Quantitative Finance

ISSN: 1469-7688 (Print) 1469-7696 (Online) Journal homepage: www.tandfonline.com/journals/rquf20

Low-latency liquidity inefficiency strategies

Christian Oesch & Dietmar Maringer

To cite this article: Christian Oesch & Dietmar Maringer (2017) Low-latency liquidity inefficiency strategies, Quantitative Finance, 17:5, 717-727, DOI: 10.1080/14697688.2016.1242765

To link to this article: https://doi.org/10.1080/14697688.2016.1242765

Published online: 04 Nov 2016.

Quantitative Finance, 2017
Vol. 17, No. 5, 717–727, http://dx.doi.org/10.1080/14697688.2016.1242765

Routledge
Taylor & Francis Group

Low-latency liquidity inefficiency strategies

CHRISTIAN OESCH* and DIETMAR MARINGER

Business and Economics Faculty (WWZ), University of Basel, Peter Merian-Weg 6, CH-4002 Basel, Switzerland

(*Received 4 September 2015; accepted 23 September 2016; published online 4 November 2016*)

The vast amount of high-frequency data heralds the use of new methods in financial data analysis and quantitative trading. This study delivers a proof-of-concept for a high frequency-based trading system based on an evolutionary computation method. Motivated by a theoretical liquidity asymmetry theorem from the market microstructure literature, grammatical evolution is used to exploit volume inefficiencies at the bid–ask spread. Using NASDAQ Historical TotalView-ITCH level two limit order book data, execution volumes can be tracked. This allows for testing of the strategies with minimal assumptions. The system evolves profitable and robust strategies with high returns and low risk.

Keywords: Market microstructure; Liquidity; Grammatical evolution

JEL Classification: G17, C53

1. Introduction

While stock returns are hard or even impossible to anticipate, some order book properties exhibit predictable dynamics. One of the recent discoveries of the market microstructure literature is the long memory effect of the order flow process; as first portrayed by Bouchaud *et al*. (2004) and Lillo and Farmer (2004). Due to large orders from institutional traders which are split up into a series of smaller trades, the signs of orders submitted to an exchange exhibit long-range correlation. This means that after a buy (sell) order, the next order is likely to be another buy (sell) order. For prices to be non-predictable, volumes in the order book need to follow a certain distribution. This background knowledge is used in this study to let grammatical evolution (GE) evolve a condition which describes volume distributions in the order book. If the condition is violated, the system places a limit order on the side of the market where the volume is too small, expecting the temporary price impact to be reversed.

There have been several publications using GE, to develop automatic trading systems for stock indices, foreign exchange spot rates as well as single stocks. The first studies, O'Neill *et al*. (2001); Brabazon and O'Neill (2004), use a simple static approach with one fixed training period and a single testing period. Both approaches generate trading signals based on past prices. Simple technical trading indicators function as inputs to the system. By a fuzzy logic decision rule, the system decides to buy, sell or do nothing. A more dynamic approach can be found in Dempsey *et al*. (2006). The authors present an online trading system which retrains strategies over the course of the sample duration.

While most of these studies do produce systems with profitable results, to our knowledge no study has been performed on order book data.

The trading system in this study is trained and tested with NASDAQ order book event data. This makes it possible to track execution volumes, employ bid and ask prices, and model real-world liquidity fees as well as latency.

The remainder of this paper describe the theory and implementation of the trading system. In section 2, the theoretical foundations from the microstructure literature are explained. Section 3 gives a short introduction to GE. Section 4 presents the trading framework, the data-set, and explains the trading scheme. In section 5, the results of the experiments are analysed. Section 6 concludes the article, summarizes the results and suggests further research.

2. Order flow and liquidity asymmetries

Most modern exchanges operate an electronic double auction limit order book. Most of these systems store and match orders submitted by trading participants based on a price–time priority rule. The term order flow describes the stream of orders submitted to these exchanges. Most models in the microstructure literature restrict themselves to deal with the sign series of order flow only.† In these models, order flow is treated as a series of $+1$ for buy and -1 for sell orders.

Bouchaud *et al*. (2004) and Lillo and Farmer (2004) have found long memory effects in the order flow on the Paris

*Corresponding author. Email: christian.oesch@unibas.ch

†This allows for the estimation of the models' parameters as the order sizes themselves fluctuate to such an amount that estimation becomes infeasible.

Bourse and on the London Stock Exchange, respectively. The autocorrelation function for the order signs only slowly decays to a non-significant level. If one accepts the (statistical) market efficiency hypothesis, this must have implications for how price series emerge from supply and demand. If this is not the case, the autocorrelation of the order signs will translate to autocorrelation in price returns.

There are several ways to derive a microstructure model which includes correlated order flow while keeping the market statistically efficient. However, the most promising is a model with permanent but history-dependent price impact (see Bouchaud *et al*. 2009). The model distinguishes three classes of agents. The first class consists of directional traders which execute large orders and consume liquidity. Their orders are split up into smaller portions to reduce price impact costs. Price impact always moves the price in an unfavourable direction for the liquidity consumer and so infers costs. Splitting up the large orders minimizes price impact for agents and generates correlated order flow on the market. The second class of market participants represents the liquidity providers which forecast order flow and set their trading offers at both bid and ask. This allows them to earn the bid–ask spread. Noise traders make up the last class. They are modelled by a random buy and sell strategy without any correlated process. For simplicity, again, this model excludes volume fluctuations. The return r_n, defined as the difference of the logarithmic mid-price $m_{n+1} - m_n$, of this market in instant n takes the form

$$r_n = \eta_n + \theta\left(\epsilon_n - \hat{\epsilon}_n\right); \quad \hat{\epsilon}_n = E_n\left[\epsilon_{n+1}|I\right] \tag{1}$$

where η_n is an external shock to the market, ϵ_n is the sign of the incoming order, θ is a constant for the size of the trade impact, and I is the information set available at instant n. Assume that $E_{n-1}\left[r_n|I\right] = 0$, i.e. that the liquidity takers and liquidity providers adjust their strategies such that the mid-price is not predictable. However, short-lived deviations from unpredictability might occur. In this model, there are only two possible outcomes to every trading period: either the liquidity provider's predictor $\hat{\epsilon}$ anticipates the correct sign of the incoming order, or it does not.

From the above model, one can derive $r_n^+ = \theta\left(1 - \hat{\epsilon}_n\right)$ and $r_n^- = \theta\left(1 + \hat{\epsilon}_n\right)$ the ex-post *absolute* value of the return of the nth transaction given that ϵ_n matches $(+)$ or does not match $(-)$ the prediction $\hat{\epsilon}_n$. This shows that a more likely outcome of a trade has a smaller impact than an unexpected outcome. Bouchaud *et al*. (2009) calls this 'asymmetric liquidity'. From this, a model results where each transaction has a permanent price impact depending on past order flow. When a buy (sell) market order is more likely, there is a liquidity imbalance in the order book, such that there is a higher than normal liquidity on the sell (buy) side than on the buy (sell) side. This leads to a smaller market impact for the expected buy (sell) order.

Note, however, that no liquidity distribution at the spread can be derived from the above model. All that model is saying, is that a more likely trade should have a smaller impact. In this study, GE is used to evolve a liquidity asymmetry condition on real market data to create a trading strategy. If the liquidity relationship is violated, the system adds volume to the side of the book which is undersupplied by adding a limit order at the spread.

3. Grammatical evolution

3.1. Overview

GE (Ryan *et al*. 1998), and other forms of evolutionary computation are inspired by the natural selection process. They apply Darwinian principles on a population of candidate solutions to solve or approximate user-defined problems. These methods do not require derivatives of the search space, but employ an iterative search which improves on existing candidate solutions. Many scientific fields embrace these methods and have applied them to various problem domains (Bäck *et al*. 1997, Chen 2013). Three main components are required to implement such a method.

The first element is an abstract representation of a possible solution to the problem which can be manipulated. In GE a genotype–phenotype mapping is applied to a string of integers. The mapping through a Backus–Naur form (BNF) grammar delivers a problem-specific representation, also called phenotype, which can be interpreted by a computer or a human.† The system allows for the representation of a complete computer program in an arbitrary programming language on a one-dimensional string of integers (O'Neill and Ryan 2001). This string, also called the genome, can be manipulated by simple operators.

The second main component is the definition of a single measure of fitness which captures a candidate solution's performance for a given objective. This is realized by the introduction of a fitness function that maps a given solution into a fitness value. The design of a suitable fitness function is the main challenge in implementing a GE application. As it influences the convergence behaviour of the search for solutions, it can hinder or prohibit the discovery of the desired solutions. Fitness functions are highly problem-specific and include a range of vastly diverse implementations.

The third component is a suitable search algorithm which efficiently discovers solutions to the given problem. While different approaches are possible, standard GE employs an evolutionary algorithm for this task. Evolutionary algorithms are population-based optimization algorithms which are inspired by biological evolution. Most evolutionary algorithms follow similar principles. Initially, a set of solutions is created, either randomly or through a sophisticated initialization algorithm which provides structural diversity. In GE, this process fills the genome with the initial integer numbers, typically in the range [0, 255]. The candidate solutions are referred to as individuals and the set of current individuals is called the population. In a second step, the initial population is evaluated and a fitness value is assigned to each individual. In a third step, individuals are selected for reproduction, based on their fitness values. After this, the selection of solutions is manipulated to create a new population through crossover and mutation of the genome. The process of evaluating and generating a new population repeats itself until a predefined goal has been reached. The goal can be stated as a fitness threshold or a

†This distinguishes GE, and similar methods, from other common computational intelligence approaches. For example in artificial neural networks, the solutions are represented in the form of a black-box which makes human interpretation difficult or impossible.

maximum amount of generations. A pseudo-code description of the full GE algorithm can be found in Brownlee (2011).

GE is closely related to genetic programming by Koza (1992). Genetic programming has been widely accepted as a way to automatically generate computer programs (Langdon 2012). However, GE's originality lies in the decoupling of the genetic information from its interpretation. The genotype–phenotype mapping can be shown to represent any optimization problem (Rothlauf 2006) and has several advantages over other approaches. For example, the modularity of the various systems in the algorithm offers great flexibility. Since the genetic information is saved on a string of integers, different search methods, other than an evolutionary approach, can be applied to the genome. These include particle swarm optimization by Eberhart and Kennedy (1995), adapted to GE by O'Neill and Brabazon (2004) or differential evolution by Storn and Price (1997), adapted to GE by O'Neill and Brabazon (2006). Independent of the search method, the grammar ensures closure of the phenotypes, as defined by Koza (1992). Closure, in simple terms, requires each possible candidate solution having a valid return value. Furthermore, the representation of a solution in an arbitrary language allows for the phenotypes to take numerous forms. This can be seen in its diverse applications. It has been implemented in a wide array of domains, such as automatic music composition (de la Puente *et al*. 2002), biochemical network modelling (Moore and Hahn 2003), evolving local search heuristics (Burke *et al*. 2012), foetal heart rate classification (Georgoulas *et al*. 2007), market index trading (O'Neill *et al*. 2001) or playing computer games (Perez *et al*. 2011). Another advantage over other approaches results from the mapping process which allows for redundant genetic information. This can improve exploration and exploitation behaviour of the search algorithm (O'Neill and Ryan 2003).

3.2. Mapping

Since the mapping process of GE is not shared with other evolutionary methods, a closer look will be provided here. To derive the phenotype a BNF grammar is used. Each BNF grammar definition consists of the four tuples $\{N, T, P, S\}$. N contains the non-terminals which are expandable into one or more terminals or non-terminals. Terminals T are defined as the items which can make up the language (e.g. $+, -, *, /, ...$) as well as predefined variables (x, y, ...). The production rules are denoted by P and specify how non-terminals are mapped into terminals. The starting symbol S defines the root node which is a member of the non-terminal set. Starting with S, every non-terminal is replaced with a non-terminal or terminal until the whole expression consists of terminals only. The four tuples represents a complete scheme to derive a phenotype from any possible genome. An example BNF grammar for GE looks as follows:

```
N = {<expr>, <op>, <function>, <var>}
T = {+, -, /, *, (, ), sin, cos, x}
S = {<expr>}.
```

P, the set of production rules, is given by:

```
<expr>::=       <op> | <var> | <function>

<op>::=         <expr>+<expr>
                | <expr>-<expr>
                | <expr>/<expr>
                | <expr>*<expr>

<function>::=   sin(<var>) | cos(<var>)

<var>::=        x | y
```

where | separates alternatives.

The production rules are used to develop a symbolic expression from the string of integers. In the above example grammar, the initial starting symbol <expr> has three options to expand: Either the initial expression <expr> is replaced with <expr><op><expr> (option 0), with <var> (option 1) or with <function> (option 2). To decide which non-terminal to choose, a modulo operation is performed on the first integer on the genome:

$$o = c\%n, \tag{2}$$

where o is the numerical index of the option to choose (0, 1 or 2), c is the value of the current integer, n is the total number of available options and % is the modulo operator which returns the remainder of an integer division. To further expand the phenotype the process is repeated for each non-terminal using the next integer on the string to decide between the options until all non-terminals have been mapped into a terminal.

Given the genome:

$$(5\ 120\ 15\ 93\ 206\ ...)\,,$$

the mapping by the above grammer will look as follows. The first number on the integer string is a 5. The initial expression, non-terminal <expr>, comes with three alternatives, the modulo operation $o = 5\%3$ returns the number $o = 2$. As the number 2 selects the choice for replacement of the starting symbol <expr> by the non-terminal <function>, the next choice is between the functions sin(<var>) and cos(<var>). The modulo operation $120\%2 = 0$, selects the sine function. Finally, the last non-terminal <var> is replaced by $(15\%2 = 1)\ y$. The full phenotype then reads $\sin(y)$.

Figure 1. The training–validation–testing cycle as spread over 5 days.

4. Framework

4.1. Overview

In the following, a trading system is created to exploit liquidity undersupply on the bid side of the market. The restriction to a long strategy, avoids short selling limitations/restrictions. As outlined in section 2, the goal of the algorithm is to detect

disparity of the order flow and liquidity relationship which inhibits the translation of correlation in order flow into autocorrelation in returns. The trading system is trained, validated and tested in a typical online set-up as, e.g. proposed in Contreras *et al*. (2013). First, 20 different populations are trained on a training window of three days. To pick a candidate for the actual trading, the fourth day is used as a validation sample. The post-sample, which represents the actual trading period, consists of the data of the day following the validation sample. After one such cycle has been completed, all the periods are moved forward one day and the experiment is repeated. In all experiments following the initial set-up, five populations are seeded from the best populations of the previous run and retrained on the new training window. Figure 1 shows the three different windows for training validation and trading.

4.2. Data

For the experiments, NASDAQ Historical TotalView-ITCH limit order book data provided by LOBSTER† is used. LOBSTER provides both rebuilt order book data as well as the corresponding delta messages. Each order book event is recorded with nanosecond precision time stamps, the order ID, the price (multiplied by 10'000) as well as the volume and a buy/sell indicator. Order book events can be the submission of a new order, the full or partial cancellation of an old order or the execution of a visible or hidden order. Only the top 10 order book levels are used in the study, to keep file sizes manageable. The stocks of Apple Inc. (AAPL), Amazon.com Inc. (AMZN), Google Inc. (GOOG), International Business Machines Corp. (IBM) and Priceline Group Inc. (PCLN) are used for the experiments. These stocks have been chosen because they represent some of the stocks with the highest market capitalization on the NASDAQ. The first data sample lasts from 1 February to 28 February 2014. This includes 19 trading days, resulting in 15 experiment cycles (training, validation and testing). The second data sample lasts from 1 July to 31 July 2014 which includes 22 trading days, resulting in 18 cycles. For all samples only a time window from 11 am to 3 pm is used to exclude any opening and closing effects of the exchange.

Table 1 provides some descriptive statistics for the data sets. Listed are the total number of observations, i.e. order book events, the monthly return, the monthly volatility of the mid-price, calculated from daily returns, the order sign correlation (o. s. corr.) as well as the NASDAQ traded volume in shares for both February and July 2014. Notable is the large amount of order book events and high volume for AAPL in July. While there is no certainty, the stock split on 9 June for 7–1, the quarterly report and other press releases in that month are possible causes for the increased trading activity. The period following July 2014 was also exceptional as the stock reached a new split-adjusted all-time high in price (2 September), as well as most trades and the highest volume (3 September). Please note that the reported trade volume is from TotalView-ITCH data alone which only lists trades that have been executed on NASDAQ. For February 2014, the NASDAQ volume made up 26.2% of Tape C securities, for July 25.8%.‡

4.3. Trading

As mentioned before, the trading system relies on the volume imbalance at the bid and ask prices. Since the bid–ask volume ratio is not defined in the theory; GE is used to create a condition. If this condition is violated, the market is expected to counteract to prevent autocorrelation in future returns. The algorithm is set up to predict this reaction and submitting an order on the buy side of the market. The algorithm is limited to a bid order (OB) strategy for simplicity and to avoid any assumptions on short sell restrictions.

At every order book event, the algorithm checks if the moving average of the order flow with time-window δ, $\mathrm{MA}(o, \delta)$ is negative. If this is the case and a bid–ask volume ratio is below s (where s is the GE's signal), a buy limit order of volume 1 at the best bid is submitted to the exchange. The condition that triggers the submission of a strategy opening limit order, looks as follows:

$$\mathrm{MA}(o, \delta) < 0 \quad \wedge \quad \frac{v_{\mathrm{bid}}}{v_{\mathrm{ask}}} < s. \tag{3}$$

Subsequently, the conditions for a cancellation or execution of the limit order are checked. The cancellation condition is the logical inverse of equation (3): If the moving average of the order flow has become positive or the volume condition is no longer satisfied, the buy limit order is cancelled. To check for execution, volume changes at the limit order price are tracked and, based on the price–time priority rule of the NASDAQ execution scheme, the order execution time is inferred.§ As soon as a trade opening buy order has been executed, a take-profit (TP) sell limit order of volume 1 is submitted to the order book. The TP price is set to the second level of the ask side. The stop-loss level is set to the second level of the bid side. The second levels are used as preliminary experiments have shown them to be advantageous to the performance of the strategy. Note that the stop-loss cannot be placed into the order book as a limit order, as it would be executed immediately. There are now three ways how the open position can be closed. The first and only profitable way is if the TP order is executed. The second way is if the bid price falls to the stop-loss level, where the position is closed by a market order. This incurs liquidity removal costs of \$0.003 per share as specified by the NASDAQ fee structure.¶ The third way results when the predefined maximum prediction period of 500 order book events has been reached. Again the position will be closed with a market order and an additional liquidity removal fee of \$0.003 applies. If there is an open order or an open position, no new trade can be opened. Liquidity provision through the take-profit

†LOBSTER: Limit Order Book System – The Efficient Reconstructor at Humboldt Universität zu Berlin, Germany. http://LOBSTER.wiwi.hu-berlin.de.

‡See https://www.batstrading.com/market_summary/ (27.08.2015).

§When a new agent limit order is submitted, the current volume queue for the corresponding price level is saved to memory. This volume queue represents the orders with higher execution priority. In the following, the volume changes (cancellations, deletions and executions) within the queue are tracked. The agent's order is only executed when the orders in front of it are depleted.

¶http://nasdaqomxbx.cchwallstreet.com/NASDAQOMXBXTools/bookmark.asp?id=sx-policymanual-bse_6000&manual=/NASDAQOMXBX/main/bx-eq-rules/ (27.08.2015).

Table 1. Descriptive statistics for the employed NASDAQ stocks. The return is for the whole month and the monthly volatility is calculated from daily returns.

		AAPL	AMZN	GOOG	IBM	PCLN
February 2014	Observations	6,161,806	3,398,600	3,242,576	3,426,468	2,041,289
	Return	4.7%	0.87%	7.25%	5.20%	17.92%
	Mon. volatility	8.82%	12.96%	7.12%	7.32%	11.32%
	o. s. corr.	0.26	0.26	0.20	0.21	0.35
	Vol. (Mio.)	40.7	21.0	10.5	13.7	4.5
July 2014	Observations	21,703,673	3,853,297	3,644,058	3,867,043	2,156,370
	Return	2.22%	−3.9%	−1.26%	5.49%	2.04%
	Mon. volatility	8.36%	6.82%	9.24%	1.78%	19.80%
	o. s. corr.	0.32	0.24	0.45	0.26	0.39
	Vol. (Mio.)	182.3	25.0	8.9	15.5	3.9

and the opening order qualify for NASDAQ rebates of $0.001 per share. However, the fees and rebates do not significantly influence the results. Latency is modelled in a simplified form, such that every communication between the agent and the order book takes up 5 ms.

Figure 2 shows two examples of a successful as well as an unsuccessful trade. The first five steps are identical. At point a, the algorithm detects a signal from GE. Of course the information on which the agent builds the signal is already delayed, as the information from the order book reaches the agent with 5ms of latency. Once the signal is detected, the agent submits a bid limit order at the best bid, which reaches the market at point b. In point c the limit order is executed and the agent is notified at point d which leads to the submission of the TP order and the determination of the stop-loss level. The TP limit order reaches the market at point e. In the case of the successful trade, the bid price rises to the TP level which triggers the limit order at point f, which the agent gets informed about in point g. For an unsuccessful trade, the bid price falls to the stop-loss level at point f which the agent will recognize at point g. The closing market order will then reach the market at point h. Of course the latency can always lead to small changes in the actual execution of the market operations. For example, the actual stop-loss market order might be more expensive if the bid price has moved farther down than the original stop-loss level.

4.4. Signal generation

To generate the opening signal s in (3), GE is used. The grammar is very general and uses a wide range of variables extracted from the order book files. All variables are calculated in a preprocessing run to optimize the performance of the algorithm. The production rules for the grammar are given by:

```
<expr>::=    <op>
           | <function>
           | <var>
           | <constant>

<op>::=      <expr>+<expr>
           | <expr>-<expr>
           | <expr>/<expr>
           | <expr>*<expr>)
           | <expr>^<expr>

<function>::= 1/(<expr>) | sqrt(<expr>)
           | sigmoid(<expr>)
           | exp(<expr>)
           | log(<expr>)
           | max(<expr>,<expr>)
           | min(<expr>,<expr>)

<var>::=     A | B | C | D | E | F | G
           | H | I | J

<constant>::= <int>.<int>

<int>::=     <digit><int> | <digit>

<digit>::=   0 | 1 | 2 | 3 | 4 | 5 | 6
           | 7 | 8 | 9
```

Table 2. Inputs of the signal generating grammar.

Var	Description
A	Moving average of last change in mid-price
B	Moving average of spread
C	Trade initiation (buy or sell)
D	Moving average of trade initiation
E	Moving average of volume ratio at bid/ask
F	Volume weighted price − mid-price
G	Moving average of volume weighted price − mid-price
H	Order sign
I	Moving average of order sign
J	$F - G$

All mathematical operators are fail-safe, such that they always return a valid numerical value. Divisions by zero, for example, will always return zero. The variables are extracted from the 10 level order book at each order book observation and serve as input variables to the signal. Table 2 gives an overview to the variables in the grammar. A is the moving average of the last change in the mid-price. As stated in the data description, all prices and variables derived from prices are multiplied by 10 000 for numerical precision reasons. The moving averages are simple averages of the variables over the

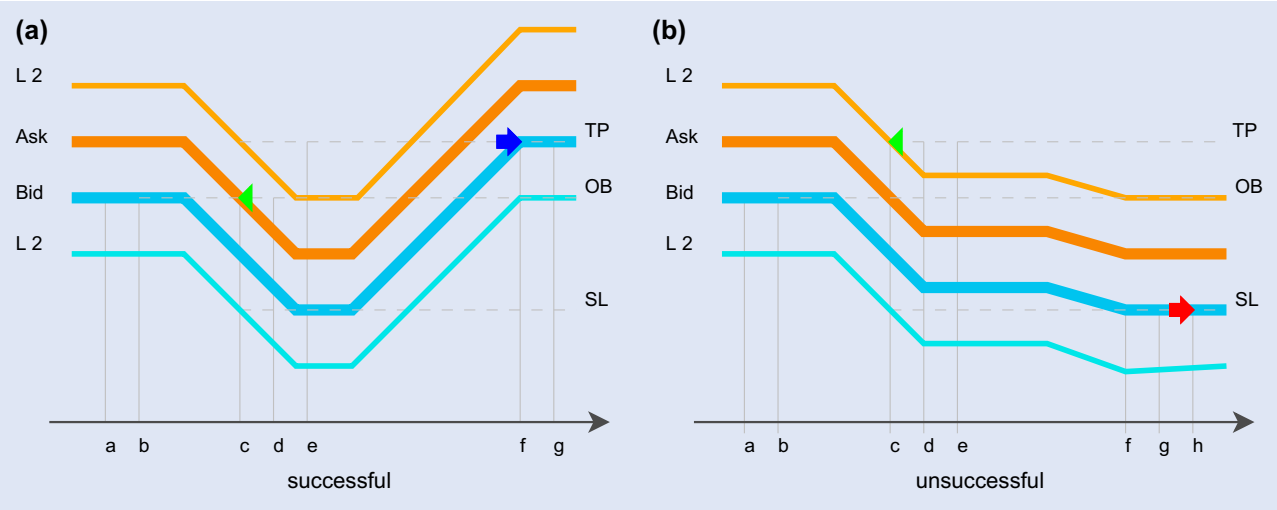

Figure 2. Illustration of two possible round trip trades. Shown are the bid and ask prices, the second levels on each side of the book (L2), the price of the opening OB, the TP order as well as the stop-loss price (SL). The triangle marks the execution of the opening order and the arrows indicate the execution of the TP (blue) and the SL (red) orders. Please note that the illustration shows cases where the order queue is fully depleted. This is not always the case.

last 20 s. B is the moving average of the difference between the bid and ask price, the spread. C indicates the trade initiation. If the current observation includes a buy initiated trade, C will be 1, if it is a sell-initiated trade C is 1, and 0 if no trade occurs. D is just the moving average of C. E is the moving average of the volume at the best bid divided by the volume at the best ask and reflects the volume imbalance at the spread. The variable F is defined as the volume weighted price of 10 order book levels minus the current mid-price. The moving average of F is G. The order sign of the submitted order is denoted by H and is $+1$ for all bid and -1 for all sell orders. The next variable I is the moving average of H. J is defined as $J = F - G$.

Table 3. Evolutionary settings.

Parameter	Setting
Generations	100
Population size	500
Crossover probability	0.8
Mutation probability	0.065
Neutral mutation	on
Tournament size	8
Replacement rate	0.6
Elitist operator	1
Initial maximum depth	12
Maximum depth	16
Genome length	1024

4.5. Fitness function

To assign a fitness value to the candidate solutions the Omega-ratio by Keating and Shadwick (2002) is used. In contrast to many other performance measures, the Omega-ratio integrates the information from all higher moments of the return distribution. It measures the ratio of the probability weighted excess returns to the probability weighted losses and is defined as follows:

$$\Omega(\bar{r}) := \frac{\int_{\bar{r}}^{\infty}[1 - F(x)]\,\mathrm{d}x}{\int_{-\infty}^{\bar{r}}[F(x)]\,\mathrm{d}x} \tag{4}$$

where $\bar{r}$ is a threshold return, separating the gains from the losses, and $F(x)$ is the cumulative distribution function of the returns. The Omega function is mathematically equivalent to the returns distribution as there is no loss of information. There is no need to assume a utility function, but only the decision rule that more is preferable to less. As the strategy does not hold any overnight positions, the threshold is set to $\bar{r} = 0$. Additionally, to limit over-fitting, individuals producing less than 200 trades (N) in the training window are set to invalid. In practical terms, this means that the fitness function is defined as follows:

$$\text{fitness} = \begin{cases} \Gamma & \text{if } N < 200 \\ \frac{\sum_i \max(\rho_i, 0)}{\sum_i \max(-\rho_i, 0)} & \text{if } N \geq 200 \end{cases} \tag{5}$$

where ρ_i is the return of trade i and Γ is a large negative fitness value.

4.6. Evolutionary settings

Table 3 lists the evolutionary settings for GE. For terminology, a deeper look into the parameters and calibration, the reader is referred to Poli *et al.* (2008). Initially 500 individuals are initialized by Ramped Half-and-Half, a standard initialization method, with a maximum depth of 12. The individuals are evaluated and the algorithm creates tournaments of sizes 8 to determine which survive and which are to be replaced by new solutions, until 40% of the population are determined to be parents. Children are produced through crossover and point mutation. To increase genetic diversity, the neutral mutation operator, as proposed in Oesch and Maringer (2015), is

Table 4. Strategy results. The returns are for the trading days and the monthly volatility is calculated from daily returns.

		AAPL	AMZN	GOOG	IBM	PCLN
February 2014	Return	6.23%	0.87%	3.10%	5.19%	1.13%
	Mon. volatility	1.59%	0.75%	0.91%	0.58%	1.58%
	No. trades	1166	690	863	1199	855
	Avg. holding time	9.72	7.94	9.94	9.45	6.64
	Omega-ratio	4.86	1.51	2.39	2.45	1.56
July 2014	Return	*	9.24%	3.38%	5.65%	0.02%
	Mon. volatility	*	2.15%	0.19%	1.47%	0.71%
	No. trades	0	1204	986	1150	201
	Avg. holding time	*	8.99	9.85	8.76	7.13
	Omega-ratio	*	4.00	1.89	2.19	1.59

Table 5. Most robust strategy February 2014. The returns are for the trading days and the monthly volatility is calculated from daily returns.

		AAPL	AMZN	GOOG	IBM	PCLN
February 2014	Return	3.95%	6.61%	5.50%	5.64%	3.30%
	Mon. volatility	0.66%	1.31%	0.89%	0.78%	1.23%
	No. trades	508	415	495	1009	203
	Avg. holding time	20.1	18.9	26.8	22.1	17.9
	Omega-ratio	7.51	6.6	4.59	1.99	4.27

switched on. Since wrapping events become destructive with neutral mutation, the genome length has been set to 1024. 100 generations are performed to create the final populations.

Once all 20 final populations have been created, the algorithm uses the validation sample to select the best population. If both the training and validation window exhibit an Omega-ratio larger than 1, the best strategy is then tested on the trading sample. If the Omega-ratio of either the training or validation window is smaller than 1, nothing is traded and the window is rolled forward for the next run. In the following section, the results from the experiments are presented.

5. Results

5.1. Overview

Table 4 shows the results for the training windows. The statistics include the monthly return and volatility, the number of trades, the time of exposure in seconds, as well as the Omega-ratio. The returns represent the results without reinvestment, as it is assumed that only one share is traded. The highest Omega-ratio is achieved for AAPL in February, where the strategy also exhibits a high monthly return of 6.32% with a very low volatility of 0.32%. The second best performance is achieved for the IBM, followed by the GOOG strategy. While the returns do not seem impressive for the PCLN and AMZN stocks, the Omega-ratio is still considerably larger than 1 and the volatility is fairly small. During both months the number of trades varies between 201 for PCLN in July and 1204 for AMZN in July which results in 11–67 trades per day on average. The duration of exposure to market risk is very short, with averages of 6.64–9.94 s per trade. On the whole month, the actual exposure to market risk is between 28 and 188 min.

While the returns themselves are not always larger than what can be achieved through a buy-and-hold strategy (see returns for the stocks in table 1), the risk is much lower. This would allow for highly leveraged trading. In appendix 1 the daily returns of the strategy and the market are listed.

5.2. Low performers

The picture changes a little bit in July 2014. First of all, the algorithm does not find any training–validation run for July for the AAPL experiment such that the Omega-ratio is larger than 1 for both windows. This stops the system from doing any trades in July for the AAPL stock. We suspect the main reason for this to be the stock split on June 9th. The ratio of tick-size to stock price changes by a large factor, such that the discreetness of the price might have a considerable influence on the behaviour of the strategy. If the TP and stop-loss levels would be defined by a different measure, this could probably be averted in future experiments. In July the strategy works a lot better for the AMZN stock, and the performance for the rest of the stocks seems to be persistent.

Further explanation for the lower performance on the AMZN in February and PCLN February and July might be the mean volatility of the bid and ask prices which is considerably higher than for the other stocks. This might also explain the shorter holding time for PCLN as the TP and stop-loss levels might be triggered much faster. While this could be only part of the explanation, it indicates that different stocks need a different trading strategy set-up. Instead of placing the TP and stop-loss levels in the second levels of the order book, moving the levels away from the spread might help strategies on stocks with a high mean bid/ask price volatility.

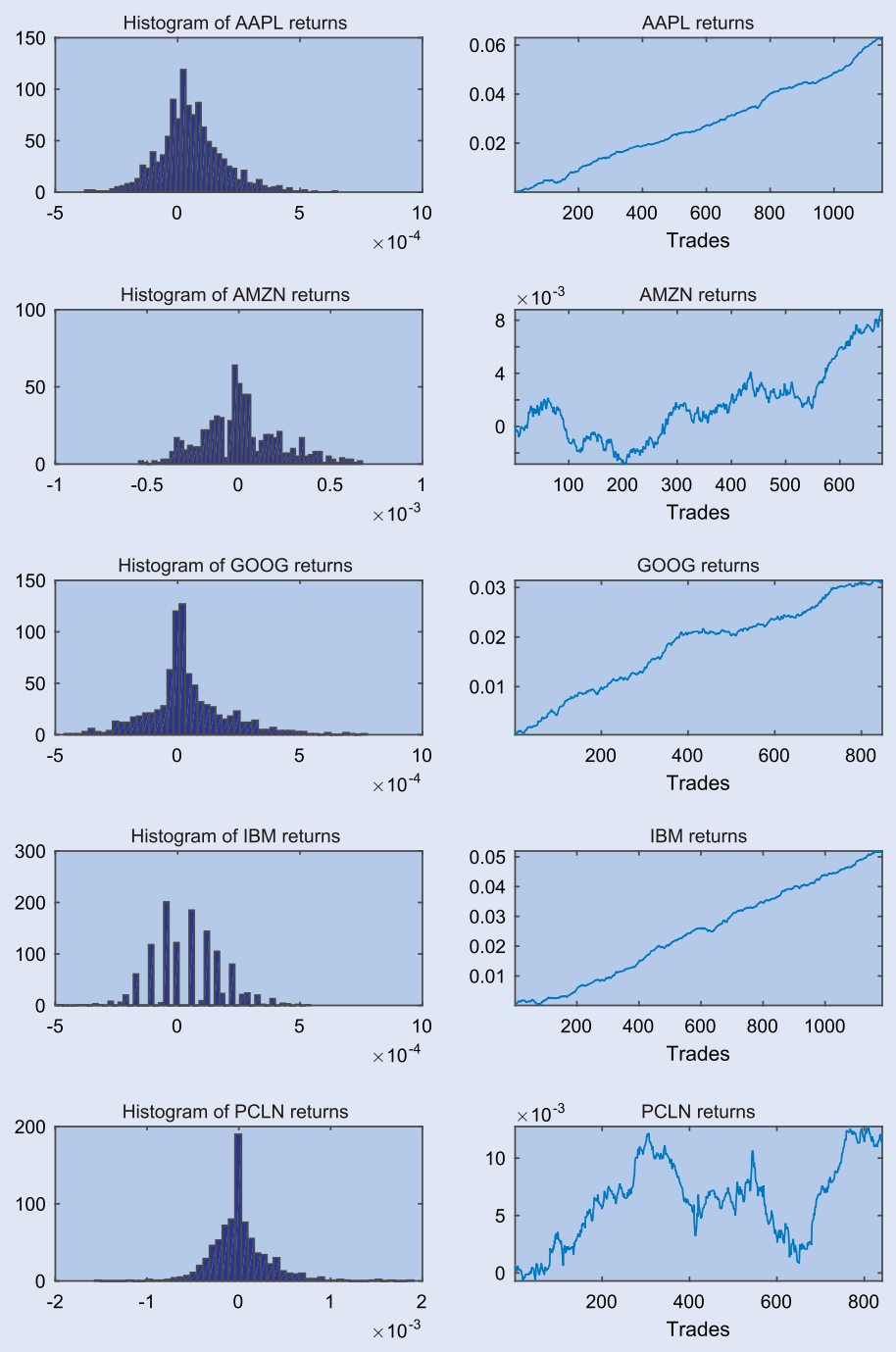

Figure 3. Strategy returns for all stocks in February 2014. The returns are aggregated chronologically from the consecutive testing windows.

Figure 3 shows the histogram of strategies' returns as well as the cumulative sum of the returns. The gaps in the histogram for the IBM returns are an artifact from the tick-sizes.

5.3. *Robustness & phenotypes*

Some of the strategies which have been trained on one of the three-day windows are actually profitable for the whole month. Table 5 shows the statistics of each of the best strategies over the whole month of February. All these strategies have longer exposure times, less trades and most have even a higher Omega-ratio, than the results from the retrained populations. We take this as a strong signal that the algorithm actually has discovered an underlying inefficiency that is consistent. However, one has to be careful with the interpretation, as the strategies have been selected to have the highest performance on the month and as such the performance are not true out-of-sample results.

To get a better understanding of the profitable strategies, one can take a look at the most robust strategies from February 2014.

$$s_{\text{AAPL}} := 1349 \times \log(A) \times (C + \text{sigmoid}(0.597 \times E^{-5}))$$

According to rule (3), a submission of an opening order requires that the ask–bid volume ratio is below the signal s. Hence, all cases where the signal s_{AAPL} is zero or negative, will definitely not lead to a trade. If A, the moving average of the last change in the mid-price, is negative, the log returns zero (as its fail-safe definition requires) and the agent will definitely not trade. For the considered data period, $\log(A)$ never exceeds 0.18. If the last order book observation included a sell trade (i.e. the trade initiation variable is $C = -1$), no trade will occur, as the sigmoid function can only take on values between 0 and 1. Most likely are trades when $A > 1$ and $C > 0$, or when $0 < A < 1$ and $C < 0$.

The most robust strategy for the AMZN stock is as follows:

$$s_{\text{AMZN}} := 74 \times D \times I \times A^{\exp\left(\sqrt{B} - \frac{A}{G}\right)}.$$

The interpretation is only trivial for D, the moving average of the initiation, and I, the moving average of the order sign process. Both values fluctuate between -1 and 1. If only one of them is negative while the rest of the expression evaluates positive, a trade is less likely. If both are positive or negative, a trade is more likely.

The most robust strategy for the GOOG stock is:

$$s_{\text{GOOG}} := \frac{A - \log(C - \log(A))^A}{C}$$

Again, interpretation is not straightforward. However, if $A < -1$ and $C < 0$ a trade is more likely.

The most robust IBM strategy in February 2014:

$$s_{\text{IBM}} := I \times C \times \left(1 + \log\left(\left(\exp(J^{9.338}\right) + 3666245 - E\right)^A\right) \times (A + 8.291)$$

The interpretation is possible if the $\log(\cdot)$ is mostly positive, which looks reasonable due to the typical levels of E and J. Depending on the signs of I, C or A trades become more or less likely.

The most robust PCLN strategy is expressed with a simple formula:

$$s_{\text{PCLN}} := 6.7 \times \left(\sqrt{B} - 3\right).$$

When the moving average of the spread, B, exceeds 9, the signal becomes positive, and trades are very likely. Remember that prices in the data set are multiplied by 10'000 for numerical reasons; this scaling also affects the spreads.

It is interesting to see that the strategies are so diverse. However, this might also be an indication, that the algorithm exploits different mechanisms than the original inspirational asymmetric liquidity imbalance from the market microstructure literature. Preliminary results have shown that, for example, the most robust AAPL strategy also works for IBM and GOOG.

6. Conclusion and future work

In this study, an online trading system is tested. It is motivated by a theoretical liquidity asymmetry and applies GE to discover trading strategies on high-frequency order book data. Limit orders are employed and volumes are tracked to check for execution of the strategies' orders. The fitness is measured by the Omega-ratio. It is shown that GE is capable of finding relationships in very large data-sets and finds profitable strategies with low risk. Furthermore some of the evolved strategies are very robust and are stable for a whole month while being trained on only two days of data.

While the results indicate inefficiencies in the supply of liquidity at the bid–ask spread, further analysis is needed. It is questionable if the strategy would work in the real world without any adoptions. The US market place has a very fractured nature. NASDAQ stocks are not only traded on the NASDAQ but also on other exchanges and on dark markets. This can induce dynamics that cannot be accounted for by TotalView-ITCH data alone. For example, due to the national offer rule, another exchange may offer a better price than what is currently offered at the NASDAQ and so the strategy's order would not be executed. While a perfect backtest is nearly impossible, the addition of other market feeds, such as BATS or ARCA, would allow for much more precise evaluation of the strategy.

There are still many ways to extend the trading system. While this study is restricted to strategies initiating a trade with a buy orders, a combination of buy and sell signals might be worth testing. Moreover, further analysis taking into account more stocks over longer periods as well as the stochastic modelling of network latency could be carried out. In future studies a direct link of volumes, order correlation and price impact will be explored. While the trading system has been set up to create profitable strategies, the structure makes it harder to interpret the phenotypes. It is interesting that many well-performing strategies enter trades only when there was an order which opposes the current moving average of the order sign process. This might be a starting point for further theoretical and empirical investigations.

Disclosure statement

No potential conflict of interest was reported by the authors.

References

Bäck, T., Fogel, D.B. and Michalewicz, Z., *Handbook of Evolutionary Computation*, 1997 (IOP Publishing Ltd.: Bristol, UK).

Bouchaud, J.P., Farmer, J.D. and Lillo, F., How markets slowly digest changes in supply and demand. In *Handbook of Financial Markets: Dynamics and Evolution*, edited by H. Thorsten and K.R. Schenk-Hoppé, Vol. 1, pp. 57–156, 2009 (Elsevier: Amsterdam).

Bouchaud, J.P., Gefen, Y., Potters, M. and Wyart, M., Fluctuations and response in financial markets: The subtle nature of 'random' price changes. *Quant. Finance*, 2004, **4**, 176–190.

Brabazon, A. and O'Neill, M., Evolving technical trading rules for spot foreign-exchange markets using grammatical evolution. *Comput. Manage. Sci.*, 2004, **1**, 311–327.

Brownlee, J., Clever algorithms: Nature-inspired programming recipes, 2011. Available online at: https://www.Lulu.com.

Burke, E.K., Hyde, M.R. and Kendall, G., Grammatical evolution of local search heuristics. *IEEE Trans. Evol. Comput.*, 2012, **16**, 406–417.

Chen, S.H., Evolutionary computation in economics and finance. *Physica*, 2013, **100**, 3–25.

Contreras, I., Hidalgo, J. and Núñez Letamendia, L., Combining technical analysis and grammatical evolution in a trading system. In *Applications of Evolutionary Computation*. Vol. 7835, Lecture Notes in Computer Science, edited by A. Esparcia-Alczar, pp. 244–253, 2013 (Springer Berlin Heidelberg: Berlin).

de la Puente, A.O., Alfonso, R.S. and Moreno, M.A., Automatic composition of music by means of grammatical evolution. In *Proceedings of the ACM SIGAPL APL Quote Quad*. Vol. 32, pp. 148–155, 2002, New York City, USA.

Dempsey, I., O'Neill,, M. and Brabazon, A., Adaptive trading with grammatical evolution. *Proceedings of the IEEE Congress on Evolutionary Computation. CEC 2006*, 2006, Vancouver, BC, Canada.

Eberhart, R.C. and Kennedy, J., A new optimizer using particle swarm theory. In *Proceedings of the Sixth International Symposium on Micro Machine and Human Science*. Vol. 1, pp. 39–43, 1995, New York, USA.

Georgoulas, G., Gavrilis, D., Tsoulos, I.G., Stylios, C., Bernardes, J. and Groumpos, P.P., Novel approach for fetal heart rate classification introducing grammatical evolution. *Biomed. Signal Process. Control*, 2007, **2**, 69–79.

Keating, C. and Shadwick, W.F., A universal performance measure. *J. Perform. Meas.*, 2002, **6**, 59–84.

Koza, J.R., *Genetic Programming: On the programming of computers by means of natural selection*, 1992 (A Bradford Book, The MIT Press: Cambridge, MA).

Langdon, W.B., *Genetic programming and data structures: Genetic programming + data structures = automatic programming!*. Vol. 1, 2012 (Springer Science & Business Media: Berlin).

Lillo, F. and Farmer, J.D., The long memory of the efficient market. *Stud. Nonlinear Dyn. Econometrics*, 2004, **8**, 1–33.

Moore, J.H. and Hahn, L.W., Petri net modeling of high-order genetic systems using grammatical evolution. *BioSystems*, 2003, **72**, 177–186.

Oesch, C. and Maringer, D., A neutral mutation operator in grammatical evolution. In *Intelligent Systems' 2014*, edited by P. Angelov, K.T. Atanassov, L. Doukovska, M. Hadjiski, V. Jotsov, J. Kacprzyk, N. Kasabov, S. Sotirov, E. Szmidt and S. Zadrozny, pp. 439–449, 2015 (Springer: Berlin).

O'Neill, M. and Brabazon, A., Grammatical swarm. *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO 2004*, pp. 163–174, 2004, Seattle, USA.

O'Neill, M. and Brabazon, A., Grammatical differential evolution. *Proceedings of the International Conference on Artificial Intelligence, ICAI 2006*, pp. 231–236, 2006, Las Vegas, USA.

O'Neill, M., Brabazon, A., Ryan, C. and Collins, J.J. Evolving market index trading rules using grammatical evolution. In *Applications of Evolutionary Computing*. Vol. 2037, Lecture Notes in Computer Science, edited by E. Boers, pp. 343–352, 2001 (Springer Berlin/Heidelberg: Berlin).

O'Neill, M. and Ryan, C., Grammatical evolution. *IEEE Trans. Evol. Comput.*, 2001, **5**, 349–358.

O'Neill, M. and Ryan, C., *Grammatical Evolution*. Vol. 4, Genetic Programming Series, pp. 33–47, 2003 (Springer: Berlin).

Perez, D., Nicolau, M., O'Neill, M. and Brabazon, A. Evolving behaviour trees for the Mario AI competition using grammatical evolution. In *Applications of Evolutionary Computation*, edited by C. Di Chio, A. Brabazon, G.A. Di Caro, R. Drechsler, M. Farooq, J. Grahl, G. Greenfield, C. Prins, J. Romero, G. Squillero, E. Tarantino, A.G.B. Tettamanzi, N. Urquhart and A. Sima Uyar, pp. 123–132, 2011 (Springer).

Poli, R., Langdon, W.B., McPhee, N.F. and Koza, J.R. A field guide to genetic programming, 2008. Available online at: https://www.Lulu.com.

Rothlauf, F., *Representations for Genetic and Evolutionary Algorithms*, 2006 (Springer-Verlag, Berlin Heidelberg: Berlin).

Ryan, C., Collins, J.J. and O'Neill, M., Grammatical evolution: Evolving programs for an arbitrary language. In *Genetic Programming*, pp. 83–96, 1998 (Springer: Berlin).

Storn, R. and Price, K., Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces. *J. Global Optim.*, 1997, **11**, 341–359.

Appendix 1. Daily returns

Table A1 lists the daily strategy (S.) and market (M.) returns in %. Please note that the total returns in table 4 are calculated under the assumption that the gains and losses are not reinvested, since only one share is traded. Additionally, we have listed the annualized Sharpe-ratio and the Omega-ratio for the daily strategy and the market returns and the correlation between the strategy and market returns. The discrepancy of the Omega-ratios to the results in table 4 come from the fact that in table 4 the ratio is calculated from trade returns, while it is calculated from daily returns in table A1. We have set the Omega-ratio to ∞ where it is not defined due to no negative returns. The returns for the AAPL strategy in July are empty as the algorithm did not create any trades.

Table A1. Daily strategy (S.) & market (M.) returns in %, the annualized Sharpe-ratio, the Omega-ratio, and the correlation between daily strategy and market returns.

	AAPL		AMZN		GOOG		IBM		PCLN	
	S.	M.	S.	M.	S.	M.	S.	M.	S.	M.
February										
	0.21	1.40	0.11	1.83	0.22	1.51	0.08	1.48	0.04	5.01
	0.22	1.79	−0.03	−0.06	0.53	−0.38	0.31	−0.06	0.24	1.04
	0.79	1.32	−0.27	0.25	0.12	1.47	0.47	1.45	−0.07	0.51
	0.41	−0.01	0.00	−3.47	0.10	−0.29	0.45	0.30	0.49	2.69
	0.47	1.59	0.16	2.28	0.15	1.11	0.58	0.89	0.44	2.36
	0.34	−0.08	0.16	0.04	0.15	0.24	0.30	1.02	−0.14	0.31
	0.26	0.37	0.10	−1.04	0.29	0.67	0.38	−0.27	−0.48	1.69
	0.40	−1.58	0.13	−1.77	0.44	−0.71	0.34	−0.13	0.20	−2.14
	0.05	−1.16	−0.02	0.70	0.10	0.15	0.36	0.72	−0.01	0.73
	0.36	−1.11	−0.16	−0.87	−0.05	−0.03	0.32	−0.80	−0.45	2.54
	0.62	0.44	0.26	1.45	0.18	0.72	0.36	0.36	0.47	−0.15
	0.15	−1.04	0.00	1.86	0.17	0.62	0.25	−0.12	−0.01	3.03
	0.14	−0.90	0.00	0.41	0.59	0.01	0.35	0.45	0.50	0.25
	0.47	1.99	0.15	0.09	0.06	−0.08	0.41	0.66	−0.11	0.06
	1.33	−0.27	0.29	0.55	0.06	−0.29	0.21	−0.05	0.00	−0.66
Ann. Sharpe-ratio	14.77	1.95	4.43	1.20	12.87	7.02	33.56	9.84	2.69	23.76
Omega-ratio	∞	1.45	2.81	1.31	65.73	3.66	∞	5.09	1.89	6.85
correlation		0.16		0.23		−0.23		0.08		−0.12
July										
	*	−2.17	0.56	−2.92	0.02	−1.92	0.10	−0.44	0.00	−2.17
	*	0.91	0.24	1.90	0.02	0.87	0.33	0.64	0.00	0.91
	*	−1.64	−0.37	−0.62	0.02	−0.86	0.60	−0.38	0.11	−1.64
	*	−0.10	1.00	5.57	0.02	1.41	0.11	0.16	−0.05	−0.10
	*	1.12	0.82	2.63	0.02	0.98	−0.03	0.99	0.00	1.12
	*	0.20	1.40	−0.25	0.03	−0.02	0.19	−0.72	0.00	0.20
	*	−0.60	0.34	0.41	0.02	−0.36	0.60	2.05	0.00	−0.60
	*	−1.82	1.13	−0.97	0.03	−1.53	0.64	0.07	0.00	−1.82
	*	0.95	0.72	1.76	0.02	3.72	0.59	0.01	0.00	0.95
	*	0.26	0.27	0.31	0.02	−0.94	0.15	−0.86	0.00	0.26
	*	1.21	0.28	0.30	0.02	0.89	0.20	1.70	−0.19	1.21
	*	0.65	0.23	−0.75	0.02	0.21	0.04	−0.24	0.26	0.65
	*	−0.16	0.15	0.13	0.02	−0.44	0.02	0.83	0.14	−0.16
	*	−0.72	0.96	−9.65	0.02	−0.73	-0.01	−0.43	−0.04	−0.72
	*	0.94	0.53	−1.11	0.02	0.27	0.19	0.71	0.00	0.94
	*	0.59	0.21	−0.13	0.02	−0.84	0.39	−0.62	−0.38	0.59
	*	0.75	0.55	0.78	0.02	0.31	1.05	−0.29	−0.06	0.75
	*	−1.07	0.23	−2.95	0.02	−2.69	0.51	−1.20	0.24	−1.07
Ann. Sharpe-ratio	*	−0.54	16.26	−1.10	72.33	−0.87	14.51	1.84	0.13	−0.54
Omega-ratio	*	0.92	25.78	0.71	∞	0.84	137.02	1.38	1.03	0.92
correlation		*		−0.02		−0.02		−0.08		−0.29