parameters and they both return an unique identifier (o) that can be used to withdraw the order if it is not executed already. In addition the simulator has a plethora of other functions that provide information about the status of the simulator and the clients. This data corresponds to data available from real-world ECNs.

The central component of the PXS is the execution engine. This component is responsible for maintaining the order books, integrating real-world orders from Island and client limit orders into these books, for executing matching orders of PXS clients, and for computing share position, cash holdings, and profit and loss of PSX clients. In any PXS simulation, all PXS clients starts with

[6] `http://www.cis.upenn.edu/~mkearns/projects/plat.html`

no cash and no shares of the stock being traded. PXS clients are allowed to buy or sell shares at any time during the simulation regardless of their current cash and share holdings. This, in turn, means that clients can sell shares they do not hold (selling short) or buy shares without cash (buying on a margin), and therefore cash balance and share holdings can be negative or positive. A client's performance or valuation, at any time, is defined as the sum of its cash balance, and its share position times the current (last) price of the stock. Here a liquidity assumption is made and this is potentially unrealistic. Returning a client's share holdings to zero by trading in all held shares on the market and receive the current price for all of them is perhaps possible for a small amount of shares, but large share transactions will move the price unfavorably.

The PXS simulator was used in the Penn-Lehman Automated Trading Project (PLAT project). In addition to the simulator the PLAT project held competitions with around 30 participants ordered in groups. The competed in creating trading strategies as clients in the PXS simulator. This type of competition has proven itself to be scientific and sociologically successful, delivering results in both areas. All competitions were executed on Microsoft stock (MSFT). This made things a lot simpler because of less data management and more comparable results between groups. PLAT worked exclusively on intraday trading and share position could never exceed $\pm$ 100,000 shares. The latter was done to prevent the competition from being dominated by strategies that simply place large bets in the form of excessive share positions.

The main competition was held for each of the 10 trading days of the weeks of April 28$^{\text{th}}$ and May 5$^{\text{th}}$ of 2003. Each day the simulation ran from 9:30 AM to 4:00 PM, the normal trading hours of the NASDAQ exchange. 14 different strategies were included in the competition, divided into the blue and the red pool. The overall profitability was the main criteria for evaluating the clients, but to further encourage client realism and good trading practices a rich set of scoring criteria were used. These criteria were daily profit and loss, overall consistency profit and loss to reward strategies with cumulative profitable performance over several days, daily intraday position reversals to reward strategies that do well at both buying and selling stock, robustness to market variation to reward strategies that perform well in both bull and bear markets, daily risk saturation to keep clients trading close to the 100,000 shares mark and daily position unwinding to encourage clients to cash out their positions during the trading day with activity in the PXS market and thereby obviate the liquidity assumption. The risk saturation criteria demanded that strategies should hold a position of at least 50,000 shares on a daily basis, either in a long or short position, this to avoid that strategies overreacted to the 100,000 share limit.

The best strategies according to the rules described earlier were a client called CBR-SOBI and

DAMM-STAT. These strategies were quite different with the former amongst the top performers in terms of profit and with the latter barely making a profit, but earning points amongst the other criteria for success. DAMM-STAT made the maximum points for adhering to the principles of risk saturation and for unwinding its position in the market in the last days of trading. Many of the lower ranked strategies had positive earnings, but failed to follow basic practices and behaviors that were encouraged. 11 of the 14 clients ended up with a positive return following the 10 days of execution. CBR-SOBI gave an average return of $ 4187, $\pm$ $ 3733 with a 95 % confidence interval. A more detailed view of the results can be seen in figure 5 in reference [38]. This table includes the results for all clients with points for the various categories in addition profit and loss.

Evolving efficient limit order strategy using Grammatical Evolution

In this paper from 2010 by Cui et al. [15] the authors presents a system for optimal trade execution (see section 2.5.6) using grammatical evolution and VWAP as the benchmark. The experiments were done using a totally artificial market. Grammatical Evolution (GE) is an evolutionary automatic programming technique which allows for the generation of computer programs in a arbitrary language.

In the experiments explained in the article, a large order of 10% of average daily volume (ADV) of the artificial market is considered. It is stated that an order of between 5% and 15% of ADV should be possible to trade during one day without making a substantial market impact. The timespan the order should be traded over was 5 hours.

The experiments were comprised of a training and a testing period. In the training period, GE was used to evolve the trading execution strategies. Each individual was exposed to 20 trading days in the artificial market. The fitness (the optimality of each individual) was calculated as their average VWAP ratio over the 20 trading days. The experiments were run for 40 generations with variable length using one-point crossover at a probability of 0.9, one-point bit mutation at a probability of 0.01, roulette selection, steady-state replacement and a population size of 100. The best evolved strategy is tested out of sample over 240 days in the artificial market.

2.8 Commercial Products

There are a lot of companies around the world that offer complete electronic trading platforms with direct market access (henceforth DMA) and portfolios of algorithms for automated electronic trading/algorithmic trading. By providing DMA to the stock exchanges of the world, traders can

interact with the order book of an stock exchange electronically and it is this technology that makes algorithmic trading possible. Ultra-Low Latency Direct Market Access (ULLDMA) is the market gateway used in high-frequency trading (HFT). These systems can handle huge volumes and have an extremely low delay, which is typically below 500 microseconds[7]. The Swedish financial group SEB and the brokerage firm Neonet are examples of companies that has a portfolio of algorithmic trading products. The various algorithms and trading strategies these companies provide can have very exotic names like "Sniper", "Guerrilla" and "Iceberg", but the logic behind them can be quite simple.

One of the most common strategies in DMA is to cut large orders into smaller pieces [55]. This can be done to hide large orders from the market participants, a technique called "iceberging". Slicing an order into smaller pieces can also be done with the intention to minimize market impact. The "Guerrilla" algorithm developed by Credit Suisse tries to determine which transactions can be done without causing noticeable jumps in a stock's trading pattern and this is an example of a strategy developed to minimize market impact [55]. "Benchmark" algorithms are used to achieve a specific benchmark such as the volume weighted average price (VWAP) over a certain time period.

2.9 Methods

This section will give a background and short explanation of the different methods used in the implementation of the order book simulator and in the suggested trading strategies.

2.9.1 Simulation

Simulation is a powerful tool used for the evaluation and analysis of systems and processes taking place in the real world [9] and this tool is widely used in finance. There exists a number of approaches to simulation. This section will focus on the approach called discrete-event simulation, since this is the approach taken in the order book simulation discussed in chapter 3. A simulation consists of a model, or models, representing a system or process. This model incorporates time and the changes that occur over time. In a discrete-event simulation the model is also discrete, meaning that its state only changes at discrete points in time and not continuously, as opposed to continuous simulation. Discrete-event simulation can be further divided into synchronous and asynchronous models. In synchronous-time models, the time advances by fixed increments such as

[7] http://en.wikipedia.org/wiki/Direct_Market_Access

an hour or a day. In contrast, asynchronous-time models moves from one event to the next, typically by uneven increments [32]. The order book simulator (OBS) uses the latter model, moving from change to change in the order book, based on the historical data. Discrete-event simulation using an asynchronous-time model have not seen the same popularity in financial analysis as the continuous-time model [32] [33].

When measuring trading strategy performance by applying them to historical data, the simulation is often, in finance, called backtesting [38] [12]. Even though backtesting is commonplace when measuring performance of trading strategies, it can not provide with a 100% certainty whether a trading strategy is sound or not. This is because markets change and past results does not necessarily predict future results, especially considering the efficient market hypothesis (see section 2.2). Strategies involving large transactions would also require some sort of market impact model, since transactions with large volumes can move asset prices unfavorably, i.e. the market movements in the historical data might not look the same if the strategies backtested were actually applied in the timespan of the historical data.

There are a lot of security trading games to be found on the Internet, where traders can test their trading strategies in a sandbox environment using "play money". These game type simulators can be based on real, but usually delayed, market data or they can be purely fictional, but neither type involves real financial risk. An example of such a trading game is Investopedia's Stock Simulator found here: `http://simulator.investopedia.com/`. Another example is StockTrak Global Portfolio Simulations found here: `http://www.stocktrak.com/`.

2.9.2 Dynamic Programming

Dynamic programming is a method used for solving complex problems by breaking them down into simpler subproblems [2]. If a problem has a polynomial number of subproblems, like the subset sum problem or the knapsack problem, one would have to solve the same subproblem many times. An alternative would be to keep track of the solutions to these subproblems and simply look up the answer when needed. If this approach is taken one would end up with an algorithm running in polynomial time. The filling-in of a table with subproblems and their respective solutions to get the answer to the parent problem is the core of dynamic programming [2].

2.9.3 Local Search

Local search (LS) is the name given for a family of algorithms used to solve hard combinatorial optimization problems [36]. LS algorithms have proven themselves especially effective for the traveling salesman problem (TSP). TSP involves finding a minimal cost tour through a network of cities with distances between them, visiting each city once [61]. LS algorithms have been used successfully in other areas as well, such as job shop scheduling [57] and for the subset sum problem [23], which is the problem a LS algorithm is used for in the simulator as explained in section 3.2.9.

A local search algorithm is according to A.V. Aho et al. [4] a variation on the following procedure:

1. Start with a random solution
2. Apply to the current solution a transformation from given set of transformations to improve the solution. The improvement becomes the new "current" solution.
3. Repeat until no transformation in the set improves the current solution.

The resulting solution may or may not be optimal, i.e. the algorithm may have been "lucky" and stumbled upon a globally optimal solution or it has found a locally optimal solution. When finding a globally optimal solution takes exponential time, like TSP, and you have to test all possible transformations to find the global optimum, applying a local search algorithm, finding a local optimum, may be practical.

The optimization methods tabu search [24] [27] [25] [26] and simulated annealing [39] [10] are methods related to, or stemming from, the local search family of methods. Tabu search is attributed to the American Fred W. Glover. Local search algorithms, especially those that are called greedy algorithms[8], run the risk of getting trapped in local optima in case of multi-modal[9] objective functions [51]. The rationale behind tabu search is to guide other methods to escape the trap of local optimality. In tabu search, a number of moves (usually the most recent moves) are remembered as the tabu list, and this list can in turn can be used to direct the search. By remembering "bad moves" and marking them as taboo (tabu) the algorithm avoids visiting these solutions again.

Simulated annealing, unlike tabu search, is inspired by processes occurring in the natural world. The term "annealing" is used in metallurgy to describe a technique involving the heating, and controlled cooling of a material to increase the size of its crystals and reduce their defects. The length

[8] An algorithm that follows the problem solving heuristic of making the locally optimal choice at each stage, with the hope of finding the global optimum [3].

[9] Optimization problems with multiple good solutions.

of the cool-down period will determine how long time the atoms or molecules in the material have to find positions in an ordered pattern like a crystal structure. The highest order, which corresponds to a global minimum of the free energy, can be achieved only when the cooling proceeds slow enough. Alternatively the frozen status will be characterized by some number of local energy minimums [51]. Simulated annealing use the analogy of the cooling material to construct a mathematical model for optimization. The ability to escape from inferior local optima, by allowing deteriorations with a certain probability, is the main feature of simulated annealing. According to Schwefel this kind of "forgetting principle" is uncommon in most numerical optimization approaches, except evolutionary algorithms (see section 2.9.4), which has a similar principle built in.

2.9.4 Evolution Strategy

Evolution strategy (ES) is a numerical optimization technique influenced by biological structures and processes [50]. ES is under the umbrella of evolutionary algorithms (EA), together with contemporaries evolutionary programming (EP) and genetic algorithms (GA). All these paradigms try to benefit by mimicking phenomena seen in "adaptive populations of problem solvers" such as birth and death, variation and selection, in an generational loop [7].

According to Schwefel, the task of mimicking biological structures and processes with the object of solving a technical problem is as old as engineering itself, and mimicry itself is even older than mankind. The mimicry of biology in engineering first saw scientific applications in the field of bionics, which is as portmanteau[10] of biology and electronics. In bionics one tries to emulate solutions seen in biology as solutions to problems in engineering, this based on the assumption that nature, through evolution, has already come up with good or even optimal solutions [50].

Rechenberg [47] suggested a two-membered evolution scheme, which can be used to solve non-discrete, non-stochastic, parameter optimization problems. The evolution scheme is said to be "two-membered" because each population contains exactly two individuals. The evolution scheme imitates organic evolution and use the principles of mutation and selection, which Charles Darwin [16] recognized to be the most important in his famous work on evolution by natural selection. Mutation and selection are taken as rules for variation of the parameters and selection for filtering during the iteration sequence. Schwefel lists out the steps in the evolution scheme, in the language of biology, as follows:

[10] A blend of two or more words into a new one.

Step 0: Initialization A given population consists of two individuals, one parent and one descendant. They are each identified by their genotype according to a set of n genes. Only the parental genotype has to be specified as starting point.

Step 1: Mutation The parent $E^{(g)}$ of the generation g produces a descendant $N^{(g)}$, whose genotype is slightly different from that of the parent. The deviations refer to the individual genes and are random and independent of each other.

Step 2: Selection Because of their different genotypes, the two individuals have a different capacity for survival (in the same environment). Only on of them can produce further descendants in the next generation, namely the one which represents the higher survival value. It becomes the parent E^{g+1} of the generation $g + 1$.

2.9.5 Artificial Neural Networks

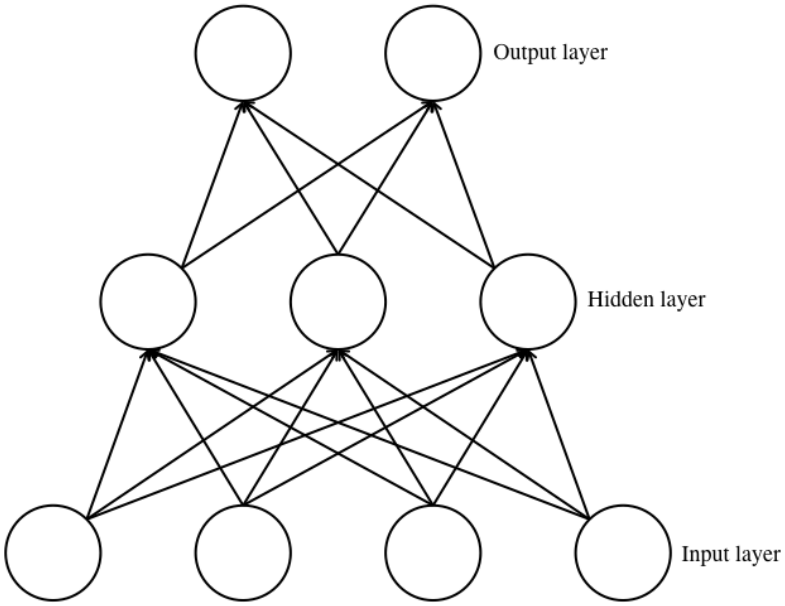
Artificial Neural Networks (ANNs, singular ANN) are computational modeling tools, used in many disciplines for modeling complex real-world systems. ANNs are inspired by biological neural networks, such as the human nervous system [5]. There are a lot of attractive characteristics that makes ANNs popular modeling tools. Their nonlinearity allows for a better fit to the data, their noise-insensitivity makes predictions possible in the presence of uncertain data or data containing measurement errors, with high parallelism they can achieve fast processing and hardware failure-tolerance, they can learn and adapt to changing environments and they can generalize which enables them to handle unlearned data [5]. ANNs have seen a popularity in many fields of research and there is a book written on the topic and its uses in finance by G. Deboeck [17].

ANNs are constructed of interconnected adaptive simple processing elements called artificial neurons or nodes. An ANN can be described as i directed graph in which each node i performs a transfer function f_i of the form

$$y_i = f_i \left(\sum_{j=1}^{n} w_{ij} x_j - \theta_i \right) \tag{2.5}$$

where y_i is the output of the node i, x_j is the j^{th} input to the node and w_{ij} is the connection weight between nodes i and j. θ_i is the threshold (or bias) of the node. Usually, f_i is nonlinear, e.g. a heaviside, sigmoid or Gaussian function [60]. Depending on how neurons are connected, ANNs can be classified as feedforward or recurrent. A feedforward ANN has no looping, i.e. you cannot

Figure 2.3: Multilayer perceptron showing input, hidden and output layers and nodes with feedforward links.

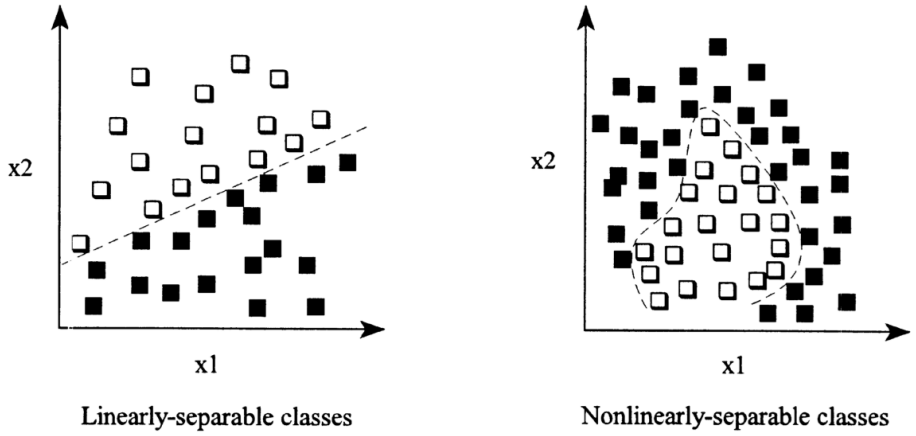


start on a node n in the network and move through the network ending on n again (directed acyclic graph or DAG). In a recurrent ANN there are no such limits.

Figure 2.3 shows an illustration of an ANN, a feedforward multilayer perceptron (MLP). The MLP can be used to solve problems which are not linearly separable as illustrated in figure 2.4.

The process of "learning" in ANNs is typically accomplished using examples [60]. This process is also known as "training" the neural net. Learning is achieved by adjusting the connection weights iteratively so that the ANN can perform a specified task. The learning of an ANN can be roughly divided into three types of learning according to Yao [60]. Learning by direct comparison of the output of the ANN and the desired or target output is called supervised learning. The comparison is often formulated as a minimization of an error function such as the total mean squared error (MSE) between the output of the ANN and the actual output (target output) summed over all available data. The minimization of the error function can be done by employing e.g. a gradient descent-based algorithm such as backpropagation to adjust the weights on the connections between the neurons. Another type of learning, called reinforcement learning, can be used in cases where the desired output is unknown, but the actual output of the ANN can be said to be correct or not. The third type of learning, unsupervised learning, is solely based on the correlations among input data, i.e. there is

Figure 2.4: Linear vs. nonlinear separability.



no information about what constitutes as "correct output" that can be used in the learning [60].

2.9.6 ADATE

Automatic Design of Algorithms Through Evolution, otherwise knows as ADATE, is a system for automatic programming developed by Roland Olsson at Østfold University College. Automatic programming is inductive inference of algorithms [58], a system to automatically create algorithms through the use of evolution. ADATE has its own web page that can be found on the following address: `http://www-ia.hiof.no/~rolando/`. This web page holds a manual that explains the technical part of running ADATE and a number of papers written about the system.

2.10 Stockholm Stock Exchange

The Stockholm Stock Exchange (SSE) is the largest stock exchange in the Nordic countries and one of the ten largest in Europe [46]. The stock exchange had 310 companies listed in quarter two of 2004[11], with a combined market cap of 2.521 trillion Swedish krona. The market cap is calculated by multiplying the share price with the number of shares outstanding for each company. SSE was acquired by OMX in 1998. In February 2008 OMX was incorporated in the NASDAQ OMX group, who now control NASDAQ and seven European stock exchanges.

The decision to use market data from the SSE in this thesis is based on the SSE's dominant position in the Nordic market, the diversity of the companies listed there and the availability of

[11] `http://en.wikipedia.org/wiki/Stockholm_stock_exchange`

order data by electronic means.

2.10.1 Rules

NASDAQ OMX has a extensive portal with documents regarding rules and regulations for the nordic NASDAQ OMX exchanges including the SSE. The portal resides at this URL: `http://www.nasdaqomx.com/listingcenter/nordicmarket/rulesandregulations/`.

Chapter 3

Implementation

This chapter gives a detailed description of the order book simulator, that can be used as a platform for discovering trading strategies, and an explanation of the data set the order book simulator uses. The order book simulator is not only useful for strategies using the information of the order book in strategy formation, but also for all other strategies. This is because no matter how a strategy is conceived, you can not be sure that the strategy's opening and closing of market positions would be possible, it depends on factors in the order book. With the order book simulator one can determine if the projected trade would be possible with the desired volume and price.

The order book simulator simplifies the market model of NASDAQ OMX Stockholm quite a lot, by e.g. only have the functionality ti simulate limit and market orders. There are a lot of other kinds of orders with special rules, such as market maker orders, iceberg orders and even fully hidden orders. It would be an enormous task to model every aspect of the market model, and it is perhaps even unnecessary depending on if the trading strategies rely on a particular aspect or not. A list of all the different kinds of orders can be seen on page 26 in the market model manual for the stock exchanges under the NASDAQ OMX umbrella [43].

3.1 Preparatory Work and Data Collection

The application PrimeTrader from the Norwegian company Netfonds gives access to stock information from the Nordic stock markets. The application can be used for free and without a subscription, but you will get a 15 minute delay on all data updates. The program has an option to export data to Excel and this is done by selecting an option in the program settings. This program will in turn create a local web server that forwards stock information. By utilizing different URL parameters,

the output of the web server can be customized. You can get order information about a specific paper or you can get order information for a range of papers by accessing lists of favorites created in PrimeTrader. The same can be done for quotes and trades. The output is a web page with a simple HTML table format. This data, which is intended for import into Excel spreadsheets, can also be read by parsers made in any programming language. In this project the data is read from the web server every second by a program created in Java and stored in a file. This is done every day from 07:45 to 17:30 (08:00 to 17:45 because of the delay). The stock exchange is open from 09:00 till 17:30, but orders are committed from 07:45. The list of Large Cap companies at Stockholm Stock Exchange contains 78 companies and each of these has 20 visible price levels. This equates to 1560 lines read every second and 54 756 000 lines per day. After the stock market has closed this file is parsed an each order is stored to a database. When this file is parsed many lines of data will be equal to lines already encountered, because the order book for each company may not contain any changes from the last read. These duplicate lines are ignored when storing to the database.

The program PrimeTrader presumably knows when to update its display based on commands received from Netfond's servers. The data collector program used in this project does not function like PrimeTrader. Because the collector program can not respond to commands from Netfond's servers it reads from the PrimeTrader localhost server at regular fixed intervals. This means that there is a potential for missed orders, i.e. between two reads multiple changes in the order book can happen.

When all the orders are safely stored in the PostgreSQL database the trades conducted this day, for each paper are also stored in the database. This information is gathered directly from the Netfonds data export service on-line. By parsing data from URLs like `http://hopey.netfonds.no/tradedump.php?paper=ABB.ST&csv_format=csv` one can get the daily trades of one company's shares. The paper you want is specified after paper= by substituting "ABB.ST" with a company ticker of your choice followed by ".ST" for specifying the Stockholm Exchange. The data collection in this project does this at the end of the day for all 78 companies.

3.1.1 Alternative Data Sets

There are other options when it comes to assembling a data set for backtesting trading strategies. The approach taken in this thesis requires a stable environment for a data gathering script running for several months (the longer it runs, the more data is collected), collecting data each second of every day. In situations where this approach is not practical, or even feasible, it is perhaps desirable

to use a ready-made data set or construct a data set by combining historical data readily available online. Yahoo! has their own portal for finance residing at `http://finance.yahoo.com/`. This portal contains information from many markets including the SSE. Yahoo! Finance has historical quotes data for the companies in the OMX Stockholm 30 index going back to January 1st 2003. Unfortunately it is not possible to get information about the limit order book, or other intra-day data, for the companies in the OMX Stockholm 30 index, but it is possible to look at the topmost five orders of many other companies, e.g. the companies on NASDAQ.

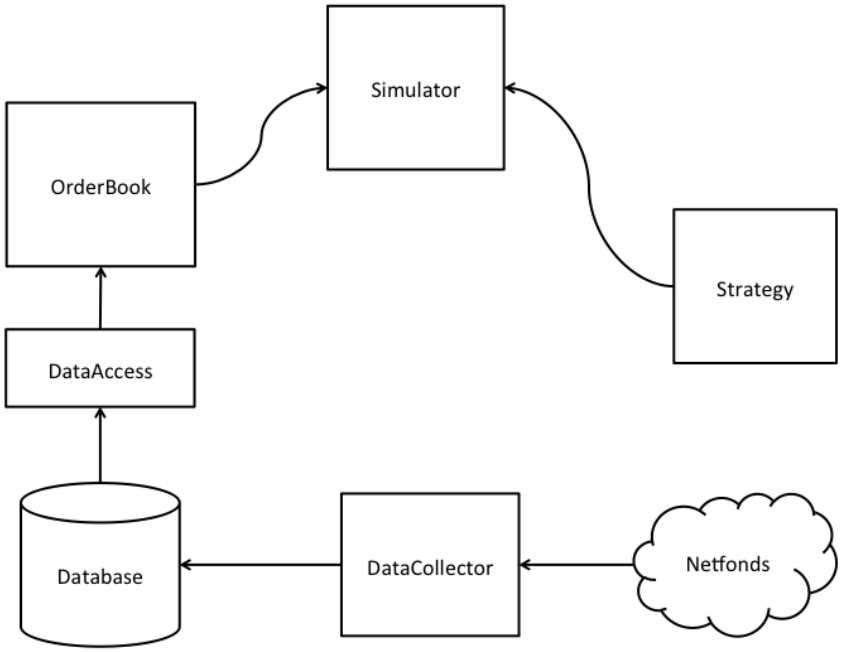
3.2 The Order Book Simulator

The order book simulator discussed in this thesis does not model the entire stock market, including investors, brokers and market makers. Rather, it focuses on the order book containing the limit orders issued for buying and selling shares. The basic functionality of the order book simulator is to accurately calculate at what time hypothetical buy or sell limit orders (synthetic orders) would be executed, or if they would be filled at all, and if they are executed how much of the desired volume is achieved. This is done by combining these synthetic orders with real, historical order book data collected from the Stockholm Stock Exchange as explained in section 3.1. The synthetic orders will move up and down in the order book following their respective price level. If a synthetic order reach the ask or bid level, depending on what type of order is in question, it may be traded if it can be matched with a trade occurring at the same time.

3.2.1 Why an Order Book Simulation?

One major advantage of an order-based simulation platform such as that provided by the order book simulator is that it obviates the need for a fill or execution model. When only price information is available for a stock, and a proposed strategy wishes to place a limit order away from the current price, any simulation must make a decision about whether such an order is executed at a future time based on just the price data. Obviously, if the price never reaches the limit order price, the order would never get filled but if the price does cross the limit order price, it might or might not have been executed in the real market, depending on the depth of demand in the books at that price. A typical fill model might probabilistically execute the limit order based on historical data and the volume of the order [38].

Figure 3.1: System overview of the order book simulator



3.2.2 System Overview

Figure 3.1 shows a simplified schematic overview of the order book simulator. The order book simulator is implemented using the programming language Java. The figure show the main components of the order book simulator, the component for collecting data, the database, the model of an order book and the simulation component that drives the simulation with a strategy for testing.

3.2.3 The Data Set

The data set used by the order book simulator is separated into two tables, “trades” and “orders”. The trades table contain information about all trades that have happened within the time range of the data set, and the orders table contain information about how the order book for the 78 companies in the data set looked like at any particular time. The column names of the trades table have the following meaning:

paper The ticker of the company.

Exchange The exchange in abbreviated form.

ts The time stamp of the trade.

price The price paid.

quantity The number of shares traded.

board Empty column (always NULL).

source The source of the trade, either “U” (Unknown) or “Auto trade”.

buyer The buyer.

seller The seller.

initiator The initiator of the trade, either “S” (Seller), “B” (Buyer) and sometimes blank (NULL).

Each row in the orders table represents a limit order at a particular price level and at a particular time. The maximum visibility of one particular order book is 20 price levels from the top of the book. Information about the buy and sell side of the book are stored in the same row. The column names of the orders table have the following meaning:

paper The ticker of the company.

exchange The exchange in abbreviated form.

ts_src The time stamp of the order book.

lvl The level of the price, ranges from 0 (bid/ask) to 19.

bid_number The number of buy orders on this price level.

bid_depth The combined volume of the buy orders on this price level.

bid_price The bid price.

offer_number The number of sell orders on this price level.

offer_depth The combined volume of the sell orders on this price level.

offer_price The offer price.

ts_parse The time stamp at which the row was fetched from the data provider.

Examples of how the data in the trades table looks like can be seen in table 3.1 and examples of how the data in the orders table looks like can be seen in table 3.2 and table 3.3. The two tables showing data from the orders table contain two complete order books at two consecutive points in time. If one looks closely at the time stamps in table 3.1 you can see that trades took place at 13:40:07, 13:40:08 and 13:40:11. The changes the trades at 13:40:07 have done to the order book is likely to be reflected in the order book at 13:40:07 seen in table 3.2. The next order book is recorded at 13:40:27, seen in table 3.3. It is unknown why the trades happening at 13:40:08 and 13:40:11 have not made changes in the order book until 13:40:27.

The data set currently, as of May 2011, holds over 600 million rows of order data and over 17 million trades, for the 78 companies. The database takes up approximately 100 gigabytes of hard drive space.

3.2.4 The Problem of Withdrawn Orders

One of the first problems encountered with the order book simulator, or rather the data used in the order book simulator, is the lack of unique identifiers for the orders and the grouping of orders on the same level (price). The Penn-Lehman project's simulator PSX, which uses data from Island.com, do not have this problem to the same extent, as their data contains unique identifiers for every order [38]. Listing 3.1 shows a snapshot of the order book for Saab Automobile AB (SAAB-B) around 1 PM on January 11th 2011. In this snapshot there are four bids for a combined volume of 1935 shares at 127.40 SEK. There is no information in this snapshot that could uniquely separate these four orders into single orders with a single volume, but this is the data provided by Netfonds. This creates a problem. If a order for 500 shares where added to this level by the order book simulator, the number of orders on the level would rise to 5 and the volume would increase to 2435. Later the number of orders on this level could rise to for example 8. If the number then decreased to 7 because an order was withdrawn there is no way of telling if this order was before or after the simulated order in the queue, thus an estimation of when the simulated order would be filled can not be completely accurate.

Listing 3.1: SAAB-B order book January 11. 2011 12:57:29

```
| Ticker: SAAB-B                                                    |
| Time:   2011-01-11 12:57:29 (2011-01-11 12:57:29)                 |
|                BUY                ||               SELL          |
```

Table 3.1: Trades for VOLV-B between 13:40:07 and 13:40:44 on the 20th of April 2011

paper	exchange	ts	price	quantity	board	source	buyer	seller	initiator
VOLV-B	ST	2011-04-20 13:40:07	110.5000	86		Auto trade	FORA	ENS	S
VOLV-B	ST	2011-04-20 13:40:07	110.5000	1000		Auto trade	SRE	ENS	S
VOLV-B	ST	2011-04-20 13:40:07	110.5000	1324		Auto trade	OHM	ENS	S
VOLV-B	ST	2011-04-20 13:40:07	110.5000	1772		Auto trade	ABC	ENS	S
VOLV-B	ST	2011-04-20 13:40:07	110.5000	2085		Auto trade	MLI	ENS	S
VOLV-B	ST	2011-04-20 13:40:07	110.5000	2583		Auto trade	FORA	ENS	S
VOLV-B	ST	2011-04-20 13:40:07	110.6000	817		Auto trade	SWB	NON	B
VOLV-B	ST	2011-04-20 13:40:08	110.6000	242		Auto trade	SWB	CSB	B
VOLV-B	ST	2011-04-20 13:40:08	110.6000	409		Auto trade	SWB	NON	B
VOLV-B	ST	2011-04-20 13:40:08	110.6000	600		Auto trade	SWB	NON	B
VOLV-B	ST	2011-04-20 13:40:08	110.6000	821		Auto trade	SWB	CSB	B
VOLV-B	ST	2011-04-20 13:40:11	110.5000	100		Auto trade	FORA	CAR	S
VOLV-B	ST	2011-04-20 13:40:39	110.6000	92		Auto trade	CDG	CSB	B
VOLV-B	ST	2011-04-20 13:40:39	110.6000	100		Auto trade	CDG	CSB	B
VOLV-B	ST	2011-04-20 13:40:39	110.6000	100		Auto trade	CDG	CDG	B
VOLV-B	ST	2011-04-20 13:40:39	110.6000	146		Auto trade	FORA	CSB	B
VOLV-B	ST	2011-04-20 13:40:39	110.6000	149		Auto trade	CDG	SGP	B
VOLV-B	ST	2011-04-20 13:40:39	110.6000	232		Auto trade	FORA	SGP	B
VOLV-B	ST	2011-04-20 13:40:39	110.6000	500		Auto trade	CDG	SAB	B
VOLV-B	ST	2011-04-20 13:40:43	110.7000	882		Auto trade	SWB	IMC	B
VOLV-B	ST	2011-04-20 13:40:44	110.6000	7		Auto trade	FORA	SWB	S
VOLV-B	ST	2011-04-20 13:40:44	110.6000	300		Auto trade	FORA	SWB	S
VOLV-B	ST	2011-04-20 13:40:44	110.6000	500		Auto trade	FORA	SWB	S
VOLV-B	ST	2011-04-20 13:40:44	110.6000	993		Auto trade	CDV	SWB	S
VOLV-B	ST	2011-04-20 13:40:44	110.6000	1000		Auto trade	SRE	SWB	S
VOLV-B	ST	2011-04-20 13:40:44	110.7000	25		Auto trade	SWB	GSI	B
VOLV-B	ST	2011-04-20 13:40:44	110.7000	300		Auto trade	SWB	CDG	B
VOLV-B	ST	2011-04-20 13:40:44	110.7000	321		Auto trade	SWB	GSI	B
VOLV-B	ST	2011-04-20 13:40:44	110.7000	700		Auto trade	SWB	NON	B
VOLV-B	ST	2011-04-20 13:40:44	110.7000	976		Auto trade	SWB	OPV	B
VOLV-B	ST	2011-04-20 13:40:44	110.7000	1000		Auto trade	SWB	SRE	B

Table 3.2: The order book for VOLV-B 13:40:07 on the 20th of April 2011

paper	exchange	ts_src	lvl	bid_number	bid_depth	bid_price	offer_number	offer_depth	offer_price	ts_parse
VOLV-B	ST	2011-04-20 13:40:07	0	6	8672	110.5000	9	5189	110.6000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	1	20	23390	110.4000	19	31397	110.7000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	2	28	43735	110.3000	25	47195	110.8000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	3	13	34837	110.2000	12	22470	110.9000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	4	9	16036	110.1000	21	37320	111.0000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	5	10	34959	110.0000	6	19232	111.1000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	6	6	26780	109.9000	9	29064	111.2000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	7	4	2602	109.8000	3	14655	111.3000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	8	2	31970	109.7000	8	26147	111.4000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	9	2	2209	109.6000	4	4500	111.5000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	10	1	40690	109.5000	3	41608	111.6000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	11	2	11209	109.4000	2	30508	111.7000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	12	3	46877	109.3000	2	3000	111.8000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	13	2	43808	109.2000	4	30554	111.9000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	14	2	14951	109.1000	51	54985	112.0000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	15	6	4706	109.0000	1	460	112.1000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	16	5	5824	108.9000	1	61424	112.2000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	17	2	3000	108.8000	3	66177	112.4000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	18	2	1490	108.7000	9	7575	112.5000	2011-04-20 13:55:16
VOLV-B	ST	2011-04-20 13:40:07	19	4	2213	108.5000	2	1600	112.6000	2011-04-20 13:55:16

Table 3.3: The order book for VOLV-B 13:40:27 on the 20th of April 2011

paper	exchange	ts_src	lvl	bid_number	bid_depth	bid_price	offer_number	offer_depth	offer_price	ts_parse
VOLV-B	ST	2011-04-20 13:40:27	0	10	11854	110.5000	7	3382	110.6000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	1	21	23967	110.4000	20	29297	110.7000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	2	28	43735	110.3000	26	47474	110.8000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	3	13	34837	110.2000	12	19870	110.9000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	4	9	16036	110.1000	21	37320	111.0000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	5	10	34959	110.0000	7	20232	111.1000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	6	6	26780	109.9000	10	30049	111.2000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	7	4	2602	109.8000	3	14655	111.3000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	8	2	31970	109.7000	8	26147	111.4000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	9	2	2209	109.6000	4	4500	111.5000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	10	1	40690	109.5000	3	41608	111.6000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	11	2	11209	109.4000	2	30508	111.7000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	12	3	46877	109.3000	2	3000	111.8000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	13	2	43808	109.2000	4	30554	111.9000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	14	2	14951	109.1000	51	54985	112.0000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	15	6	4706	109.0000	1	460	112.1000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	16	5	5824	108.9000	1	61424	112.2000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	17	2	3000	108.8000	3	66177	112.4000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	18	2	1490	108.7000	9	7575	112.5000	2011-04-20 13:55:35
VOLV-B	ST	2011-04-20 13:40:27	19	4	2213	108.5000	2	1600	112.6000	2011-04-20 13:55:35

NUMBER	VOLUME	PRICE		PRICE	VOLUME	NUMBER
1	142	127,70		127,90	461	1
3	636	127,50		128,00	2775	4
4	1935	127,40		128,10	975	3
3	775	127,30		128,30	1005	3
2	2478	127,20		128,50	116	1
1	600	127,10		128,80	50	1
1	900	126,80		129,00	2817	2
2	395	126,70		129,40	400	1
1	500	126,00		129,50	600	1
1	100	125,50		129,70	400	2
1	50	120,00		129,90	100	1
1	50	119,90		130,00	1130	6
1	714	119,10		130,40	115	1
1	200	110,00		132,00	100	1
1	1000	81,00		133,00	50	1
				135,00	100	1
				136,00	100	1
				137,00	100	1

3.2.5 The OrderBook Class

The central component of the order book simulator is the Java class OrderBook. This class is a model of the real order book and it has methods that corresponds to the real world actions one can take on an order book including issuing and withdrawing limit buy or sell orders. Since this order book simulator works with historical data one can do the simulation on, and see the order book for any company on the Stockholm Stock Exchange large cap list, on any day that is included in the data set.

The OrderBook class has methods to initialize the order book, filling it with orders for the indicated ticker and time stamp, and to update the order book with data for each change in the historical order book stored in the data set. Initializing an OrderBook object is simple and involves little logic. The buy and sell orders are stored in separate maps of key value pairs with the key being the price and the value being a list of orders for the given key (price). The update method involves much more logic. There are 12 different changes in the order book that can occur on each update

and they will be listed here. Many if these cases occur because information happening between the 1 second update intervals of the data collector is lost. The time stamp always change at each update. The cases and the programatic actions performed when each of the cases are encountered can be seen in table 3.4 on page 41.

Case 1a The bid/ask number has increased with exactly 1 and the volume has increased. This can be interpreted as a new order added to the order book. This new order's volume $v = |v_t - v_{t-1}|$ where v is the new volume, v_t is the total volume of the order book for the current price level right now and v_{t_1} is the total volume of the order book at the last update, i.e. the volume of the new order is the change in total volume from the last update.

Case 1b The bid/ask number has increased with more than one. This means that multiple orders has entered the order book since the last update. The volume for this new order is calculated like in case 1a. This order will be considered a composite order since it comprises several orders.

Case 2 The bid/ask number has increased and the volume has decreased. An increase in the bid/ask number implies that orders have been added to the order book, but the loss in volume suggests that these new order's combined volume is less than the combined volume of the orders that must have been withdrawn or traded (if the change occurs at bid/ask price).

Case 3 The bid/ask number has increased and the volume is unchanged. As in case 2, an increase in bid/ask number implicate new orders, but in this case the combined volume of the new orders and the combined volume of the orders that must have been withdrawn or traded is exactly equal, leaving the total volume unchanged.

Case 4 The bid/ask number has decreased and the volume has increased. An increase in volume implies that new orders have joined the order book. Since the bid/ask number has decreased, orders have also been withdrawn or traded and these orders have been more numerous than then the ones being added, but they have a smaller combined volume.

Case 5a The bid/ask number has decreased with exactly one and the volume has decreased. It is clear that orders have been withdrawn or traded. A search through the trade records can determine which is the case for the bid/ask price level. At any other level a withdrawal is the only thing that could have occurred.

Case 5b The bid/ask number has decreased with more than one and the volume has decreased. This means that more or less the same as case 5a has happened, but more orders have been withdrawn or trades have filled multiple orders. As in case 5a the trade records must be searched to find the best explanation.

Case 6 The bid/ask number has decreased and the volume is unchanged. A drop in bid/ask number implies that orders have been withdrawn or traded. But orders must also have been added, making up the same total volume, but since the bid/ask number has decreased these new orders must have been fewer and larger orders.

Case 7 The bid/ask number is unchanged and the volume has increased. The same amount of orders must have been withdrawn or traded as well as added, but the newly added orders have larger volumes.

Case 8 The bid/ask number is unchanged and the volume has decreased. The same amount of orders must have been withdrawn or traded as well as added, but the newly added orders have smaller volumes.

Case 9 The bid/ask number and the volume for this price level are unchanged since the last iteration and no action is required.

Case 10 The price level was not present in the order book on the last update and a new price level must therefore be created.

It is important to note that for the cases 1a and 1b, withdrawals and trades could still be happening, even if both the bid/ask number increase and the total volume increase. The same goes for cases 5a and 5b and also for case 9. The overall change to the bid/ask number and volume is an increase (1a or 1b) or decrease (5a or 5b) or no change at all (9), but new orders and/or withdrawals/trades could still be occurring. In these cases it is impossible to know if a withdrawal (1a or 1b) or new order addition (5a or 5b) has taken place. If both ask/bid number and volume increase, like in case 1a and 1b, and trades have occurred on the price level in question, the new orders would be larger than the ones being filled. For case 9 trades would imply that new orders have been added making up the same bid/ask number and volume as the last update, but this occurrence seems unlikely.

Table 3.4: Case actions, number and volume in the context of this table refers to the change in number and volume from one order book change to the next.

Case	No.	Vol.	Preferred action	Alternative action I	Alternative action II	Alternative action III
1a	↑ 1	↑	Add a new regular order with volume	-	-	-
1b	↑	↑	Add a new composite order with number and volume	-	-	-
2	↑	↓	Subtract volume from a random composite order, add number to the last non-synthetic order	Subtract volume from a random regular order, add number to the last non-synthetic order	Subtract volume from a random composite order after merging, add number to the last non-synthetic order	-
3	↑	-	Add number to the last non-synthetic order	-	-	-
4	↓	↑	Subtract number (+1) from a random composite order, add a new regular order with volume	Subtract number (+1) from a random composite order after merging, add a new regular order with volume	Subtract number from a random composite order, add volume to the last non-synthetic order	Subtract number from a random composite order after merging, add volume to the last non-synthetic order
5a	↓ 1	↓	Remove a random regular order with volume	Subtract number (1) and volume from a random composite order	Subtract number (1) and volume from a random composite order after merging	-
5b	↓	↓	Find an exact subset of orders and remove its members	Find an approximate subset of orders and remove its members	-	-
6	↓	-	Subtract number from a random composite order	Subtract number from a random composite order after merging	-	-
7	-	↑	Subtract number (1) from a random composite order, add a new regular order with volume	Subtract number (1) from a random composite order after merging, add a new regular order with volume	Add volume to the last non-synthetic order	-
8	-	↓	Subtract volume from a random composite order	Subtract volume from a random regular order	Subtract volume from a random composite order after merging	-
9	-	-	No action required	-	-	-
10	-	-	Create new price level	-	-	-

3.2.6 Solutions for Case 5a and Case 5b

The solutions to cases 5a and 5b are the most difficult to implement because they are subject to the problem discussed in section 3.2.4. There are several different possible solutions, but no solution can be 100% accurate 100% of the time as explained earlier. Choosing the best solution is a question of selecting a worst or best case scenario, in terms of synthetic order progression, or something in between worst and best, i.e. removing the withdrawn orders at random indices, guided by the indications that the historical data gives. The order book simulator tries to adhere to the latter principle.

Case 5a is the easier of the two. When the bid or ask number decrease with exactly one, the order book simulator calculates the change in volume from the current order book and the last. It then tries to find an order in the order book, for the current price level, with this volume and then remove it. If there are several such orders, one of them are selected at random and remove. This is a very crude an inaccurate way of determining which of the orders to remove. If the removed order is the only one in the order book on this price level with that exact volume, it is likely that the order book simulator removes the correct order. But it is quite probable that there exists multiple orders with this volume. Orders have a tendency to have volumes with rounded numbers. So if there are 10 orders of 100 shares and the total volume drop with 100, one of these 10 orders would be removed, even though it could be any of the 9 other orders that is actually the order being withdrawn. This is important, because this behavior determines how fast synthetic orders could potentially get ahead, in the queue of orders waiting to be executed. If the order book simulator is more likely to remove orders in front of simulated orders the strategies run in the order book simulator could get an unfair advantage by getting orders filled prematurely. An alternative solution would be a worst case approach (for the trading strategy), where withdrawn orders are only removed from behind the synthetic orders if possible. This would give the trading strategies an unfair disadvantage seeing how a majority of the orders are withdrawn rather than filled, i.e. simulated orders would have a difficult time getting ahead in the queue, since the only thing bringing them forward are trades and trades are less numerous than withdrawals. A third solution would be to remove orders at random indices, with the change in volume as guide, e.g. remove a random order of 100 shares if such an order exists, or else subtract 100 shares from a random composite order. This is the solution used in the order book simulator. Using this solution will not change a synthetic order's chances of being traded unfairly and the solution should not make the order book simulator diverge too much from the actual market behavior.

Case 5b is little bit more complicated. The case is pretty much the same, but more than one

order must be removed from the order books. This could involve removing a subset of orders that together make up the decrease in total volume. This problem is a variation of the subset sum problem[1]. The problem of finding a combination of volumes that make up a specified total volume is NP-complete[2], but computation time should not be a problem here because the lists of orders for each price level are not especially long. If such a combination of volumes exist, the corresponding orders can be removed. If this combination can not be found one could remove the remaining volume from a composite order (if the subset sum algorithm finds an approximate solution), or subtract the entire change in volume and bid/ask number from a single composite order. The latter solution have the same possible inaccuracy as the first solution for case 5a. Given the example with 10 orders of 100 shares, a subset sum algorithm could easily find a combination given a decrease in volume of 300 shares, but that does not mean that the correct 3 orders will be removed. If the subset sum algorithm finds only one solution it is more likely that it has pointed out the correct orders. The same worst case solution as in case 5a could be implemented for case 5b (where orders only are removed from behind synthetic ones if possible), but such a solution has detrimental effects to strategies as explained above. Orders could also be removed at random indices and randomness plays an important role in the subset sum function explained below. The subset sum approach to these cases is the approach taken in the order book simulator.

3.2.7 OrderBook Functions

This section will go through the most important functions used to handle the different cases explained earlier in more detail. An order, being independent or part of a list, is defined as (n, v, p, t) where n is the number of orders combined in this order ($n = 1$ for regular orders, $n > 1$ for composite orders), v is the volume (depth), p is the price and t is the time of arrival. An order has more members in the order book simulator itself, but this definition is sufficient for understanding the pseudo-code following below. The property $.size$ denotes the number of items in a list, $.composite$ all composite orders in a list, $.regular$ all regular orders in a list and the function $.index(i)$ the index if item i in a given list. The $random(x, y)$ function used in the algorithms below supplies a random integer between x (inclusive) and y (exclusive). Functions that were to big to be practical in pseudo-code can be seen in Java code in appendix B.

[1] http://en.wikipedia.org/wiki/Subset_sum
[2] http://en.wikipedia.org/wiki/NP-complete

step()

This function is used to move the order book from one change event to the next. The Java code for this function can be seen in listing B.1. This function handles the inclusion of synthetic orders.

initialize()

This function is used when the order book is empty and filled for the first time with historical data. The Java code for this function can be seen in listing B.2.

update()

This function is used when stepping ahead to the next change in the order book as given by the data set. All the actions done to handle the cases explained in table 3.4 are taken in this function. The Java code (excerpt) for this function can be seen in listing B.3.

cleanup()

This function is used to remove price levels in the order book that should no longer be present in the order book, either because all orders on a given price level are withdrawn, or because they are withdrawn. The Java code for this function can be seen in listing B.2.

removeTradedOrders()

Between each order book update, trades can have occurred. This function serves to remove the orders that have been completely filled by these trades. If any of these orders are synthetic orders their partial or complete execution are recorded including time of execution, price, volume filled and the ID of the order. The Java code for this function can be seen in listing B.7.

addVolumeLast(orders[], volume, time)

This function is used to add an increase in volume to the end of a list of orders. The increase is added to the last non-synthetic order. If the last non-synthetic order is also the last order in the list, it will get an updated time stamp. Pseudo-code for this function can be seen in algorithm 1.

Algorithm 1 $addVolumeLast(orders[], volume, time)$

```
N = orders.size
last = 0
for i = N − 1 → 0 do
  if orders[i] not synthetic then
    last ← i
    break
  end if
end for
if last = N − 1 then
  order[i].volume ← order[i].volume + volume
  order[i].time ← time
else
  order[i].volume ← order[i].volume + volume
end if
```

addNumberLast(orders[], number, time)

This function is used to add an increase in number to the end of a list of orders. The increase is added to the last non-synthetic order. If the last non-synthetic order is also the last order in the list, it will get an updated time stamp. Pseudo-code for this function can be seen in algorithm 2.

Algorithm 2 $addNumberLast(orders[], number, time)$

```
N = orders.size
last = 0
for i = N − 1 → 0 do
  if orders[i] not synthetic then
    last ← i
    break
  end if
end for
if last = N − 1 then
  orders[last].number ← orders[last].number + number
  orders[last].time ← time
else
  orders[last].number ← orders[last].number + number
end if
```

addNumber(orders[], number)

This function is used to add a number to a random composite order. If the list contains no composite orders the number is added to a regular order. It is preferred to add the number to a composite order since composite orders already comprises several orders. By adding the number to a regular order, the regular order will become a composite order. Pseudo-code for this function can be seen in algorithm 3.

Algorithm 3 $addNumber(orders[], number)$

```
C[] = orders.composite
if C.size > 0 then
    r = random(0, C.size)
    O = C[r]
    i = orders.index(O)
    orders[i].number = order[i].number + number
else
    R[] = orders.regular
    r = random(0, R.size)
    O = R[r]
    i = orders.index(O)
    orders[i].number = order[i].number + number
end if
```

removeVolume(orders[], volume)

This function is used to remove a fall in volume from a list of orders. The preferred action is to remove the volume from a composite order with a combined volume higher than the volume to remove, preventing a negative volume. If there are no composite orders that can support the subtraction, the volume is removed from a large enough regular order. If no such order exists the function calls the $mergeOrders()$-function to combine orders and then it calls itself. Pseudo-code for this function can be seen in algorithm 4.

removeNumber(orders[], number)

This function is used to remove a fall in number from a list of orders. The preferred action is to remove the number from a composite order with a combined number higher than the number to remove, preventing a negative number. If there are no composite orders that can support the

Algorithm 4 $removeVolume(orders[], volume)$

```
C[] = orders.composite
if C.size > 0 then
    r = random(0, C.size)
    O = C[r]
    i = orders.index(O)
    orders[i].volume = order[i].volume − volume
else
    R[] = orders.regular
    if R.size > 0 then
        r = random(0, C.size)
        O = C[r]
        i = orders.index(O)
        orders[i].volume = order[i].volume − volume
    else
        mergeOrders(orders)
        removeVolume(orders, volume)
    end if
end if
```

subtraction, the function calls the $mergeOrders()$-function to combine orders and then it calls itself. Pseudo-code for this function can be seen in algorithm 5.

Algorithm 5 $removeNumber(orders[], number)$

```
C[] = orders.composite
if C.size > 0 then
    r = random(0, C.size)
    O = C[r]
    i = orders.index(O)
    orders[i].number = order[i].number − number
else
    mergeOrders(orders)
    removeNumber(orders, number)
end if
```

remove(orders[], volume, number)

This function is used when there is a drop in both volume and number. The Java code for this function can be seen in listing B.5. In this function the case actions for cases 5a and 5b are taken, including the subset sum algorithms.

mergeOrders(orders[])

In some situations in the order book simulator there is not a composite or regular order (depending in the case) to explain the change in number and volume. In such cases a random composite (is such an order exists) or a regular order (if an composite order does not exist) must be merged enough times so that the subtraction can be done. This random order is merged with the next or preceding order on the price level, depending on there is next or preceding order. If there is both an order before the randomly selected order, and one behind in the queue, the order to be merged together also selected at random. The Java code for this function can be seen in listing B.6.

3.2.8 Exact Subset Sum Function Using Dynamic Programming

The order book simulator use a subset sum function implemented using dynamic programming. Dynamic programming is explained in section 2.9.2. This function finds exact solutions to the subset sum problem, if they are present in the order book. This subset sum function is used to find candidate solutions for the problem of withdrawn orders explained in section 3.2.4. If no such solution exists, the order book simulator tries to find and approximate solution using the function explained below. Note that the algorithm only use the drop in volume to find combinations, and that the solutions therefore can contain an incorrect number of orders. The pseudo-code for the function can be seen in algorithm 6.

3.2.9 Approximate Subset Sum Function Using Local Search

The order book simulator employs a subset sum function implemented using a local search algorithm. The concept of local search is explained in section 2.9.3. This subset sum function is used to find candidate approximate solutions for the problem of withdrawn orders explained in section 3.2.4, if none are found using the exact implementation explained above. The subset sum function can be seen in pseudo-code in algorithm 7. The idea of the algorithm is to find combinations of orders that can be removed according to changes in the volume. Note that the algorithm, just like the exact version, only use the drop in volume to find combinations, and that the solutions therefore can contain an incorrect number of orders.

The algorithm starts with the list of orders ($Xs[]$) that contain the subset, an empty list for the solution ($Ss[]$), the target volume (T) and a variable containing the best absolute difference between the target ($best$), and the sum of the orders in the solution seen so far. In a for loop doing a limited amount of iterations the algorithm then does one of three operations depending on the situation. If

Algorithm 6 $exactSubset(Xs[], T)$

```
N = Xs.size
Ss = [N + 1]
Ts = [T + 1]
for i = 1 → N + 1 do
    Ss[i] = Xs[i − 1]
end for
for i = 1 → T + 1 do
    Ts[i] = 0
end for
Ts[0] = −1
for j = 1 → N + 1 do
    for i = 1 → T + 1 do
        if (i ≥ Ss[j])AND(Ts[i − SS[j]] ≠ 0)AND(j ≠ Ts[i − Ss[j]])AND(Ts[i] = 0) then
            Ts[i] = j
        end if
    end for
end for
solution = []
if Ts[T] > 0 then
    for i = T; i > 0; i = i − Ss[Ts[i]] do
        solution ← Ss[Ts[i]]
    end for
    return solution
else
    return NULL
end if
```

the sum of the volumes in the solution is zero an order from $Xs[]$ is added. If the sum is less than the target there is an equal chance of either adding an additional order to the solution or replacing one of the orders already in the solution with another one (swap). If the sum is greater than the target there is an equal chance of either removing an order from the solution or swapping an order. These operations (add, remove, swap) are only carried out if the solution is better, i.e. the absolute difference between the target and sum is lower after the operation. If this is not the case nothing is done in that particular iteration. When a predetermined number of iterations have been carried out the solution is considered optimal and impossible to improve further.

This algorithm runs the risk of getting trapped in inferior local optima, and it can not make moves that make the solution worse, in terms of the absolute difference between the sum and the target value. This is exactly the problem tabu search (see section 2.9.3) can be used to avoid. But the potentially non-optimal solutions the algorithm provides serves the order book simulator well enough, seeing how a better local optimum or even global optimum would not make the solution "correct" considering the problem of withdrawn orders.

The $random(x, y)$ function used in algorithm 7 gives a random integer between x (inclusive) and y (exclusive).

3.2.10 Data Access

Access to the data set is done through the DataAccess interface which is implemented using regular SQL statements which in turn are created using the simple third party Java library Squiggle[3]. New implementations could easily be added if necessary e.g. if the data resides in flat files or elsewhere. Flat files would not be optimal though, given the amount of data in the database.

[3] `http://code.google.com/p/squiggle-sql`

Algorithm 7 $approximativeSubset(Xs[], T)$

```
Xs[] ← pool
T ← target
Ss = []
best = Integer.MAX
for i = 0 → 1000 do
    pCount = Xs.size
    sCount = Ss.size
    sSum = sum of Cs
    if pCount = 0 AND ss ≤ T then
        break
    end if
    pIndex = random(0, pCount)
    if sSum = 0 then
        Ss ← Xs[pIndex]
        best = |T− sum of Ss|
    else
        sIndex = random(0, sCount)
        pOption = Xs[pIndex]
        sOption = Ss[sIndex]
        newBest = 0
        if sSum < T then
            if random(0, 1) then
                newBest = |T − (sSum + pOption)|
                if newBest < best then
                    Ss ← Xs[pIndex]
                    best = newBest
                end if
            else
                newBest = |T − (sSum − sOption + pOption)|
                if newBest < best then
                    Xs ← Ss[pIndex]
                    Ss ← Xs[pIndex]
                    best = newBest
                end if
            end if
        else
            if sSum > T then
                if random(0, 1) OR pCount = 0 then
                    newBest = |T − (sSum − sOption)|
                    if newBest < best then
                        Ss ← Xs[sIndex]
                        best = newBest
                    end if
                else
                    newBest = |T − (sSum − sOption + pOption)|
                    if newBest < best then
                        Xs ← Ss[pIndex]
                        Ss ← Xs[pIndex]
                        best = newBest
                    end if
                end if
            end if
        else
            break
        end if
    end if
end for
return Cs
```

Chapter 4

Testing

This chapter will explain how the order book simulator has been tested. The testing is done to verify that the order book simulator works as intended with preferably optimal consistency and accuracy.

4.1 Preliminary Test

The order book simulator was tested extensively during development to ensure that it worked as intended. Listing D.1 in appendix D shows one of tests that was part of this testing. A sell limit order for 100 shares was entered into the order book of Hennes & Mauritz (HM-B) at 11:07:50. The limit price was 209.40SEK. The order entered the order book with 4 price levels about it on the sell side of the order book. After arriving at the top of the order book several times it was in front of the queue of the orders with the asking price at 11:48:55. Following a trade of 260 shares at that ask price at 11:49:39, the 100 shares was traded at the desired limit price. The order book following this trade at 11:49:39 shows that our limit order has been removed after its issuance. By doing experiments like these it is shown that the simulator works, but further testing was needed before one could say that 11:49:39 was the correct fill time for this limit order.

4.2 Simulator Accuracy

Before it is possible to start looking for profitable trading strategies using the simulator, we have to test the accuracy of the order book simulator. This is done with a series of tests in which actual orders are superseded by identical synthetic orders. If the simulator is accurate these synthetic orders should be executed in the same way as their real world counterparts. This testing is not done

on all the 78 companies in the database, but five of the companies in the OMX Stockholm 30 index[1]. It was initially planned to do a test on each of the companies in the OMX Stockholm 30 index, but difficulty in finding good test cases made it extremely difficult. The results from the five tests can be seen in table 4.1.

As described in section 3.2 about the simulator implementation, the orders in the dataset are not uniquely identifiable and are grouped together with other orders with the same price. Trades in the dataset does not contain information about which order each trade has filled. This makes finding suitable test cases extremely hard because the factors making the simulator accurate also affects finding good test cases. It follows logically that if you could, with absolute certainty, follow an order from its issuance to it being traded, there would be no problem implementing a completely accurate simulator. The solution is to search the dataset manually, looking for test cases where one can know with relative good certainty when an order was added to the order book and when it was removed because it was filled. The level of difficulty finding test cases increase with the number of order book changes for a given ticker. It is also difficult to find test cases where there is a long time interval between order submission and the actual trade. It is desirable to have such test cases because they would more conclusively point out problems with the simulator accuracy. This because an order that has been in the order book a relatively long time has a bigger chance of being in the wrong place in the queue with the other orders on the same price level. The tests conducted will contain cases with small intervals between order submission and trade and cases with longer intervals. The day of each test is chosen arbitrarily.

Because it is almost impossible to find good test cases in the tests explained above, especially those with long waiting periods between issuance and filling of an order, there might be better to try other approaches to testing the accuracy of the order book simulator. One approach is to find an order that has been in the order book for a long time, perhaps even several hours. Replacing this order with a synthetic one and running the simulation several times to check if the order is filled (or even just withdrawn) at the same time in every simulation run could prove useful. This is because of the stochastic approach taken in most of the case solutions. If an order is filled at the same time each run, it could indicate that the time of filling is the correct one. And this can in turn indicate that the price level the order has been on has mostly been subjected to the preferred actions of case 1a and 5a, and thereby not been subjected to too many actions involving random selection. In periods of high volatility, periods that could be especially interesting for trading strategies, there is a higher potential for cases other than 1a and 5a, because of rapid price changes. Determining if synthetic

[1] http://www.nasdaqomxnordic.com/index/

Table 4.1: Test results for the five companies of OMX Stockholm 30. 100@150.00SEK means by 100 shares when the trading price reach 150.00SEK

Company (Ticker)	Order	Issued	Traded	Traded (SIM)	Error
Getinge B (GETI-B)	SELL: 1182@148.80 SEK	2011-04-21 09:19:40	09:22:37	09:22:37	0
Modern Times Group B (MTG-B)	BUY: 200@425.6SEK	2011-03-15 17:22:32	17:22:56	17:22:56	0
Securitas B (SECU-B)	SELL: 130@74.40SEK	2011-02-17 09:02:15	09:02:46	09:02:46	0
Swedish Match (SWMA)	BUY: 306@206.00SEK	20110310 13:17:00	13:20:53	13:20:53	0
Lundin Petroleum (LUPE)	SELL: 160@84.45SEK	2011-01-11 13:33:26	13:34:56	13:34:56	0

orders are executed at the exact right time in such periods is next to impossible with the current data set.

No matter how many simulation runs that are carried out, it is not possible to determine the exact accuracy of the order book simulator without having test cases where you know when an order entered the order book and when it was removed. The biggest problem is understandably when orders are removed, new regular orders[2] in the order book are quite common, thus the time a single order entered the order book is quite certain (Other changes could have occurred as explained earlier). The correct time of the filling depends on if the given order's journey through the order book is the correct one, i.e. if it is at the bid/ask level of the order book when it is supposed to.

4.2.1 Test Results

The value of the test results is quite limited. This is because all the test cases are cases where there is a short time span between order issuance and filling, i.e. the replacement orders have not had much time to move around in the order book because of random case solutions. But the simulator works with 100% accuracy for these cases. The test cases with the biggest time span between issuance and filling of the limit order, which perhaps are the most interesting, can be seen in listing E.1 and E.2, which is taken from the order book simulator, can be seen in appendix E. In these cases, and the three with just seconds between issuance and filling, it was relatively easy to determine when an order was filled in the real world.

4.3 Case Frequency

This section contains information about the frequency of the different cases seen in table 3.4. By looking at the frequency of the different cases, one can determine if the more "strange" ones, like case 2 and 4, are common or not. The testing is done on two companies, ABB (ABB) and Saab (SAAB-B). ABB is the company with the most order book changes stored in the database (see table 4.2). By doing this it is possible to compare a company with a high level of activity in the order book (such as ABB, which is amongst the companies in the OMX Stockholm 30 index), to a company outside the index with a less busy order book (such as Saab, which is among the least updated order books). The three sample days are the two days where the order book saw the most traffic, i.e. the days that the given ticker had the most rows stored.

[2]Non-composite orders that enter the order book when the total number of orders, on a given level, increase with one from one change in the order book to the next

If we look at the numbers in table 4.3 for SAAB-B, we can see that 78.3% of cases on the buy side, where both the number of orders and the total volume on one price level increase, are of type 1a. This percentage is even higher in the sell side with 87.8% case 1a. This means that the majority of new orders added after case 1a or 1b are singular orders. This is encouraging results seeing how it is best to have as few composite orders as possible. With few composite orders the action taken after the number decrease with one and the volume decrease (case 5a) has more orders to choose from when subtracting an singular order, i.e. simulator is less likely to have to guess which composite order that contains the missing order. Table 4.3 also tell us that cases where the number and volume move opposite to each other are relatively uncommon with just 49 case 2 occurrences on the buy side (23 on the sell side) and 42 case 4 occurrences on the buy side (22 on the sell side). On the whole is also apparent that the cases 1a, 1b, 5a, and 5b are more common than cases where the number and volume moves in opposite directions or one of the values stay unchanged while the other move. It is quite obvious that most frequent case encountered by the simulator this particular run was case 9, where both number and value for a price level stayed the same from on update to the next.

If we turn our eyes to table 4.4 and look at the case frequency for ABB, on the busiest day in the order book, the numbers are substantially higher. This is because of the higher level of activity in the ABB order books. As with SAAB-B the majority of cases are of type 9, i.e. the most common occurrence is that a price level stays unchanged from update to update. If the look at the other cases, cases 1a/1b and 5a/5b are more common than the other "strange" an more difficult to interpret cases. And 1a and 5a are more common than 1b and 5b, with 68.6% of cases 1a/1b being 1a and 66.3% of cases 5a/5b being 5a. So even if the order book for ABB is much more busy than that of SAAB-B, there are still a lot of updates where the number increase (1a) or decrease (5a) with just one, which results in fewer volumes of shares "hidden" in composite orders.

4.3.1 Case Solution Frequency

This section contains information about the frequency of the solutions taken to solve the various cases. This is done to answer questions such as: How often can an the preferred solution be taken, and how often must orders be merged? This testing is done with the same two companies as in section about case frequencies above, ABB and Saab.

The majority of case 5b occurrences are solved by the approximate subset sum algorithm using local search on the buy side, and the exact subset sum algorithm has a tiny majority on the sell side when look at the numbers for SAAB-B. For 5a cases on SAAB-B there are a majority of cases

Table 4.2: The companies of the OMX Stockholm 30 index sorted on number of rows in the data set.

Ticker	Number of rows
ABB	14695750
SAND	14468074
AZN	14250500
ATCO-A	14073269
ERIC-B	14040623
SEB-A	14016392
VOLV-B	13962334
HM-B	13891614
NDA-SEK	13853294
TLSN	13706095
ELUX-B	13563934
SKF-B	13524105
BOL	13461352
SWED-A	13405730
SHB-A	13024941
ASSA-B	12386024
ALFA	12310448
SSAB-A	12247517
ATCO-B	12098159
SCV-B	11944274
INVE-B	11795357
SKA-B	11680471
TEL2-B	11663936
SCA-B	10821285
LUPE	10673345
MTG-B	10456638
SWMA	10407227
GETI-B	10082819
SECU-B	9823433

where a regular order with the exact volume missing can be found. This goes for both sides of the book. This can potentially be explained by the high number of 1a cases, i.e. plenty regular orders to choose from. If we look at ABB there is a large majority of 5b cases solved by the approximate subset sum algorithm. And the majority of 5a cases are solved by subtracting the volume from a composite order, rather than finding an exact regular order. Overall looking at ABB, the solution taken is often the preferred one, except for case 5a.

Table 4.3: Frequency of cases and solutions for SAAB-B on the 1st of December 2010.

		Buy side		Sell side	
Case	Solution	Merges	Count	Merges	Count
1a	P	-	2342	-	1709
1b	P	-	649	-	238
2	P	-	13	-	12
2	I	-	36	-	11
3	P	-	45	-	24
4	II	-	28	-	10
4	III	16	14	15	12
5a	P	-	1179	-	992
5a	I	-	480	-	307
5a	II	331	269	200	181
5b	P	-	320	-	139
5b	II	-	405	-	136
6	P	-	21	-	22
6	I	11	11	8	7
7	P	-	101	-	49
7	I	86	86	69	69
7	II	-	146	-	71
8	P	-	149	-	61
8	I	-	165	-	113
8	II	14	9	5	5
8	III	-	122	-	65
9	-	-	77629	-	68661
10	-	-	2196	-	1018
Total		458	86415	297	73912

Table 4.4: Frequency of cases and solutions for ABB on the 23th of November 2010.

		Buy side		Sell side	
Case	Solution	Merges	Count	Merges	Count
1a	P	-	7365	-	7253
1b	P	-	3371	-	3070
2	P	-	342	-	392
2	I	-	76	-	90
2	II	15	6	46	12
3	P	-	99	-	81
4	P	-	-	-	1
4	I	-	-	-	-
4	II	-	381	-	418
4	III	27	24	23	15
5a	P	-	2531	-	2654
5a	I	-	3071	-	3171
5a	II	1745	935	2013	1075
5b	P	-	765	-	727
5b	I	-	2556	-	2432
6	P	-	71	-	49
6	I	8	7	6	5
7	P	-	1171	-	1346
7	I	67	67	86	86
7	II	-	53	-	62
8	P	-	1426	-	1522
8	I	-	312	-	391
8	II	111	34	93	30
8	III	-	43	-	29
9	-	-	208985	-	209055
10	-	-	1269	-	994
Total		1937	234960	2267	234960

Chapter 5

Future Work

This chapter contains thoughts and ideas on the possible road ahead for this project, including improvements to the order book simulator, the data set and suggestions for further testing and trading strategy discovery.

5.1 Improvements

This section will contain thoughts about how the order book simulator and dataset can be improved to produce better and more accurate results, and thereby become a more usable platform for trading strategy performance testing and evaluation.

5.1.1 Improving the Order Book Simulator

The order book simulator discussed in this thesis is far from perfect an there are a number of things that can be done to make it better in terms of accuracy and usefulness. A list of possible improvements follows bellow.

- In a real stock market the traders can include a maximum lifetime of the limit orders when they are submitted. Have the orders not been filled before this threshold they are automatically removed from the order books. The order book simulator should have the same functionality. This is easily implemented by checking all the synthetic orders in the order book at each update, and remove any orders that have a maximum lifetime that have passed.
- In the current implementation of the order book simulator there are two functions that use subset sum algorithms to find solutions for the problem of which orders that can have been

removed when the total volume and number of orders decrease from one change in the order book to the next. Both these functions exclusively use the change in volume to find the solutions. A better implementation should also consider the change in the total number of orders. This will make it less likely that the existing functions have to add or remove a number of orders if the suggested solution contains an incorrect number of orders.

- A composite order that composes a higher number of orders than its order depth is illogical because each order must at least represent an order depth of one share (e.g. an limit order $(n, v, p) = (5, 2, 77.55)$ where n is the number of orders, v is the total depth of these orders and p is the price. In this case each order represents a depth of below one share). But this can happen in the current implementation of the order book simulator. Measures taken to prevent this should be included in the order book simulator, and this should be relatively easy. The reason why this occurs is probably how the function $removeVolume()$ works. This function's primary method of removing a discrepancy in the total depth is to remove it from a composite order, secondary to remove it from a regular order and if no order with a sufficient depth exists, the function goes on to merge orders together to create a order with the sufficient depth. The function should go on to merge orders if it can not find a composite order that supports the required subtraction without ending up with a depth lower the order number.

- It would have been incredibly interesting to transform the order book simulator into a agent-based system like the Penn-Lehman simulator (PSX, see section 2.7). Master students in machine learning could then compete in creating the best trading strategies using e.g. decision trees, artificial neural nets and ADATE as a way of learning those methods. This could possibly stimulate the learning.

- The order book simulator currently does not simulate the trades happening at the opening and closing calls. A model of these two occurrences that happen twice a day, must be included in the order book simulator to support strategies that rely on taking (or leaving) a position on the beginning (or the end) of the day.

- In the current implementation of the simulator, the synthetic orders take the place of historic orders whenever trades are subtracted from the front of the order book. The simulator should subtract the same amount it subtracts from a synthetic order from the next non-synthetic order in order to get a better explanation of a potential drop in the number of orders or the total volume.

- Synthetic orders remain at the top and can because of this have a price higher than the sell price (lower than the buy price). The clean-up function in the simulator must remove synthetic orders that remain at the top as a results of all historical orders being withdrawn.
- The case actions for all cases that subtract volume could be improved by making a function that looks for similarities in volume, e.g. if the volume missing is 347 and there is a composite order that has a volume of 1647, this might imply that some of drop can be explained by this composite order. Considering high-frequency trading with many orders with small volumes this approach might not be entirely fruitful.

5.1.2 Further Testing of the Order Book Simulator

The order book simulator requires more extensive testing to ensure its usability. The most interesting results perhaps, will come when the simulator is ready for strategy discovery. The chances are maybe slim that revolutionary new profitable strategies are found, but such a finding would be very interesting.

5.1.3 Improving The Data Set

The data set contains huge amounts of data. It could be an idea to collect data for fewer companies than the 78 companies in the SSE Large Cap list. Focusing on the stocks in the OMX Stockholm 30 index could be a better approach, since this list contains the companies with the largest volumes which might also be the most interesting. Focusing on a smaller number of companies would make it possible to increase the polling rate on the market data server from Netfonds to e.g. twice a second or four times a second. This would in turn make it easier to follow changes in the order book, since the data would probably contain fewer inconsistencies, i.e. there will hopefully be less type 1b, 2, 4 and 5b cases and more type 1a and 5a cases, when moving from change to change in the order book. Focusing on the stocks in the OMX Stockholm 30 index will also screen out many less liquid stocks since some of the stocks in the large cap list have very few trades.

5.2 Trading Strategies

This section contains suggestions for trading strategy discovery using the order book simulator. These suggestions will not be limited to strategies capitalizing on the advantages of an open limit order book, even though providing a test bed for limit order strategies is the main objective of

the order book simulator. Any strategy can be tested with order book simulator without major modifications to it. These strategies are just ideas, and they need to be investigated further before proceeding.

5.2.1 Trading Strategy Using Evolutionary Strategy, ANN and SOBI

Static order book imbalance, as explained in section 2.5.9, is an indicator of an imbalance in the order books that can be taken advantage of. With the order book simulator in place it is possible to develop trading strategies that use the principle of static order book imbalance in their decision frameworks. By calculating the VWAP with a weight added to each price level in the order book it is possible to optimize this weight by using an artificial neural net combined with a evolutionary strategy, in order to find the best times to take and leave positions in the market. The optimization of this weight could potentially bias the calculation of the VWAP positively, making imbalances clearer to an algorithmic trading strategy. The fitness can be calculated in terms of excess return in combination with a high Sharpe ratio.

5.2.2 Other Strategies

Pairs trading, as seen in section 2.5.3, is based on quite a simple concept, but the methods used to find good pairs, when to open and close positions etc. can be more complex. Optimizing these parameters, with the objective of maximizing profits, could be done with e.g. a evolution strategy or ADATE. The data set currently does not hold daily quotes data, but such data could easily be obtained if it is needed as part of the pairs formation. When trading the pairs the order book simulator could then be used to test if the desired opening and closing positions would be possible on the given time and day.

Chapter 6

Conclusion

From the outset of this project the goal was to create a simulation of the order books for 78 companies listed at NASDAQ OMX Stockholm. By combining historical intra-day order book data and the trade log containing all the stock trades for these 78 companies with synthetic limit orders, the order book simulation can determine when a synthetic limit order had been filled, had it been issued in the real market. To be able to do this simulation the historical data needed to be collected. A computer was set up to read and store the order book for the companies, every second of every day from 07:45 to 17:30. Despite several power failures and a hardware error, huge amounts of data has been collected. Working with such vast amounts of data has proven to be quite tricky and this combined with the implantation of the simulator has taken up much time.

Testing of the order book simulator has shown that even with the rapid changes in the order books of companies such as ABB, which is among the most active order books, there are a lot of cases where the number of orders on a price level increase with just one order, or decrease with one order. This does of course not imply that information about order issuances and withdrawals are not missed between dataset updates, but it should give the order book simulator a better chance of piecing together where the potential changes in the order books has happened. This is because a high number of regular orders decrease the number of composite orders where the internal volume sizes are unknown and essentially unknowable, and because the minimization of composite orders provides more regular orders to choose from (one of which can be the correct one) when the number of orders on a price level drop by one order. It is important however to note that, even if there are a lot of type 1a and 5a cases in the dataset, there are still a lot of other cases present in the dataset that has the potential to make the simulation of a limit order execution less precise, because the case solutions for those action include a lot of randomization.

Running test cases by replacing real historical orders have shown that the order book simulator is accurate for a special case of orders where there is a relatively short time span between order issuance and filling. These cases are cases where a highly probable historical execution filling can be determined by examining the data set manually. If one had access to another data set with unique identifiers linking trades with orders, albeit just for a limit time, one could find good test cases in the order book simulator, but such databases are usually expensive. Being able to simulate an order book with free and readily available information, combined into a working dataset, is one of the major selling points of the order book simulator discussed in this paper. Especially considering amateur stock traders doing trading as a hobby.

At the end it can be said that, even if the order book simulator is not completely accurate in relation to the historical data, it could still prove useful when testing trading strategies. A lot of other factors in the order book simulator are correct, such as the bid/ask prices, the total depth on each level etc, even if a limit order is not executed at the correct time each time. It is important however, that the actions done in the order book to handle the different cases does not bias the results needlessly. Unfortunately it is not feasible to replace the data set used by the order book simulator should a better one present itself. The order book is highly specialized at working with the distinct form the data set gathered from Netfonds take. Parts of the order book simulator could be used, but major modifications are likely in such an event.

References

[1] F. Allen and R. Karjalainen. Using genetic algorithms to find technical trading rules. *Rodney L White Center for Financial Research-Working Papers-*, 1993.

[2] Aho A.V., Hopcroft J.E., and Ullmann J.D. Dynamic Programming. In *Data Structures and Algorithms*, pages 311–321. Addison-Wesley, 1983.

[3] Aho A.V., Hopcroft J.E., and Ullmann J.D. Greedy Algorithms. In *Data Structures and Algorithms*, pages 321–324. Addison-Wesley, 1983.

[4] Aho A.V., Hopcroft J.E., and Ullmann J.D. Local Search Algorithms. In *Data Structures and Algorithms*, pages 336–343. Addison-Wesley, 1983.

[5] IA Basheer and M. Hajmeer. Artificial neural networks: fundamentals, computing, design, and application. *Journal of Microbiological Methods*, 43(1):3–31, 2000.

[6] M. Beechey, D. Gruen, J. Vickery, and Reserve Bank of Australia. *The efficient market hypothesis: a survey*. Citeseer, 2000.

[7] H.G. Beyer and H.P. Schwefel. Evolution strategies–A comprehensive introduction. *Natural computing*, 1(1):3–52, 2002.

[8] J. Brogaard. High frequency trading and its impact on market quality. *Northwestern University Kellogg School of Management Working Paper*, 2010.

[9] II Carson and S. John. Introduction to modeling and simulation. In *Proceedings of the 37th conference on Winter simulation*, pages 16–23. Winter Simulation Conference, 2005.

[10] V. Černỳ. Thermodynamical approach to the traveling salesman problem: An efficient simulation algorithm. *Journal of optimization theory and applications*, 45(1):41–51, 1985.

[11] A. Chaboud, E. Hjalmarsson, C. Vega, and B. Chiquoine. Rise of the machines: Algorithmic trading in the foreign exchange market. *FRB International Finance Discussion Paper No. 980*, 2009.

[12] E.P. Chan. Backtesting. In *Quantitative trading: how to build your own algorithmic trading business*, pages 31–68. Wiley, 2008.

[13] E.P. Chan. The whats, whos, and whys of quantitative trading. In *Quantitative trading: how to build your own algorithmic trading business*, pages 1–9. Wiley, 2008.

[14] K. Chaudhuri and Y. Wu. Mean reversion in stock prices: evidence from emerging markets. *Managerial Finance*, 29(10):22–37, 2003.

[15] W. Cui, A. Brabazon, and M. O'Neill. Efficient trade execution using a genetic algorithm in an order book based artificial stock market. In *Proceedings of the 11th Annual Conference Companion on Genetic and Evolutionary Computation Conference: Late Breaking Papers*, pages 2023–2028. ACM, 2009.

[16] C. Darwin and J. Carroll. *On the origin of species*. Broadview Press, 2003.

[17] G. Deboeck. *Trading on the edge: neural, genetic, and fuzzy systems for chaotic financial markets*. John Wiley & Sons Inc, 1994.

[18] B. Do, R. Faff, and K. Hamza. A new approach to modeling and estimation for pairs trading. In *Proceedings of 2006 Financial Management Association European Conference*, 2006.

[19] R.J. Elliott, J. Van Der Hoek, and W.P. Malcolm. Pairs trading. *Quantitative Finance*, 5(3):271–276, 2005.

[20] R.F. Engle and C.W.J. Granger. Co-integration and error correction: representation, estimation, and testing. *Econometrica: Journal of the Econometric Society*, pages 251–276, 1987.

[21] Eugene F. Fama. Efficient capital markets: A review of theory and empirical work. *The Journal of Finance*, 25(2):pp. 383–417, 1970.

[22] E. Gatev, W.N. Goetzmann, and K.G. Rouwenhorst. Pairs trading: Performance of a relative-value arbitrage rule. *Review of Financial Studies*, 19(3):797, 2006.

[23] D. Ghosh, N. Chakravarti, SS Fatima, M. Wooldridge, and NR Jennings. A competitive local search heuristic for the subset sum problem. *Computers and Operations Research*, 26(3):271–280, 1999.

[24] F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research*, 13(5):533–549, 1986.

[25] F. Glover. Tabu search-part I. *ORSA Journal on computing*, 1(1):190–206, 1989.

[26] F. Glover. Tabu search-part II. *ORSA Journal on computing*, 2(1):4–32, 1990.

[27] F. Glover and H.J. Greenberg. New approaches for heuristic search: A bilateral linkage with artificial intelligence. *European Journal of Operational Research*, 39(2):119–130, 1989.

[28] J. Hasbrouck and G. Saar. Low-latency trading. *Johnson School Research Paper Series*, 2010.

[29] T. Hendershott. Automated Trading.

[30] T. Hendershott, C.M. Jones, and A.J. Menkveld. Does Algorithmic Trading Improve Liquidity? *Journal of Finance, Forthcoming*, 2008.

[31] B.I. Jacobs and K.N. Levy. Long/short equity investing. *The Journal of Portfolio Management*, 20(1):52–63, 1993.

[32] B.I. Jacobs, K.N. Levy, and H.M. Markowitz. Financial market simulation. *Journal of Portfolio Management*, pages 142–152, 2004.

[33] B.I. Jacobs, K.N. Levy, and H.M. Markowitz. Simulating Security Markets in Dynamic and Equilibrium Modes. *Financial Analysts Journal*, 66(5), 2010.

[34] R. Jarrow and P. Protter. A short history of stochastic integration and mathematical finance: the early years, 1880-1970. *Lecture Notes-Monograph Series*, 45:75–91, 2004.

[35] M.C. Jensen. Some anomalous evidence regarding market efficiency. *Journal of Financial Economics*, 6(2/3):95–101, 1978.

[36] David S. Johnson, Christos H. Papadimtriou, and Mihalis Yannakakis. How easy is local search? *J. Comput. Syst. Sci.*, 37:79–100, August 1988.

[37] S.M. Kakade, M. Kearns, Y. Mansour, and L.E. Ortiz. Competitive algorithms for VWAP and limit order trading. In *Proceedings of the 5th ACM conference on Electronic commerce*, pages 189–198. ACM, 2004.

[38] M. Kearns and L. Ortiz. The penn-lehman automated trading project. *Intelligent Systems, IEEE*, 18(6):22–31, 2005.

[39] S. Kirkpatrick, D.G. Jr., and M.P. Vecchi. Optimization by simmulated annealing. *science*, 220(4598):671–680, 1983.

[40] T.J. Linsmeier, N.D. Pearson, University of Illinois at Urbana-Champaign. Dept. of Agricultural, Consumer Economics, University of Illinois at Urbana-Champaign. Dept. of Accountancy, and University of Illinois at Urbana-Champaign. Dept. of Finance. Risk measurement: An introduction to value at risk. 1996.

[41] B.G. Malkiel. The efficient market hypothesis and its critics. *The Journal of Economic Perspectives*, 17(1):59–82, 2003.

[42] F. Modigliani and L. Modigliani. Risk-adjusted performance. *The Journal of Portfolio Management*, 23(2):45–54, 1997.

[43] NASDAQ OMX Nordic. *Market Model*, 2.1 edition, 2011.

[44] P. Nath. High Frequency Pairs Trading with US Treasury Securities: Risks and Rewards for Hedge Funds. *SSRN eLibrary*, 2003.

[45] C. Neely, P. Weller, and R. Dittmar. Is technical analysis in the foreign exchange market profitable? A genetic programming approach. *Journal of Financial and Quantitative Analysis*, 32(04):405–426, 1997.

[46] J. Niemeyer and P. Sandas. An empirical analysis of the trading structure at the Stockholm Stock Exchange. *Journal of Multinational Finance Management*, 3(3):63–101, 1994.

[47] I. Rechenberg. Evolutionsstrategie–Optimierung technisher Systeme nach Prinzipien der biologischen Evolution. 1973.

[48] S.M. Ross. *An elementary introduction to mathematical finance: options and other topics*. Cambridge University Press, 2003.

[49] P. Samuelson. Proof that properly anticipated prices fluctuate randomly. *Management Review*, 6(2):41–49, 1965.

[50] H.P. Schwefel. Evolution strategies for numerical optimization. In *Evolution and optimum seeking*, number v. 1 in Sixth-generation computer technology series, pages 105–151. Wiley, 1995.

[51] H.P. Schwefel. Tabu search and other hybrid concepts. In *Evolution and optimum seeking*, number v. 1 in Sixth-generation computer technology series, pages 162–164. Wiley, 1995.

[52] W.F. Sharpe. Mutual fund performance. *The Journal of Business*, 39(1):119–138, 1966.

[53] G.C. Silaghi and V. Robu. An agent strategy for automated stock market trading combining price and order book information. In *Computational Intelligence Methods and Applications, 2005 ICSC Congress on*, pages 4–pp. IEEE.

[54] N. Svangard, S. Lloyd, P. Nordin, and C. Wihlborg. Evolving short-term trading strategies using genetic programming. In *Proceedings of the 2002 Congress on Evolutionary Computation CEC2002*, pages 2006–2010, 2002.

[55] P. Temperton. Trading with the help of ”guerrillas” and ”snipers”. *Financial Times Fund Management*, 2007.

[56] E.P.K. Tsang and S. Martinez-Jaramillo. Computational finance. *IEEE Computational Intelligence Society Newsletter*, 3(8), 2004.

[57] R.J.M. Vaessens, E.H.L. Aarts, and J.K. Lenstra. Job shop scheduling by local search. In *INFORMS Journal on Computing*. Citeseer, 1994.

[58] G. Vattekar. ADATE User Manual, 2006.

[59] G. Vidyamurthy. *Pairs Trading: quantitative methods and analysis*. J. Wiley, 2004.

[60] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, 1999.

[61] W. Zhang and M. Looks. A novel local search algorithm for the traveling salesman problem that exploits backbones. In *International Joint Conference on Artificial Intelligence*, volume 19, page 343. Citeseer, 2005.

List of Figures

List of Tables

Listings

List of Algorithms

Appendix A

Glossary

HFT: High-frequency trading.
AT: Algorithmic trading.
QT: Quantitative trading.
VWAP: Volume-Weighted Average Price. The VWAP of a stock over a specified market period is the average price paid per share during that period, so the price of each transaction in the market is weighted by its volume[37].
TWAP: Time Weighted Average Price.
Broker: An individual or firm which operates between a buyer and a seller and usually charge a commission. For most products a license is required.
Dealer: An individual or firm which buys and sells for its own account.
Broker/dealer: An individual or firm buying and selling for itself and others. A registration is required.
Principal: A role of broker/dealer when buying or selling securities for its own account.
Market maker: A brokerage or bank that maintains a firm bid and ask price in a given security by standing ready, willing, and able to buy or sell at publicly quoted prices (called making a market). These firms display bid and offer prices for specific numbers of specific securities, and if these prices are met, they will immediately buy for or sell from their own accounts.
Specialist: A stock exchange member who makes a market for certain exchange-traded securities, maintaining an inventory of those securities and standing ready to buy and sell shares as necessary to maintain an orderly market for those shares. Can be individual, partnership, corporation or group of firms.
Call (periodic) auction: Selling stocks by bid at intervals throughout the day. The orders are stored

for execution at a single market clearing price.

Continuous action: Byers enter competitive bids and sellers place competitive offers simultaneously. Continuous, since orders are executed upon arrival.

Dealership market: Trading occur between principals buying and selling to their own accounts. Firm price quotations are available prior to order submission. **Market clearing price:** The price of goods (securities) or a service at which quantity supplied is equal to quantity demanded. Also called equilibrium price.

Security: A fungible, negotiable instrument representing financial value. Securities are broadly categorized into debt securities (banknotes, bonds, debentures), equity securities (e.g. common stocks) and derivative contracts (forwards, futures, options, swaps).

Market order: Immediate execution at the best price available when the order reaches the marketplace.

Limit order: To execute a transaction only at a specified price (the limit) or better.

Stop order: An order to buy or sell a security once the price of the security has climbed above (or dropped below) a specified stop price.

DMA: Direct Market Access.

ULLDMA: Ultra-Low Latency Direct Market Access.

Synthetic order: Hypothetical order used in the order book simulator that is not a part of the data set.

Position: A general reference to an investment holding. A position can be long or short, and it can be in any asset class, such as stocks, bonds, futures, or options. A position can be open (current) or closed (past), but in general use, unless a position is specifically referred to as closed, the assumption is that it references an open position.

Asset: Any item of economic value owned by an individual or corporation, especially that which could be converted to cash. Examples are cash, securities, accounts receivable, inventory, office equipment, real estate, a car, and other property. On a balance sheet, assets are equal to the sum of liabilities, common stock, preferred stock, and retained earnings. From an accounting perspective, assets are divided into the following categories: current assets (cash and other liquid items), long-term assets (real estate, plant, equipment), prepaid and deferred assets (expenditures for future costs such as insurance, rent, interest), and intangible assets (trademarks, patents, copyrights, goodwill).

All definitions are taken from `http://www.investorwords.com` if not noted otherwise.

Appendix B

Order Book Functions

Listing B.1: Step function in Java code

```
public void step()
{
    if (state == HISTORIC)
        setPreviousChange(time);
    else if (state == SYNTHETIC)
        setPreviousSyntheticChange(time);
    else if (state == BOTH)
    {
        setPreviousChange(time);
        setPreviousSyntheticChange(time);
    }

    // Historic change
    if ((nextChange != null && nextSyntheticChange == null)
            || (nextChange != null && (nextChange.getTime() <
                nextSyntheticChange.getTime())))
    {
        setTime(nextChange);
        setTrades();
        state = HISTORIC;

        if (initialized)
        {
            update();
        }
```

```
        else
        {
            initialize();
            initialized = true;
        }
    }
    // Synthetic change
    else if ((nextChange == null && nextSyntheticChange != null)
            || (nextSyntheticChange != null && (nextChange.getTime() >
                nextSyntheticChange.getTime())))
    {
        setTime(nextSyntheticChange);
        actual = time;
        trades = new TradeList();
        addSyntheticOrders(syntheticOrders.getOrders(time));
        state = SYNTHETIC;
    }
    // Both historic and synthetic
    else
    {
        setTime(nextChange);
        setTrades();

        if (!worstCase)
            addSyntheticOrders(syntheticOrders.getOrders(time));

        state = BOTH;

        if (initialized)
        {
            update();
        }
        else
        {
            initialize();
            initialized = true;
        }

        if (worstCase)
            addSyntheticOrders(syntheticOrders.getOrders(time));
    }
```

```
    setNextChange();
    setNextSyntheticChange();
}
```

Listing B.2: Initialize function in Java code

```
private void initialize()
{
    previousOrderRowList = orderRowList;
    orderRowList = dataAccess.getOrders(ticker, time);

    if (!orderRowList.isEmpty())
    {
        actual = orderRowList.getFirst().getTimeStampSource();
        updates++;

        for (OrderRow orderRow : orderRowList)
        {
            if (orderRow.getBidPrice() != null)
            {
                BigDecimal buyPrice = orderRow.getBidPrice();
                Order buyOrder = new Order(newBuyOrderID(),
                                           Type.BUY,
                                           false,
                                           orderRow.getBidDepth(),
                                           buyPrice,
                                           orderRow.getBidNumber(),
                                           orderRow.getTimeStampSource());

                if (buyOrders.containsKey(buyPrice))
                {
                    buyOrders.get(buyPrice).add(buyOrder);
                }
                else
                {
                    OrderList buy = new OrderList();
                    buy.add(buyOrder);
                    buyOrders.put(buyPrice, buy);
                }
            }
```

```
            if (orderRow.getOfferPrice() != null)
            {
                BigDecimal sellPrice = orderRow.getOfferPrice();
                Order sellOrder = new Order(newSellOrderID(),
                                            Type.SELL,
                                            false,
                                            orderRow.getOfferDepth(),
                                            sellPrice,
                                            orderRow.getOfferNumber(),
                                            orderRow.getTimeStampSource());


                if (sellOrders.containsKey(sellPrice))
                {
                    sellOrders.get(sellPrice).add(sellOrder);
                }
                else
                {
                    OrderList sell = new OrderList();
                    sell.add(sellOrder);
                    sellOrders.put(sellPrice, sell);
                }

            }
        }
    }
}
```

Listing B.3: Excerpt of update function in Java code

```
OrderList buy = buyOrders.get(buyPrice);

int newBidDepth = orderRow.getBidDepth();
int oldBidDepth = buy.getHistoricalVolume();

int newBidNumber = orderRow.getBidNumber();
int oldBidNumber = buy.getHistoricalNumber();

int bidDepthChange = Math.abs(newBidDepth - oldBidDepth);
int bidNumberChange = Math.abs(newBidNumber - oldBidNumber);

Date newDate = orderRow.getTimeStampSource();
```

```
// CASE 1a & 1b: Bid number increase and bid depth increase
if (newBidNumber > oldBidNumber && newBidDepth > oldBidDepth)
{
    if (bidNumberChange == 1)
    {
        event = new Event(Event.CASE_1A, Type.BUY);
    }
    else
    {
        event = new Event(Event.CASE_1B, Type.BUY);
    }

    buy.add(new Order(newBuyOrderID(),
                      Type.BUY,
                      false,
                      bidDepthChange,
                      buyPrice,
                      bidNumberChange,
                      newDate));

    event.setSolution(Event.SOLUTION_A);
    recordStatistic(event);
}
// CASE 2: Bid number increase and bid depth decrease
else if (newBidNumber > oldBidNumber && newBidDepth < oldBidDepth)
{
    event = new Event(Event.CASE_2, Type.BUY);

    int[] merges = { 0 };
    int solution = removeVolume(buy, bidDepthChange, merges);
    addNumberLast(buy, bidNumberChange, newDate);

    if (merges[0] == 0 && solution == Event.SOLUTION_A)
        event.setSolution(Event.SOLUTION_A);
    else if (merges[0] == 0 && solution == Event.SOLUTION_B)
        event.setSolution(Event.SOLUTION_B);
    else
    {
        event.setSolution(Event.SOLUTION_C);
        event.setMerges(merges[0]);
```

```
    }

    recordStatistic(event);
}
// CASE 3: Bid number increase and bid depth unchanged
else if (newBidNumber > oldBidNumber && newBidDepth == oldBidDepth)
{
    event = new Event(Event.CASE_3, Type.BUY);

    addNumberLast(buy, bidNumberChange, newDate);

    event.setSolution(Event.SOLUTION_A);
    recordStatistic(event);
}
// CASE 4: Bid number decrease and bid depth increase
else if (newBidNumber < oldBidNumber && newBidDepth > oldBidDepth)
{
    event = new Event(Event.CASE_4, Type.BUY);

    int[] merges = { 0 };
    int solution;

    if (buy.getHistoricalNumber() > bidDepthChange + 1)
    {
        solution = removeNumber(buy, bidNumberChange + 1, merges);
        buy.add(new Order(newBuyOrderID(),
                          Type.BUY,
                          false,
                          bidDepthChange,
                          buyPrice,
                          1,
                          newDate));

        if (merges[0] == 0 && solution == Event.SOLUTION_A)
            event.setSolution(Event.SOLUTION_A);
        else
        {
            event.setSolution(Event.SOLUTION_B);
            event.setMerges(merges[0]);
        }
    }
```

```
    else
    {
        solution = removeNumber(buy, bidNumberChange, merges);
        addVolumeLast(buy, bidDepthChange, newDate);

        if (merges[0] == 0 && solution == Event.SOLUTION_A)
            event.setSolution(Event.SOLUTION_C);
        else
        {
            event.setSolution(Event.SOLUTION_D);
            event.setMerges(merges[0]);
        }
    }

    recordStatistic(event);
}
// CASE 5a & 5b: Bid number decrease and bid depth decrease
else if (newBidNumber < oldBidNumber && newBidDepth < oldBidDepth)
{
    int[] merges = { 0 };
    int solution = remove(buy, bidDepthChange, bidNumberChange, merges);

    if (bidNumberChange == 1)
    {
        event = new Event(Event.CASE_5A, Type.BUY);

        if (merges[0] == 0 && solution == Event.SOLUTION_A)
            event.setSolution(Event.SOLUTION_A);
        else if (merges[0] == 0 && solution == Event.SOLUTION_B)
            event.setSolution(Event.SOLUTION_B);
        else
        {
            event.setSolution(Event.SOLUTION_C);
            event.setMerges(merges[0]);
        }
    }
    else
    {
        event = new Event(Event.CASE_5B, Type.BUY);

        if (solution == Event.SOLUTION_A)
```

```
            event.setSolution(Event.SOLUTION_A);
        else
            event.setSolution(Event.SOLUTION_B);
    }

    recordStatistic(event);
}
// CASE 6: Bid number decrease and bid depth unchanged
else if(newBidNumber < oldBidNumber && newBidDepth == oldBidDepth)
{
    event = new Event(Event.CASE_6, Type.BUY);

    int[] merges = { 0 };
    int solution = removeNumber(buy, bidNumberChange, merges);

    if (merges[0] == 0 && solution == Event.SOLUTION_A)
        event.setSolution(Event.SOLUTION_A);
    else
    {
        event.setSolution(Event.SOLUTION_B);
        event.setMerges(merges[0]);
    }

    recordStatistic(event);
}
// CASE 7: Bid number unchanged and bid depth increase
else if (newBidNumber == oldBidNumber && newBidDepth > oldBidDepth)
{
    event = new Event(Event.CASE_7, Type.BUY);

    if (buy.getHistoricalNumber() > 1)
    {
        int[] merges = { 0 };
        int solution = removeNumber(buy, 1, merges);
        buy.add(new Order(newBuyOrderID(),
                          Type.BUY,
                          false,
                          bidDepthChange,
                          buyPrice,
                          1,
                          newDate));
```

```
            if (merges[0] == 0 && solution == Event.SOLUTION_A)
                event.setSolution(Event.SOLUTION_A);
            else
            {
                event.setSolution(Event.SOLUTION_B);
                event.setMerges(merges[0]);
            }
        }
        else
        {
            addVolumeLast(buy, bidDepthChange, newDate);

            event.setSolution(Event.SOLUTION_C);
        }

        recordStatistic(event);
    }
    // CASE 8: Bid number unchanged and bid depth decrease
    else if (newBidNumber == oldBidNumber && newBidDepth < oldBidDepth)
    {
        event = new Event(Event.CASE_8, Type.BUY);

        if (buy.getHistoricalNumber() > 1 || buy.containsSynthetic())
        {
            int[] merges = { 0 };
            int solution = removeVolume(buy, bidDepthChange, merges);

            if (merges[0] == 0 && solution == Event.SOLUTION_A)
                event.setSolution(Event.SOLUTION_A);
            else if (merges[0] == 0 && solution == Event.SOLUTION_B)
                event.setSolution(Event.SOLUTION_B);
            else
            {
                event.setSolution(Event.SOLUTION_C);
                event.setMerges(merges[0]);
            }
        }
        else
        {
            buy = new OrderList();
```

```
        buy.add(new Order(newBuyOrderID(),
                          Type.BUY,
                          false,
                          orderRow.getBidDepth(),
                          buyPrice,
                          orderRow.getBidNumber(),
                          orderRow.getTimeStampSource()));

        buyOrders.put(buyPrice, buy);

        event.setSolution(Event.SOLUTION_D);
    }

    recordStatistic(event);
}
// CASE 9: No change
else
{
    event = new Event(Event.CASE_9, Type.BUY);
    event.setSolution(Event.SOLUTION_A);
    recordStatistic(event);
}
```

Listing B.4: Cleanup function in Java code

```
private void cleanup()
{
    // Clean-up and remove orders not in the updated order book
    for (OrderRow previousOrderRow : previousOrderRowList)
    {
        if (previousOrderRow.getBidPrice() != null)
        {
            for (int i = 0; i < orderRowList.size(); i++)
            {
                if (orderRowList.get(i).getBidPrice() != null)
                {
                    BigDecimal lastPrice = previousOrderRow.getBidPrice();
                    BigDecimal newPrice = orderRowList.get(i).getBidPrice();
                    if (newPrice.compareTo(lastPrice) == 0)
                        break;
                    if (newPrice.compareTo(lastPrice) < 0)
                    {
```

```
                    OrderList buy = buyOrders.get(lastPrice);
                    // The order list contains simulated orders, remove
                        all not simulated
                    if (buy.containsSynthetic())
                        buy.removeNotSynthetic();
                    // Remove the entire price level
                    else
                        buyOrders.remove(lastPrice);
                    break;
                }
                // No same price, no less price => does not exist
                if (i == orderRowList.size() - 1)
                {
                    OrderList buy = buyOrders.get(lastPrice);
                    if (buy.containsSynthetic())
                        buy.removeNotSynthetic();
                    else
                        buyOrders.remove(lastPrice);
                    break;
                }
            }
        }// end updated for
    }

    if (previousOrderRow.getOfferPrice() != null)
    {
        for (int i = 0; i < orderRowList.size(); i++)
        {
            if (orderRowList.get(i).getOfferPrice() != null)
            {
                BigDecimal lastPrice = previousOrderRow.getOfferPrice();
                BigDecimal newPrice = orderRowList.get(i).getOfferPrice()
                    ;
                if (newPrice.compareTo(lastPrice) == 0)
                    break;
                if (newPrice.compareTo(lastPrice) > 0)
                {
                    OrderList sell = sellOrders.get(lastPrice);
                    // The order list contains simulated orders, remove
                        all not simulated
                    if (sell.containsSynthetic())
```

```
                        sell.removeNotSynthetic();
                    // Remove the entire price level
                    else
                        sellOrders.remove(lastPrice);
                    break;
                }
                // No same price, no less price => does not exist
                if (i == orderRowList.size() - 1)
                {
                    OrderList sell = sellOrders.get(lastPrice);
                    if (sell.containsSynthetic())
                        sell.removeNotSynthetic();
                    else
                        sellOrders.remove(lastPrice);
                    break;
                }
            }
        }// end updated for
    }
}// end clean-up
}
```

Listing B.5: Remove function in Java code

```
private int remove(OrderList orderList, int volume, int number, int[] merges)
{
    // Number decrease with one
    if (number == 1)
    {
        // Look for a regular order with exact volume
        OrderList removeList = orderList.getOrdersEQ(volume, 1); // regular

        // Remove order with this volume if it exists
        if (removeList != null)
        {
            Order remove = removeList.get(Library.randomInteger(0, removeList
                .size()));
            orderList.remove(remove);

            return Event.SOLUTION_A;
        }
```

```
        // There was no order with this volume, remove this amount from one
            or more orders
        else
        {
            removeList = orderList.getOrdersGR(volume, 1); // composite

            if (removeList != null)
            {
                Order remove = removeList.get(Library.randomInteger(0,
                    removeList.size()));
                Order replaceWith = new Order(remove.getID(),
                                              remove.getType(),
                                              remove.isSynthetic(),
                                              remove.getVolume() - volume,
                                              remove.getPrice(),
                                              remove.getNumber() - 1,
                                              remove.getTime());

                orderList.replace(remove, replaceWith);

                return Event.SOLUTION_B;
            }
            else
            {
                mergeOrders(orderList);
                merges[0]++;
                remove(orderList, volume, number, merges);

                return Event.SOLUTION_C;
            }
        }
    }
    // Number decrease with more than one (subset sum problem)
    else
    {
        OrderList notSynthetic = orderList.getNotSynthetic();
        OrderList solution = exactSubset(notSynthetic, volume);
        int sol;

        // There was an exact solution (disregarding number)
        if (solution != null)
```

```
        {
            orderList.removeAll(solution);

            sol = Event.SOLUTION_A;
        }
        // There wasn't an exact solution
        else
        {
            solution = approximateSubset(notSynthetic, volume, 1000);
            int removeVolume = solution.getHistoricalVolume();
            int diff = removeVolume - volume;

            // The solution was too big
            if (removeVolume > volume)
            {
                OrderList removeList = solution.getOrdersGREQ(Math.abs(diff),
                    0);
                assert(removeList != null);
                Order remove = removeList.minVolume();
                solution.remove(remove);
                diff = solution.getHistoricalVolume() - volume;

                orderList.removeAll(solution);
                removeVolume(orderList, Math.abs(diff), merges);
            }
            // The solution was too small
            else
            {
                orderList.removeAll(solution);
                removeVolume(orderList, Math.abs(diff), merges);
            }

            sol = Event.SOLUTION_B;
        }

        // Check if number is correct after subset is removed
        int removeNumber = solution.getHistoricalNumber();

        if (removeNumber != number)
        {
            int diff = Math.abs(removeNumber - number);
```

```
            if (removeNumber < number)
                removeNumber(orderList, diff, merges);
            else
                addNumber(orderList, diff);
        }

        return sol;
    }
}
```

Listing B.6: MergeOrders function in Java code

```
private void mergeOrders(OrderList orderList)
{
    int N = orderList.size();
    Order first, second = null;

    OrderList composite = orderList.getComposite();
    OrderList notSynthetic = orderList.getNotSynthetic();

    assert(notSynthetic.size() > 1) : notSynthetic;

    if (composite != null)
        first = composite.get(Library.randomInteger(0, composite.size()));
    else
        first = notSynthetic.get(Library.randomInteger(0, notSynthetic.size()
            ));

    int indexFirst = orderList.indexOf(first);
    int indexAfter = -1;
    int indexBefore = -1;

    // Find the next non synthetic order
    for (int i = indexFirst + 1; i < N; i++)
    {
        if (!orderList.get(i).isSynthetic())
        {
            indexAfter = i;
            break;
        }
    }
```

```
// Find the previous non synthetic order
for (int i = indexFirst - 1; i >= 0; i--)
{
    if (!orderList.get(i).isSynthetic())
    {
        indexBefore = i;
        break;
    }
}

if (indexAfter != -1 && indexBefore != -1)
{
    if (Library.randomBoolean())
    {
        second = orderList.get(indexAfter);
    }
    else
    {
        second = orderList.get(indexBefore);
    }
}
else
{
    if (indexAfter != -1)
    {
        second = orderList.get(indexAfter);
    }
    else if (indexBefore != -1)
    {
        second = orderList.get(indexBefore);
    }
}

assert(first != null && second != null) : first + "\n" + second;

orderList.remove(second);

try
{
    Order replaceWith = new Order(first.getType() == Type.BUY ?
        newBuyOrderID() : newSellOrderID(),
```

```
                                        first.getType(),
                                        first.isSynthetic(),
                                        first.getVolume() + second.getVolume(),
                                        first.getPrice(),
                                        first.getNumber() + second.getNumber(),
                                        first.getTime()); // Date of first

            orderList.replace(first, replaceWith);
        }
        catch (NullPointerException npe)
        {
            System.out.println(first);
            System.out.println(second);
        }
    }
```

Listing B.7: RemoveTrades function in Java code

```
private void removeTradedOrders()
{
    // Remove trades
    if (!trades.isEmpty())
    {
        // Check for equilibrium time
        if (trades.getTrades(Library.stringToDate(day + " 09:00:00")) != null
            )
        {
            // TODO Equilibrium handling (also end equilibrium)
        }
        else
        {
            for (Trade trade : trades)
            {
                BigDecimal tradePrice = trade.getPrice();
                int tradeVolume = trade.getQuantity();

                OrderRow previousOrderRow;
                OrderRow currentOrderRow;

                if (!trade.sourceIsUnknown())
                {
```

```
                previousOrderRow = previousOrderRowList.getOrderRow(
                    tradePrice, Type.BUY);
                currentOrderRow = orderRowList.getOrderRow(tradePrice,
                    Type.BUY);

                if (buyOrders.containsKey(tradePrice))
                {
                    // Postpone removal to next change if the previous
                        depth is equal to the current depth
                    if ((previousOrderRow != null && currentOrderRow !=
                        null)
                            && (previousOrderRow.getBidDepth() ==
                                currentOrderRow.getBidDepth()))
                    {
                        previousTrades.add(trade);
                        continue;
                    }

                    OrderList buy = buyOrders.get(tradePrice);

                    if (buy.getTotalVolume() > tradeVolume)
                    {
                        buy.removeVolumeFromStart(tradeVolume, trade,
                            results);
                        continue;
                    }
                    else
                    {
                        previousTrades.add(trade);
                        continue;
                    }
                }
                else if (sellOrders.containsKey(tradePrice))
                {
                    // Postpone removal to next change if the previous
                        depth is equal to the current depth
                    if ((previousOrderRow != null && currentOrderRow !=
                        null)
                            && (previousOrderRow.getOfferDepth() ==
                                currentOrderRow.getOfferDepth()))
                    {
```

```
                        previousTrades.add(trade);
                        continue;
                    }

                    OrderList sell = sellOrders.get(tradePrice);

                    if (sell.getTotalVolume() > tradeVolume)
                    {
                        sell.removeVolumeFromStart(tradeVolume, trade,
                            results);
                        continue;
                    }
                    else
                    {
                        previousTrades.add(trade);
                        continue;
                    }
                }
                else
                {
                    previousTrades.add(trade);
                    continue;
                }
            }
            else
            {
                // TODO Unknown source, ignore this rare event?
            }
        }
    }
}
```

Appendix C

Order Book for ABB

Listing C.1: Order book for ABB at 14:33:54 on the 1st of April 2011 according to the order book simulator

```
-------------------------------------------------------------------------------
ORDER BOOK:

Day:        2011-04-01
Ticker:     ABB
Time:       14:33:54

Previous:   14:33:49 (00:00:00)
Next:       14:33:56 (END)
Updates:    4085 (historic)

BUY:                                    SELL:

  153.40:                                 153.50:
     127,  1, 107914, 14:32:45               4818,  1, 507236, 14:31:43
     250,  1, 107926, 14:32:58               6002,  1, 507213, 14:32:21
     122,  1, 107955, 14:33:29                 59,  1, 507214, 14:32:25
                                            3000,  1, 507248, 14:33:08
                                             500,  1, 507252, 14:33:09
                                            2043,  1, 507257, 14:33:26
                                             600,  1, 507260, 14:33:28
                                           10441,  1, 507262, 14:33:29
                                            6025,  2, 507282, 14:33:47

    =499,  3                               =33488, 10
```

```
153.30:
  1359,  1, 107882, 14:32:17
  1936,  7, 107950, 14:32:45
  2667,  1, 107930, 14:32:58
  1078,  1, 107939, 14:33:16
  1002,  1, 107946, 14:33:21
   235,  1, 107962, 14:33:32
   102,  1, 107974, 14:33:38
   244,  2, 107975, 14:33:42
   142,  1, 107985, 14:33:54

 =8765, 16

153.20:
  1165,  1, 107802, 14:31:05
  6534,  1, 107812, 14:31:14
   600,  1, 107836, 14:31:31
   935,  1, 107858, 14:31:54
    40,  1, 107859, 14:31:57
   222,  7, 107982, 14:32:58
 11400,  1, 107976, 14:33:42

=20896, 13

153.10:
  2774,  3, 107890, 14:09:52
 16740,  1, 107838, 14:31:43
   703,  1, 107863, 14:32:00
   897,  1, 107887, 14:32:19
   379,  1, 107922, 14:32:51
  2000,  1, 107935, 14:33:08
  1000,  2, 107949, 14:33:26
   942,  1, 107970, 14:33:37
 15534,  1, 107983, 14:33:47
   166,  1, 107986, 14:33:54

=41135, 13

153.00:
```

```
153.60:
  3848,  4, 507272, 14:31:43
  2352,  7, 507276, 14:33:02
  1400,  2, 507263, 14:33:29
 13000,  1, 507268, 14:33:37
 15025,  2, 507283, 14:33:47

=35625, 16

153.70:
 13306,  4, 507285, 14:32:25
   644,  1, 507230, 14:32:42
   634,  1, 507242, 14:32:58
   270,  1, 507245, 14:33:02
  1200,  1, 507256, 14:33:16
  2000,  1, 507266, 14:33:34
  5000,  2, 507286, 14:33:42
    25,  1, 507287, 14:33:49

=23079, 12

153.80:
  1651,  1, 507217, 14:30:42
   732,  1, 507155, 14:31:14
  6618,  1, 507164, 14:31:22
  2674,  1, 507186, 14:31:51
  2000,  1, 507238, 14:32:55
  2000,  1, 507250, 14:33:08
    25,  1, 507288, 14:33:49

=15700,  7

153.90:
```

```
          50,  2, 107708, 12:09:13          1752,  1, 507055, 14:29:21
        1544,  1, 107804, 14:31:05           395,  5, 507177, 14:30:03
        6000,  1, 107814, 14:31:14          1440,  2, 507127, 14:30:42
         871,  1, 107839, 14:31:43           997,  1, 507140, 14:30:55
        2056,  1, 107884, 14:32:17          8467,  1, 507156, 14:31:14
          60,  1, 107908, 14:32:38          1200,  1, 507162, 14:31:20
       30065,  1, 107957, 14:33:29          1533,  1, 507225, 14:32:36

      =40646,  8                           =15784, 12

      152.90:                              154.00:
        1144,  1, 107822, 14:29:56          4450,  5, 507195, 14:30:18
         381,  1, 107795, 14:31:00           200,  1, 507138, 14:30:45
         600,  1, 107805, 14:31:05          6000,  1, 507157, 14:31:14
        5494,  1, 107815, 14:31:14
        2031,  1, 107840, 14:31:43
        1553,  1, 107903, 14:32:36
         474,  1, 107987, 14:33:54

      =11677,  7                           =10650,  7

      152.80:                              154.10:
         775,  1, 107899, 14:30:18           250,  1, 507086, 14:29:11
         977,  1, 107867, 14:32:03          1000,  2, 507148, 14:30:03
        5110,  1, 107941, 14:32:07           200,  1, 507110, 14:30:18
         367,  1, 107893, 14:32:28          1200,  1, 507116, 14:30:20
         218,  1, 107904, 14:32:36          4000,  1, 507129, 14:30:42
         600,  1, 107958, 14:33:29          4168,  1, 507153, 14:31:05
        1580,  1, 107973, 14:33:37

       =9627,  7                           =10818,  7

      152.70:                              154.20:
        1000,  1, 107743, 14:30:02           550,  1, 507117, 13:03:20
         121,  1, 107909, 14:32:38           900,  1, 507188, 13:10:44
        6000,  1, 107936, 14:33:08          1100,  1, 507111, 14:30:18
                                             200,  1, 507125, 14:30:22
                                            2800,  2, 507200, 14:31:57

       =7121,  3                            =5550,  6
```

```
        152.60:                                 154.30:
          1156,  3, 107558, 14:10:33              50,  2, 507132, 14:29:30
          6000,  1, 107984, 14:33:49

         =7156,  4

        152.50:                                 154.40:
          1574,  6, 107960, 08:00:39              147,  1, 505852, 13:01:39
          1264,  1, 107947, 14:33:21              113,  1, 506635, 14:10:43

         =2838,  7                               =260,  2

        152.40:                                 154.50:
             7,  1, 107335, 14:15:57            8233,  3, 505414, 12:23:12
                                                1628,  1, 505221, 12:43:47
                                                 321,  1, 505247, 12:45:14
                                                 495,  1, 505248, 12:45:22
                                                 135,  1, 505250, 12:45:29

                                               =10812,  7

        152.10:                                 154.60:
           200,  1, 106373, 13:22:31            2239,  2, 505941, 13:19:11
                                                 500,  1, 505980, 13:26:45
                                                 121,  1, 506967, 14:26:20
                                                 121,  1, 507229, 14:32:40

                                                =2981,  5

        152.00:                                 154.70:
           100,  1, 107246, 14:13:44            8462,  1, 506034, 09:31:34
                                                 560,  1, 506050, 13:28:12
                                                2478,  1, 506035, 13:29:58

                                               =11500,  3

        151.80:                                 154.80:
         35006,  1, 107843, 14:31:43            5307,  1, 506683, 14:13:01
                                                 243,  1, 506674, 14:13:07

                                                =5550,  2
```

```
151.60:                                 154.90:
    60,  1, 106013, 13:03:31              6360,  5, 504426, 11:52:14

151.50:                                 155.00:
 10480,  3, 106148, 13:11:42            16572,  8, 504620, 12:01:58
    39,  1, 107965, 14:33:33              364,  1, 505066, 12:33:43
                                          273,  1, 506223, 13:46:03
                                         1000,  1, 506307, 13:50:43
                                         1339,  1, 506368, 13:54:41
                                          421,  1, 506382, 13:55:51
                                          843,  1, 506383, 13:55:55
                                           73,  1, 506387, 13:55:59
                                          211,  1, 506392, 13:56:09
                                         1295,  1, 506443, 13:58:51
                                          464,  1, 506450, 13:59:10
                                          395,  1, 506455, 13:59:23
                                          214,  1, 506458, 13:59:32
                                          788,  1, 506466, 14:00:02
                                         1011,  1, 506490, 14:02:39
                                          260,  1, 506495, 14:03:14
                                          808,  1, 506498, 14:03:17
                                          264,  1, 506506, 14:03:32
                                          726,  1, 506537, 14:05:58

=10519,  4                              =27321, 26

151.20:                                 155.10:
    84,  1, 107827, 14:31:05              200,  1, 506924, 12:58:59

151.10:                                 155.20:
   400,  1, 107807, 14:31:05             1000,  1, 507098, 14:29:51

151.00:                                 155.30:
  1550,  2, 107808, 14:31:05              778,  1, 506636, 14:10:43
                                         2830,  1, 507179, 14:31:43

                                        =3608,  2

150.80:                                 155.40:
   700,  1, 107966, 14:33:33             1439,  1, 507270, 14:33:37
```

Appendix D

Preliminary Test

Listing D.1: Results from the order book simulator for HM-B - 1st of April 2011 - limit order for 100 shares at 209.40SEK - issued at 11:07:50

```
------------------------------------------------------------------------------
ORDER BOOK:

Day:          2011-04-01
Ticker:       HM-B
Time:         11:07:50

Previous:     11:07:47 (00:00:00)
Next:         11:07:52 (END)
Updates:      1710 (both)

BUY:                                    SELL:

  208,90:                                 209,00:
     102,   1, 103324, 11:06:54              220,   1, 503387, 11:05:47
     223,   1, 103327, 11:07:16              579,   2, 503388, 11:06:12
     400,   1, 103333, 11:07:39              481,   1, 503393, 11:06:42
     400,   1, 103334, 11:07:41             1373,   3, 503404, 11:06:54
    3059,   1, 103337, 11:07:50               99,   1, 503406, 11:07:16
                                              81,   2, 503412, 11:07:41
                                             426,   1, 503413, 11:07:47
                                             314,   1, 503417, 11:07:50

   =4184,   5                              =3573,  12
```

```
208,80:
  1216,  2, 103300, 11:01:04
   195,  1, 103269, 11:04:27
  1291,  1, 103282, 11:04:39
   192,  1, 103285, 11:04:50
   100,  1, 103299, 11:04:54
   840,  1, 103295, 11:05:12
   301,  1, 103302, 11:05:34
  1500,  1, 103317, 11:06:42
   400,  1, 103322, 11:06:50
   460,  1, 103335, 11:07:41

 =6495, 11

208,70:
   706,  1, 103318, 11:00:21
   900,  1, 103221, 11:02:06
  1450,  1, 103226, 11:02:35
   300,  1, 103236, 11:03:11
   120,  1, 103238, 11:03:17
   300,  1, 103240, 11:03:26
   626,  1, 103245, 11:03:36
   900,  1, 103256, 11:04:11
   884,  1, 103292, 11:04:57
  3196,  1, 103326, 11:06:56
  1989,  2, 103328, 11:07:18

=11371, 12

208,60:
   316,  1, 103137, 10:30:22
     3,  1, 103307, 10:39:30
   184,  1, 103120, 10:57:17
   541,  1, 103201, 11:01:21
  2737,  1, 103205, 11:01:30
   256,  1, 103215, 11:01:51
   500,  1, 103257, 11:04:11
   300,  1, 103270, 11:04:27
  1865,  1, 103284, 11:04:39
  1431,  1, 103288, 11:04:54

209,10:
  1025,  1, 503333, 11:04:11
  3675,  7, 503369, 11:04:14
  1000,  1, 503375, 11:05:05
  2796,  1, 503390, 11:06:32
  1141,  1, 503402, 11:06:50
   100,  1, 503414, 11:07:47

 =9737, 12

209,20:
   127,  1, 503327, 11:04:00
   358,  1, 503329, 11:04:01
  1370,  1, 503380, 11:04:14
  1456,  1, 503381, 11:05:27
  2187,  2, 503394, 11:06:42
  1000,  2, 503398, 11:06:44
  2556,  1, 503410, 11:07:27

 =9054,  9

209,30:
   454,  1, 503320, 10:54:23
   500,  1, 503238, 11:00:59
  1300,  4, 503399, 11:01:21
  1000,  1, 503292, 11:02:44
   700,  1, 503340, 11:04:14
  1636,  1, 503349, 11:04:20
  1736,  1, 503371, 11:04:54
   104,  2, 503409, 11:07:26
   100,  1, 503415, 11:07:47
```

```
     52,  1, 103293, 11:04:57
   1500,  1, 103297, 11:05:27
    131,  1, 103332, 11:07:32
    201,  1, 103336, 11:07:47

  =10017, 14                          =7530, 13

  208,50:                             209,40:
   3000,  1, 101959, 09:53:25          1800,  1, 503252, 11:01:12
   2009,  1, 102474, 10:12:56          2151,  4, 503400, 11:04:34
     88,  1, 102428, 10:13:47           200,  1, 503395, 11:06:42
    597,  4, 103179, 11:00:54           100,  1, -1, 11:07:50*

   =1594,  6                          =5200,  2
```

```
ORDER BOOK:

Day:        2011-04-01
Ticker:     HM-B
Time:       11:48:55

Previous:   11:48:49 (11:07:50)
Next:       11:49:01 (END)
Updates:    2135 (historic)

BUY:                                SELL:

  209,30:                             209,40:
   3639,  4, 103976, 11:48:55          100,  1, -1, 11:07:50*
                                       426,  1, 503964, 11:37:06
                                        40,  2, 504097, 11:39:33
                                       929,  1, 504114, 11:46:21
                                      1200,  1, 504116, 11:46:23
                                        48,  1, 504126, 11:46:57
                                      2309,  4, 504143, 11:48:04
                                        68,  1, 504150, 11:48:42
                                      2776,  2, 504152, 11:48:55

                                     =7896, 14
```

```
-------------------------------------------------------------------------
TRADES:

  BUYER:

    209,40: 260     (11:49:39) B, Auto trade
    Total: 260

-------------------------------------------------------------------------
Result [type=SELL, volume=100, price=209.4000,
        time=Fri Apr 01 11:49:39 CEST 2011, partial=false, orderID=-1]
-------------------------------------------------------------------------
ORDER BOOK:

Day:        2011-04-01
Ticker:     HM-B
Time:       11:49:39

Previous:   11:49:33 (11:07:50)
Next:       11:49:51 (END)
Updates:    2141 (historic)

BUY:                                    SELL:

  209,30:                                 209,40:
    2117,   4, 103976, 11:48:55             266,   1, 503964, 11:37:06
    3838,   3, 103981, 11:49:01              40,   2, 504097, 11:39:33
     368,   1, 103986, 11:49:08            1200,   1, 504116, 11:46:23
    1432,   2, 103991, 11:49:33              48,   1, 504126, 11:46:57
                                            585,   4, 504143, 11:48:04
                                           2776,   2, 504152, 11:48:55
                                           1120,   2, 504160, 11:49:39

   =7755, 10                               =6035, 13
```

Appendix E

Test Cases

Listing E.1: GETI-B test case - 24th of April 2011

```
-------------------------------------------------------------------------------
ORDER BOOK:

Day:        2011-02-24
Ticker:     GETI-B
Time:       09:19:40

Previous:   09:19:38 (00:00:00)
Next:       09:19:43 (END)
Updates:    242 (historic)

BUY:                                    SELL:

  148,50:                                 148,80:
    788,  1, 100473, 09:19:35               100,  1, 500420, 09:19:30
                                           1182,  1, 500429, 09:19:40*

                                          =1282,  2


-------------------------------------------------------------------------------
ORDER BOOK:

Day:        2011-02-24
Ticker:     GETI-B
Time:       09:20:12
```

```
Previous:   09:20:04 (00:00:00)
Next:       09:20:14 (END)
Updates:    248 (historic)

BUY:                                    SELL:

  148,50:                                 148,80:
    1338,   1, 100476, 09:19:43             100,   1, 500420, 09:19:30
                                           1182,   1, 500429, 09:19:40*
                                            200,   1, 500437, 09:20:12

                                           =1482,   3

-------------------------------------------------------------------------
ORDER BOOK:

Day:        2011-02-24
Ticker:     GETI-B
Time:       09:20:20

Previous:   09:20:14 (00:00:00)
Next:       09:20:27 (END)
Updates:    250 (historic)

BUY:                                    SELL:

  148,50:                                 148,80:
    1338,   1, 100476, 09:19:43             100,   1, 500420, 09:19:30
                                           1182,   1, 500429, 09:19:40*

                                           =1282,   2

-------------------------------------------------------------------------
TRADES:

  BUYER:

    148,80: 493     (09:20:24) B, Auto trade
    148,80: 100     (09:20:24) B, Auto trade
    Total: 593
```

```
-------------------------------------------------------------------------------
ORDER BOOK:

Day:        2011-02-24
Ticker:     GETI-B
Time:       09:20:27

Previous:   09:20:20 (00:00:00)
Next:       09:20:34 (END)
Updates:    251 (historic)

BUY:                                    SELL:

  148,50:                                 148,80:
    1338,  1, 100476, 09:19:43              689,  1, 500429, 09:19:40*
                                            100,  1, 500441, 09:20:27

                                          =789,  2

-------------------------------------------------------------------------------
TRADES:

  BUYER:

    148,80: 593     (09:21:44) B, Auto trade
    Total: 593

-------------------------------------------------------------------------------
ORDER BOOK:

Day:        2011-02-24
Ticker:     GETI-B
Time:       09:21:44

Previous:   09:21:30 (00:00:00)
Next:       09:21:51 (END)
Updates:    260 (historic)

BUY:                                    SELL:
```

```
  148,50:                                  148,80:
    1338,   1, 100476, 09:19:43              96,   1, 500429, 09:19:40*
     300,   1, 100486, 09:20:37             100,   1, 500441, 09:20:27
     400,   1, 100489, 09:21:44             200,   1, 500456, 09:21:44

   =2038,   3                               =396,   3

-------------------------------------------------------------------------

ORDER BOOK:

Day:        2011-02-24
Ticker:     GETI-B
Time:       09:21:51

Previous:   09:21:44 (00:00:00)
Next:       09:21:54 (END)
Updates:    261 (historic)

BUY:                                     SELL:

  148,50:                                  148,80:
    1338,   1, 100476, 09:19:43              96,   1, 500429, 09:19:40*
     300,   1, 100486, 09:20:37             100,   1, 500441, 09:20:27
     400,   1, 100489, 09:21:44
     769,   2, 100491, 09:21:51

   =2807,   5                               =196,   2

-------------------------------------------------------------------------

TRADES:

  BUYER:

    148,80: 100     (09:22:37) B, Auto trade
    148,80: 96      (09:22:37) B, Auto trade
    Total: 196

-------------------------------------------------------------------------

ORDER BOOK:

Day:        2011-02-24
```

```
Ticker:     GETI-B
Time:       09:22:38

Previous:   09:22:18 (00:00:00)
Next:       09:22:50 (END)
Updates:    269 (historic)

BUY:                                    SELL:

  148,50:                                 148,80:
    1595,   1, 100498, 09:19:43             100,   1, 500464, 09:22:38
     712,   1, 100491, 09:21:51
      31,   1, 100493, 09:21:54
     200,   1, 100499, 09:22:38

   =2538,   4
```

```
Result [type=SELL, volume=393, price=148.8000,
        time=Thu Feb 24 09:20:24 CET 2011, partial=true, orderID=-1]
Result [type=SELL, volume=100, price=148.8000,
        time=Thu Feb 24 09:20:24 CET 2011, partial=true, orderID=-1]
Result [type=SELL, volume=593, price=148.8000,
        time=Thu Feb 24 09:21:44 CET 2011, partial=true, orderID=-1]
Result [type=SELL, volume=96, price=148.8000,
        time=Thu Feb 24 09:22:37 CET 2011, partial=false, orderID=-1]
```

Listing E.2: SWMA test case - 10th of March 2011

```
ORDER BOOK:

Day:        2011-03-10
Ticker:     SWMA
Time:       13:17:00

Previous:   13:16:53 (00:00:00)
Next:       13:17:05 (END)
Updates:    1910 (historic)

BUY:                                    SELL:
```

```
  206,00:                                  206,10:
     306,   1, 102230, 13:17:00*              965,   1, 501968, 13:05:46
                                             1800,   1, 501940, 13:07:22
                                              257,   1, 502015, 13:15:37

                                            =3022,   3
```

```
ORDER BOOK:

Day:        2011-03-10
Ticker:     SWMA
Time:       13:17:05

Previous:   13:17:00 (00:00:00)
Next:       13:17:11 (END)
Updates:    1911 (historic)

BUY:                                     SELL:

  206,00:                                  206,10:
     306,   1, 102230, 13:17:00*              1965,   1, 502034, 13:07:22
     800,   1, 102234, 13:17:05               257,   1, 502015, 13:15:37

   =1106,   2                              =2222,   2
```

```
ORDER BOOK:

Day:        2011-03-10
Ticker:     SWMA
Time:       13:17:11

Previous:   13:17:05 (00:00:00)
Next:       13:17:21 (END)
Updates:    1912 (historic)

BUY:                                     SELL:

  206,00:                                  206,10:
     306,   1, 102230, 13:17:00*              1965,   1, 502034, 13:07:22
```

```
     800,  1, 102234, 13:17:05              257,  1, 502015, 13:15:37
     417,  1, 102240, 13:17:11

   =1523,  3                              =2222,  2
```

```
ORDER BOOK:

Day:         2011-03-10
Ticker:      SWMA
Time:        13:17:21

Previous:    13:17:11 (00:00:00)
Next:        13:17:33 (END)
Updates:     1913 (historic)

BUY:                                    SELL:

  206,00:                                 206,10:
     306,  1, 102230, 13:17:00*             1965,  1, 502034, 13:07:22
     800,  1, 102234, 13:17:05              257,  1, 502015, 13:15:37
     417,  1, 102240, 13:17:11
     367,  1, 102243, 13:17:21

   =1890,  4                              =2222,  2
```

```
ORDER BOOK:

Day:         2011-03-10
Ticker:      SWMA
Time:        13:19:32

Previous:    13:19:00 (00:00:00)
Next:        13:19:37 (END)
Updates:     1920 (historic)

BUY:                                    SELL:

  206,00:                                 206,10:
     306,  1, 102230, 13:17:00*             1965,  1, 502034, 13:07:22
```

```
     800,  1, 102234, 13:17:05            257,  1, 502015, 13:15:37
     417,  1, 102240, 13:17:11
     367,  1, 102243, 13:17:21
     676,  1, 102248, 13:19:32

   =2566,  5                          =2222,  2

--------------------------------------------------------------------------------
ORDER BOOK:

Day:        2011-03-10
Ticker:     SWMA
Time:       13:20:23

Previous:   13:20:11 (00:00:00)
Next:       13:20:31 (END)
Updates:    1925 (historic)

BUY:                                   SELL:

  206,00:                              206,10:
     306,  1, 102230, 13:17:00*           1965,  1, 502034, 13:07:22
     800,  1, 102234, 13:17:05             257,  1, 502015, 13:15:37
     417,  1, 102240, 13:17:11             823,  1, 502044, 13:20:11
     367,  1, 102243, 13:17:21

   =1890,  4                          =3045,  3

--------------------------------------------------------------------------------
ORDER BOOK:

Day:        2011-03-10
Ticker:     SWMA
Time:       13:20:40

Previous:   13:20:35 (00:00:00)
Next:       13:20:46 (END)
Updates:    1928 (historic)

BUY:                                   SELL:
```

```
 206,00:                                  206,10:
    306,   1, 102230, 13:17:00*             1965,   1, 502034, 13:07:22
    800,   1, 102234, 13:17:05               257,   1, 502015, 13:15:37
    417,   1, 102240, 13:17:11               823,   1, 502044, 13:20:11
    367,   1, 102243, 13:17:21
   1500,   1, 102256, 13:20:40

  =3390,   5                               =3045,   3
```

TRADES:

SELLER:

```
  206,00: 173     (13:20:53) S, Auto trade
  206,00: 68      (13:20:53) S, Auto trade
  206,00: 133     (13:20:53) S, Auto trade
  206,10: 755     (13:20:53) S, Auto trade
  Total: 1129
```

BUYER:

```
  206,10: 778     (13:20:53) B, Auto trade
  206,10: 1111    (13:20:53) B, Auto trade
  206,10: 100     (13:20:53) B, Auto trade
  206,10: 1111    (13:20:53) B, Auto trade
  206,10: 45      (13:20:53) B, Auto trade
  Total: 3145
```

ORDER BOOK:

```
Day:        2011-03-10
Ticker:     SWMA
Time:       13:20:53

Previous:   13:20:46 (00:00:00)
Next:       13:20:58 (END)
Updates:    1930 (historic)
```

BUY: SELL:

```
 206,00:                                  206,10:
    349,   1, 102257, 13:17:11               51,   1, 502049, 13:20:53
    367,   1, 102243, 13:17:21
   1500,   1, 102256, 13:20:40

  =2216,   3
--------------------------------------------------------------------------------
Result [type=BUY, volume=306, price=206.0000,
        time=Thu Mar 10 13:20:53 CET 2011, partial=false, orderID=-1]
```