Building Market Timing Strategies Using Trend Representative Testing and Computational Intelligence Metaheuristics

Ismail Mohamed and Fernando E. B, Otero

University of Kent, Chatham Maritime, Kent, UK
{im572, f.e.b.otero}@kent.ac.uk

Abstract. Market timing, one of the core challenges to design successful trading strategies, is concerned with deciding when to buy or sell an asset of interest on a financial market. Market timing strategies can be built by using a collection of components or functions that process market context and return a recommendation on the course of action to take. In this chapter, we revisit the work presented in [20] on the application of Genetic Algorithms (GA) and Particle Swarm Optimization (PSO) to the issue of market timing while using a novel approach for training and testing called Trend Representative Testing. We provide more details on the process of building trend representative datasets, as well as, introduce a new PSO variant with a different approach to pruning. Results show that the new pruning procedure is capable of reducing solution length while not adversely affecting the quality of the solutions in a statistically significant manner.

Keywords: Particle Swarm Optimization, Genetic Algorithms, Market Timing, Technical Analysis

1 Introduction

Trading in financial markets traces its history as far back as the early 13^{th} century. From humble beginnings where traders met to exchange basic commodities, financial markets have since evolved where securities, stocks, bonds, commodities, currencies and other financial instruments are traded electronically at minute fractions of a second. After the market crash of 1987 in the USA, measures were taken by the U.S. Securities and Exchange Commission (SEC) to prevent brokers abstaining from responding to investors' buy and sell orders, in an attempt to mitigate the risk of increasing losses to investors. One of these measures was the Small Order Execution System (SOES). This allowed investors, bar institutions, to use computers to submit orders on the NASDAQ exchange that were automatically matched and executed. For the first time, traders could side step middle men and market makers, and deal with the exchange directly. Traders quickly figured that they could use technology to submit orders and trade on the stock exchange, and that the conduit was widely available. With time, a new kind of trader emerged: the day trader; one that trades through the day and closes out the day with no held securities. By the mid 1990's, the SEC introduced another set of measures that allowed institutions to avail exchanges that electronically matched and executed orders,

known as Electronic Communication Networks (ECN). Although originally intended as alternative trading venues, and underestimated by the established exchanges, electronic exchanges and ECNs grew exponentially to process roughly a quarter of all trades in the US by the early 2000s. By then, the major exchanges such as NASDAQ and NYSE gave in and acknowledged the potential of electronic exchanges, and through a series of mergers and acquisitions, availed their own [21].

The birth of electronic exchanges and ECNs also ushered in a new form of trading: algorithmic trading. Traders wishing to outsmart and beat their competitors to the market started using algorithms to embody trading strategies and submit orders directly to the exchange. The sophistication and speed of these systems grew with time, and started using computational and artificial intelligence techniques by the late 1990s and traded with massive volume at fractions of a second by the mid 2000s.

In order to design a trading system, algorithmic or otherwise, a designer has to tackle four basic issues. The first issue is the *"why"* behind the trading. This concerns the objectives the designer wants to achieve from their trading. These objectives are usually defined in terms of profits, exposure to risk and the length of time the designer wants to achieve their goals over. The objectives can be constrained (e.g. "Double the initial capital, while allowing maximum losses of 25% over the next six months...") or open-ended (e.g. "Maximize profits while minimizing losses for the foreseeable future using the initial capital provided..."). Once the objectives are set, the designer now has to contend with which assets or securities to trade in, or the *"what"* behind trading. This issue is also known as portfolio optimization. Deciding what to trade in is usually based on what best serves the objectives defined by the designer, and again can be either constrained (e.g trading in the securities that belong only to a particular sector of the market) or unconstrained. Having decided on why we are trading, and what securities we are trading, the designer then has to answer *"when"* to actually buy or sell a given security. This issue is known as market timing. The final issue a designer has to contend with is *"how"*. This is also known as execution optimization, and is concerned with how to best form and execute orders for buying or selling a given security once such a decision comes so as to best serve the objectives set out by the designer.

As this chapter is concerned with tackling the issue of market timing, let us further contemplate the implications of tackling such an issue. Market timing can be formally defined as the identification of opportunities to buy or sell a given tradable item in the market so as to best serve the financial goals of the trader [14]. A common approach followed to tackle market timing is to use a collection of components or functions that process current and past market data and produce a recommendation on a course of action: buy, sell or do nothing (also known as hold). This collection of components would form the core of the market timing strategy, and the challenge presented to the designer of an algorithmic trading system is to select the most appropriate components and tune their parameters in such a manner so that it would best serve their goals.

Designers of algorithmic trading systems have increasingly employed computational intelligence techniques soon after the introduction of electronic exchanges. One such employed technique is that of Particle Swarm Optimization (PSO). In comparison to Genetic Algorithms (GA) and Genetic Programming (GP), PSO has not been as popular in the financial domain, and in particular within the market timing space. GA

and GP have been used as the core metaheuristic for market timing strategies ever since the introduction of electronic exchanges in the mid 1990s, while the earliest approach that utilized PSO for market timing was introduced in 2011 [4]. Within the financial domain, PSO has seen limited adoption in literature in comparison to GA, even though PSO has shown to have a performance advantage according to some studies [11,24].

This chapter serves as an extension of the work presented in [20]. We performed an extensive comparison between the applications of PSO and GA to market timing in terms of quality of solutions generated using an expanded set of signal generating components. We also introduced the concept of trend representative testing as a potential remedy to the limitations of step-forward testing – the incumbent method of testing when it comes to training and testing in market timing. In this chapter, we reiterate the work represented in [20], while adding more detail to the process and methodology of trend representative testing as well as a new PSO algorithm with a different approach to pruning.

The remainder of this chapter is structured as follows. In Section 2, we delve deeper into the issue of market timing and review a formalization of market timing first presented in [19] that considers both the selection of components and the tuning of their parameters simultaneously. In Section 3 we look at the application of computational intelligence metaheuristics to tackle market timing and the associated limitations of these approaches. We then take a deeper look at the concept of trend representative testing in Section 4. We provide a more elaborate description of the process of generating a trend representative dataset compared to [20], and discuss how it can potentially address the limitations identified in Section 3. This is followed by a description of the algorithms used to tackle market timing using trend representative testing in Section 5. We also introduce another variant of PSO with a different take on pruning than in [20]. The main impetus behind introducing this new variant is that we wanted to see if it is possible to reduce solution lengths, and the effect that would have on solution quality. In Sections 6 and 7 we discuss how we set up our experiments, present their results and analyze the performance of the different algorithms covered in this chapter. Finally, Section 8 presents the conclusion and suggestions for future research.

2 Timing Buy and Sell Decisions

As mentioned earlier, the issue of market timing is concerned with deciding when to take action with a given security we wish to trade. Our actions can be to either buy this security, sell it or take no action at all. Over time, distinct schools of thought have emerged on how to best time your actions on the market, and these schools of thought can be categorized into two major types: technical analysis and fundamental analysis.

Technical analysis can trace its roots to the 17^{th} and early 18^{th} century with the work of Joseph De La Vega on the Amsterdam stock exchange and Homna Munehisa on the rice markets of Ojima in Osaka. Technical analysis considers a security's current and historical price and volume movements, along with current demand and supply of the security in the market, in an attempt to forecast possible future price movements [14]. Charles Dow helped set the foundations of modern technical analysis with his Dow Theory, introduced in the 1920s. Modern technical analysis is based on three founda-

tions: price discounts all relevant information regarding the security in question, prices have a tendency to move in trends and that history repeats itself. The first foundation is based on the assumption that all factors that can effectively affect the price of a security have exerted their influence by the time a trade takes place. These factors can include the psychological state of persons or entities interested in trading the security, the expectations of these entities and the forces of supply and demand amongst other factors. The first foundation further posits that it is sufficient to only consider current and previous prices of securities of interest as a reflection of all exerted influences on such securities. The second foundation assumes that prices have a tendency to move in trends based on the expectations of entities currently trading in securities of interest. For example, if traders expect that demand will increase for a particular security they would buy that particular security in order to sell at a higher price for profit. As more traders react to this behavior by buying into the security themselves, hoping to generate a profit in the same manner, the price of the security is driven higher and higher in a cascade. The security is then considered to be in an *uptrend*. The trend takes its course, and an opposite cascade of selling occurs, the security is considered to be in an *downtrend*. Being able identify the trend a security is currently in will enable the trader to take the correct course of action within the confines of their strategy. The third foundation assumes that entities trading in securities have a tendency to consistently react in the same fashion when presented with particular market conditions. This phenomenon was proven empirically by observing the histories of a multitude of securities across multiple markets over time.

Methods of technical analysis employs the use of functions known as indicators. Indicators process price history and produce a recommendation of whether to buy or sell a given security. This output recommendation is also known as a signal. Indicators will usually have one or more parameters that affect their behavior, and using different values for these parameters will produce a different signal profile for the same input data. Readers interested in the various methodologies of technical analysis are redirected to the works of Pring [23] and Kaufman [14] for comprehensive descriptions of a multitude of technical analysis techniques as an exhaustive list of all the technical analysis tools available to the contemporary trader is beyond the scope of this chapter.

On the other hand, fundamental analysis is based on the concept that a security has two prices: a fair price and its current market price [22]. Over time, the market price will match the fair price of the security. To arrive at the fair price for a security, fundamental analysis will consider the current financial state of the entity that issued the security. This will include looking at the current and previous financial and accounting records of that entity, considering current and previous management, looking at sales projections and the track record of the entity of meeting those projections, earnings history, micro- and macroeconomic factors surrounding the entity and even the current market sentiment towards the entity among other factors. After considering those factors and arriving at a fair price, decisions to buy and sell will be based on the current discrepancy between the fair price and the market price. The assumption here is that the market price will close the gap and converge with the fair price. If the fair price is less that the current market price, then the decision will be to sell, and if the opposite is true then the decision will be to buy.

Although the more traditional of both approaches, fundamental analysis is not without its caveats. A major issue with fundamental analysis is the rate of release of information for the sources of analysis. Sales and Revenue reports are usually released on a quarterly schedule, while tax filings are only published annually. This could be problematic for strategies working on smaller time horizons, such as trading on a daily or second-by-second basis. In efforts to work around this limitation, fundamental analysis has expanded to include the emergent field of social media sentiment analysis. Sentiment analysis on social media networks is the process of mining these networks for the sentiment of their participants towards all aspects, micro or macro, that could affect a traded security. As contribution on social media occurs at a much higher frequency than the publication of financial documents and reports, traders can act much faster and utilize fluctuations in sentiment to guide buy and sell decisions.

In practice, traders building a strategy would employ techniques from both schools. A common approach is to use fundamental analysis for portfolio composition and technical analysis for market timing. The reasoning behind using techniques from both schools of thought is to reduce the exposure to risk that one or more techniques from either schools might be incorrect or consume data that is of untrustworthy nature [1], thus generating signals that could incur losses. In this chapter, we have chosen to work with technical analysis indicators for practical reasons regarding the availability of data and access to libraries of such indicators.

In order to define a market timing strategy in formal terms, we can consider a market timing strategy to be a set of components. Each component t in the set processes information regarding a security in question and returns a signal: 1 for a buy recommendation, 0 for a hold recommendation and -1 for a sell recommendation. Every component will also have a weight associated with it, as well as a set of unique parameters that influence its behavior. The weight associated with a component t affects the power of the signal generated by this particular component in the overall aggregate signal generated by the candidate solution. If the aggregate signal is positive, then the decision would be to buy. If, on the other hand, the aggregate signal is negative then the decision would be to sell. Otherwise, the recommendation would be to hold and take no action. This formulation can be presented as follows:

$$solution = \{w_1 t_1, ..., w_n t_n\}, \forall t_i : \{t_i^1, ..., t_i^x\} \tag{1}$$

$$signal = \sum_{i=1}^{n} w_i t_i \tag{2}$$

where x denotes the number of parameters for the component at hand, w represents the weight assigned to the component at hand, t represents a single component and n is total number of components within the solution. The weights for the components are all normalized to be between 0 and 1, and have a total sum of 1. Attempting to find the

[1] An example of this would be using a purely fundamental approach while trading Enron before its crash and bankruptcy in late 2001. A post-mortem investigation by the U.S. Securities and Exchange Commision (SEC) showed that the information published in the firm's financial documentation were false, leading to investments by market participants that were built on mislead assumptions.

least possible subset of components that achieve our objectives from potentially endless combinations of components, weights and parameter values, results in a rich landscape of candidate solutions that generate a variety of signals for the same market conditions.

3 Related Work

In the studies by Hu et al. [11] and Soler-Dominguez et al. [24], the authors perform a comprehensive investigation on the use of computational intelligence techniques in finance. Both studies considered a large variety of computational intelligence algorithms that included swarm intelligence (ACO, Artificial Bee Colony Optimization, PSO), evolutionary algorithms (Differential Evolution, GA, GP), fuzzy systems, neural networks and stochastic local search methods (ILS, Tabu Search, GRASP, Simulated Annealing) among others. Although the study by Hu et al. was primarily focused in the role played by computational intelligence in the discovery of trading strategies, the one by Soler-Dominguez et al was more holistic in nature and considered other applications of computational intelligence techniques as long as they where within the realm of finance. The time span covered by both studies reaches as far back as the early 1990's and ends with recent times.

By looking at the algorithms covered by both studies, we can easily arrive at the fact that Genetic Algorithms (GA), and to a slightly lesser extent Genetic Programming (GP), are the most popular algorithms by volume of publications alone. The work by Allen and Karjalanien [2] can be seen as one of the earliest work done using GA. Here, the authors build trading rules based on technical analysis indicators using a GA. The authors then benchmarked their results against a buy-and-hold strategy using out of sample data. Another approach for the use of Genetic Algorithms (GA) is to utilize them to directly optimize the parameters of one or more indicators, regardless of whether these indicators were of the technical analysis variety or the fundamental analysis one. The works of Subramanian et al [25] and de la Fuente et al. [9] can be seen as examples of this approach, with the former tackling market timing as a multiobjective optimization problem. Yet another approach would be to use GA to optimize the parameters of another primary algorithm that is in charge of producing trading signals by way of tuning its parameters in order to improve fitness. Algorithms in charge of producing trading signals in this paradigm included neural networks, self-organizing maps (SOM), fuzzy systems and an assortment of classification algorithms. An extensive cataloging of works done using this synergistic paradigm can be seen in the study by Hu et al. [11]. More recent applications of GA to the issue of market timing can be seen in the works of Kim et al. [15] and Kampouridis and Otero [12].

In comparison to GA and GP, Particle Swarm Optimization (PSO) has not seen such an extensive adoption within the space of market timing. One possible factor that can help us explain this massive difference is that GA were introduced far earlier than PSO, late 1970's versus mid 1990's respectively. The very first application of PSO to market timing was the work done by Briza and Naval, Jr. [4]. Based on the work by Subramanian et al. [25], the authors optimized the weights of a collection of instances of technical analysis indicators. To generate these instances, the authors started with five technical analysis indicators and used industry standard presets for the parameter values

to generate multiple instances from each indicator. The final signal acted upon is then the aggregate signal collected from the individual weighed instances. The approach followed by the authors in this case was a multiobjective one, were they optimized two measures of financial fitness: percentage of return and Sharpe Ratio. This application was then followed by two hybrid approaches, where PSO was used to optimize another signal generating algorithm. In [5], Chakravarty and Dash used PSO to optimize a neural network that was tasked with the prediction of movements in an index price. Liu et al. [17] also used PSO to optimize another signal generating algorithm, in this case a neural network that generated fuzzy rules for market timing, and the authors reported positive results.

In more recent publications, Chen and Kao [6] used PSO as a secondary algorithm to optimize the parameters of a primary system that is capable of forecasting the prices of an index as means of market timing. This primary system is built using a combination of fuzzy time series and support vector machines (SVM). In [16], Ladyzynski and Grzegorzewski also used PSO as a secondary algorithm to optimize the parameters of a primary system. In this case, the primary system was tasked with the identification of price chart patterns for the purposes of market timing and relied on fuzzy logic and classification trees. The authors noted that the use of PSO in this scenario vastly improved the accuracy of their primary system and that the overall hybrid system proved to be promising. Wang et al. [27] used a hybrid approach combining PSO and a reward scheme to optimize two technical indicators in order to improve Sharpe Ratio. The results of this hybrid showed that this approach outperformed other methods such as GARCH. In [3], Bera et al. used PSO to optimize the parameters of a single technical indicator and choose to apply their methodology to the foreign exchange market instead of the stock market. In their results, the authors have noted that their system was profitable under testing. In [26], Sun and Gao have used PSO in a secondary role to optimize the weights on the links of a neural network tasked with prediction of prices of securities on an exchange. The results showed that this hybrid system had an error rate of 30% in predicted prices when compared to the actual ones. In yet another approach where PSO played a secondary role, Karathanasopoulos et al. [13] used PSO to optimize the weights on a radial basis function neural network (RBF-NN) built to predict the prices of the crude oil commodity. Compared to two other classical neural network models, the authors noted that their PSO-enhanced approach outperformed the others in predictive capacity.

In the aforementioned publications regarding the use of PSO in the domain of market timing, we can observe a notable trend: PSO was more frequently used in a secondary role to optimize the performance of a primary signal generating system that used other algorithms besides PSO. There were three exceptions to this observation: [4], [27] and [3]. In these three exceptions, PSO was used as the sole algorithm responsible for the generation of signals – either by optimizing the weights of a set of technical indicators with preset parameters or optimizing the parameters of one or more indicators with preset weights. The only attempt we are aware of that considered the simultaneous optimization of both the selection of indicators and the tuning of their parameters was the one by Mohamed and Otero [19]. Here, the authors used PSO to optimize the parameters of six technical indicators, as well as tuning the weights of

their produced signals and pruning ineffective ones. Their work was tested against four stocks and showed that using PSO was a viable approach albeit with some caveats: the set of indicators used was limited in scope; the number of datasets used for training and testing was small; and no there was no benchmark to compare the performance of the PSO against.

The work proposed in this chapter addresses two key limitations identified in the literature so far: (1) it considers the optimization of both the selection of signal generating components and the values of their parameters in a simultaneous fashion; and (2) avoid the tendency of market timing strategies to overfit to particular price movements when using step-forward testing via the proposition of the use of trend representative testing as is discussed next.

4 Trend Representative Testing: Simulating Various Market Conditions While Training and Testing

The current incumbent method of training and testing used when building market timing strategies in the literature surveyed is a procedure known as step-forward testing [14,11,24]. A straightforward approach, step-forward testing starts by acquiring a stream of price data for a particular tradable asset and then arbitrarily split this stream into two sections: the chronologically earlier section being used for training, while the later is used for testing. A common approach is to ensure that the training section is proportionally twice the size of the testing section in terms of number of data points contained within each section. The main issue with step-forward testing is that while training your algorithm, you are confined to the price movements or trends observed from the data points within the training section. This means that the algorithm is only exposed to the upwards, downwards and sideways trends currently manifest in the training data both in terms of length and intensity. This introduces the likelihood of overfitting to these particular trends, and when faced with different types of trends (those with different lengths and intensities) in real life trading, all the profits that were seen while training and testing quickly evaporate. A simple example would be if an algorithm only sees upward trends during both training and testing, and then is exposed to a downwards trend in real life trading. An example of this can be seen in Figure 1. This shortcoming has been reported in both the studies by Hu et al. [11] and Soler-Dominguez et al. [24]. Furthermore, trying to apply standard tactics to avoid overfitting such as k-fold cross validation are not easy due to the structure of the data.

In order to overcome the limitations associated with step-forward testing, we propose the use of Trend Representative Testing. The ideas behind this novel approach are based on the suggestions of domain experts in [14]. The main philosophy behind Trend Representative Testing is that by exposing an algorithm during training and testing to a variety of upwards, downwards and sideways movements, we reduce the chances of the algorithm overfitting to any single one of those trends and have a better estimation of the algorithm's performance in real life trading. Our objective is then to build a library containing numerous examples of each type of trend, with various intensities and time lengths, and define an approach on the use of this library in training and testing.

Fig. 1: An example of dividing price data for step-forward testing[1]. The data shown here represents daily prices for the Microsoft (MSFT) security on the NASDAQ market between 2016 and early 2019. The red line represents the point of division, with the first two years of data being used for training, and the later year used for testing. The data represents an example of the liability for step forward testing to overfit, as the entirety of data shown here represents an extended upwards trend. Algorithms trained and tested on this data are liable to perform poorly when exposed to a downtrend.

[1] Image courtesy of Yahoo! Finance

The process of building a dataset for Trend Representative Testing is a systematic approach of analyzing raw price streams, identifying usable subsections with known trends and then storing them within a library so as to have a multitude of uptrends, downtrends and sideways movements for use in training and testing. The first step of this process is to acquire a vast amount raw price data, over an extended time frame to improve our chances of capturing the largest variety of trends in terms of direction, intensity and length. For our library, we acquired the raw price data for all securities exchanged on the Nasdaq and NYSE markets traded from 1990 to 2018. Each individual price stream is then scanned for price shocks, and upon detection, the raw price stream is divided into two sub-streams known as cords: one cord representing the data occurring before the price shock and the other representing the data occurring after the price shock. The reason we remove price shocks is that they are outlier events, categorized by a sudden change in price in response to an event. Price shocks are highly unpredictable and disruptive events. Including these sudden and disruptive changes in the training data would imply that our trained market timing strategies are capable of predicting price shocks and correctly responding to them, which is not the case. This is the main reason why we elect to remove price shocks from training and testing data. Besides being catastrophic events, price shocks are rare and training a strategy to use them would be highly impractical. Price shocks can be defined as price actions that are three times the Average True Range (ATR) within a short period of time [14]. Each generated cord is then subsampled using sliding windows of various sizes to produce strands. Each strand is then analyzed to identify the direction of the underlying trend represented (upwards, downwards or sideways) and intensity using the Directional Index technical indicator

Fig. 2: A visual example of the process behind generating a Trend Representative Testing dataset[1]. The data shows here the price data for the Dow Jones index between July 1987 and January 1988. In (A) we can see the raw price data, with the tall black rectangular highlighting the price shock caused by the events of Black Monday on October 19, 1987. This price shock is confirmed in spikes of Average True Range and True Range depicted directly under the top chart. Upon the identification of the price shock, the raw stream of data is divided into two streams we call cords: one before the price shock (B), and the other after the price shock (C). The cords are then subsampled using sliding windows of various sizes to produce what we call strands (D). Strands are then analyzed using the Directional Index technical indicator to identify the underlying trends and stored in library that forms the Trend Representative Testing dataset.

[1] Images courtesy of Yahoo! Finance

[23] and finally added to the library. A visual example of this process can be seen Figure 2.

In order to use this new dataset, we first start by building two sets: a training set and a testing set. A training set is composed of n triplets, where a triplet is a set of three strands: one uptrend, one downtrend and one sideways trend. A testing set is composed of a single triplet. During training, we select a triplet at random from the training set with each iteration, and that triplet is used to assess the performance of all the candidate solutions within that iteration. To measure the performance of a candidate solution, we evaluate the performance of the candidate solution at hand agaisnt each constituent strand within the triplet designated for the current iteration and report the average fitness. With each new iteration, a randomly selected triplet from the training set is designated as the current training triplet and this process is repeated until the training criteria are met or a set of training iterations are exhausted. For testing, the triplet from the testing set is used in a similar fashion to assess the performance of the surviving solutions form the algorithm. The idea behind trend representative testing is that we want discourage niching or specializing in one particular trend type and instead promote discovering market timing strategies that fair well against various market conditions.

5 Market Timing Algorithms

In this section we describe how our market timing formalization was encoded, explain how the algorithms were adapted to use this encoding and tackle the problem of market timing.

5.1 Individual Encoding and Measuring Fitness

Let us begin this section by describing how the formalization presented in Section 2 is encoded into a candidate solution, and how we assess the fitness of these solutions. According to our formalization, a candidate solution is a collection of signal generating components, where each of these components would have a weight and a set of parameters. For example, if we had three signal generating components such as the technical indicators Moving Average Converge Diverge (MACD), the Relative Strength Indicator (RSI) and the Chaikin Oscillator (Chaikin), we could possibly have a candidate solution of:

$$\begin{aligned} &0.3\times MACD(Fast\ Period = 12, Slow\ Period = 27,\\ &\qquad Smooth\ Period = 9)\\ &+0.2\times RSI(Overbought = 70, Oversold = 30,\\ &\qquad Period = 14)\\ &+0.5\times Chaikin(Fast\ Period = 3, Slow\ Period = 10) \end{aligned}$$

where the number preceding the indicator represents its weight and the values in the brackets represent the values of parameters per indicator. A visual example can be seen in Figure 3.

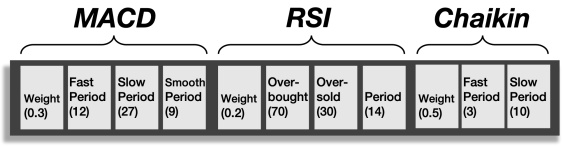

Fig. 3: An example of an encoded candidate solution with three components. Reproduced from [20]

As the algorithms used are all based on a population of individuals, each individual would represent a candidate solution. We chose to encode the individuals as multi-tier associative arrays. The top level binding associates an indicator identifier with a set of of its parameters. The bottom level binding associates a parameter identifier with its value. The parameters per indicator are dependent on its type, but all indicators have an instance of a weight parameter.

Our choice for encoding individuals as multi-tier associative arrays allows us maximum flexibility to use an unlimited number of signal generating components and be agnostic of their parameters in terms of quantity and value types, as would have been the case if we choose to encode an individual as a regular array. Encoding the individual as an array, would have confined us to particular number of components and needed the maintenance of a mapping structure to be used to decode the array's values, both limitations that we wished to avoid at this time. The semantics of how each component is handled and how its weight and parameters are tuned are then left to be implemented in the individual algorithms, as will be discussed shortly. A new candidate solution is generated by instantiating components from the available catalog with random values for the parameters and adding it to the dictionary representing the individual.

As for assessing the fitness of an individual, we chose to maximize the Annualized Rate of Return (AROR) generated by backtesting the individual at hand. Backtesting would simulate trading based on enacting the aggregate signal produced by a solution over a preset time period for a given asset. The data being used during backtesting would depend on the current strand being used from the Trend Representative Testing procedure. As for AROR, this can be defined as:

$$AROR_{simple} = \frac{E_n}{E_0} \times \frac{252}{n} \tag{3}$$

where E_n is final equity or capital, E_0 is initial equity or capital, 252 represents the number of trading days in a typical American calendar and n is the number of days in the testing period.

5.2 Genetic Algorithms

In order to adapt Genetic Algorithms (GA) to tackle market timing using our proposed formalization, we utilized a typical implementation of GA and modified the crossover and mutation operators to accommodate our individual encoding scheme. For crossover, we begin by selecting individuals for the crossover procedure using typical tournament selection, where the tournament size is a user-defined parameter. The selected individuals are then prepared by ordering the genotype by key. A crossover point is then selected

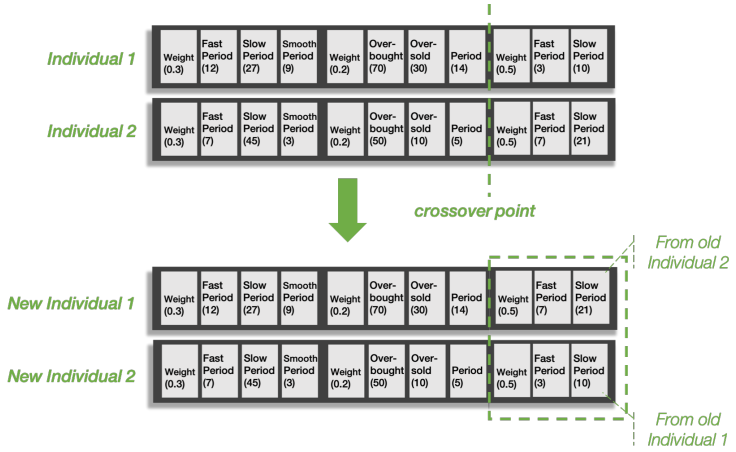

Fig. 4: An example of a crossover operation. Reproduced from [20]

at random with the constraint that it lands between the definition of two components, but not within them. This would ensure that the components in the resulting genotypes are valid, with the correct number of parameters and parameter values are within valid ranges. Using the example mentioned earlier, we can generate a crossover point either between the definition of MACD and RSI, or between RSI and Chaikin. An example of a crossover operation can be seen in Figure 4. In order to simulate mutation, a random component from within the individual's genotype is replaced by a fresh copy of the same type of component with random (but valid) values for its parameters.

The crossover and mutation operators are used to generate new populations, generation after generation. This process is repeated until an allocated budget of generations is exhausted. Elitism is implemented via the use of an archive to keep track of the best performing individuals per generation, with the fittest in the archive being reported as the proposed solution at the end of the algorithm's run.

5.3 Particle Swarm Optimization

We now turn our attention to adapting Particle Swarm Optimization (PSO) to tackle market timing. As PSO is based on a swarm of individuals representing candidate solutions, we will use the same encoding we introduced earlier. In order to be able to work with this encoding within PSO, the standard mechanics of the various PSO operators will have to be modified, and these modifications are discussed next.

The basic PSO model can be seen in Algorithm 1. This model supports both l-best and g-best neighborhood structures, based on the value of the neighborhood size parameter. A neighborhood size equal to the size of the swarm would imply a g-best neighborhood structure, while values less than the swarm size would imply an l-best neighborhood structure. In order to accommodate the proposed encoding scheme, we pushed down the implementation of the addition, subtraction and multiplication operators required for the velocity update equation (lines 8–15) to be at the component level instead of the algorithm level. This allows us to be agnostic to the types of components

Algorithm 1 Basic PSO high-level pseudocode.

```
initialize swarm S
repeat
    for every particle x_i in S do
        if f(x_i) > personal_best(x_i) then
            personal_best(x_i) ← f(x_i)
        end if
        for every component j in particle i do
            bias ← α v_ij(t)
            cognitive ← c_1 r_1 (y_ij(t) − x_ij(t))
            social ← c_2 r_2 (ŷ_ij(t) − x_ij(t))
            v_ij(t+1) ← bias + cognitive + social
            if j ∈ R then
                x_ij(t+1) ← x_ij + v_ij(t+1)
            else if j ∈ [0,1] then
                Pr(x_ij(t+1) → 1) : sigmoid(v_ij(t+1))
            end if
        end for
    end for
until stopping criteria met
return fittest particle
```

used and their parameter values. It also removes the limitation that the parameter values have to either be binary or numeric in nature, as long as there is a suitable override for the addition, subtraction and multiplication operators within the component. This opens up the possibilities for the designer to consider an arbitrary number of signal generating components for their market timing strategies. We also adopted a decreasing inertia schedule, Clerc's constriction [7] and velocity scaling as measures to promote convergence within the swarm and eliminate velocity explosion.[2] For the decreasing inertia schedule, the inertia for every particle is diminished by an amount defined by a function based on the iterations left with every passing iteration. To eliminate velocity explosion, the designer is a given a choice between using Clerc's constriction [7] or velocity scaling. With velocity scaling, the velocity $v_{ij}(t+1)$ is scaled down by a user defined factor before being used to update a particle's state. Using these modifications, PSO can now use the proposed encoding to optimize both the parameters and weights of a set of components in relation to a financial fitness metric.

A common problem faced by search metaheuristics, PSO included, is getting stuck in local optima. Multiple measures have been devised since the introduction of the basic model to remedy that problem with varying degrees of success [8]. Inspired by the work done by Abdelbar with Ant Colony Optimization (ACO) in [1], we introduced a variation of PSO that stochastically updates the velocity of its particles only when it is favorable in terms of fitness. We will refer to this variation of PSO in the remainder of this chapter as PSO^S. The modifications required for PSO^S are reviewed next.

[2] Early experiments indicated the tendency of particles to adopt ever increasing values for velocity if left unchecked, leading to the particles quickly seeking the edges of the search space and moving beyond it.

Every particle x in the swarm S represents a candidate solution. From our earlier discussion, this means that a particle's state is a collection of weighted components, where each component has its own set of parameters. A particle starts out with an instance of all the available signal generating components, each instantiated with random weights and parameter values. In contrast with the basic PSO model, the cognitive and social components of the velocity update equation are modified to be calculated as:

$$cognitive = \begin{cases} y_i(t) - x_i(t) & \text{if rand()} < |\frac{f(y_i(t))}{f(x_i(t))+f(y_i(t))}| \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

$$social = \begin{cases} \hat{y}_i(t) - x_i(t) & \text{if rand()} < |\frac{f(\hat{y}_i(t))}{f(x_i(t))+f(\hat{y}_i(t))}| \\ 0 & \text{otherwise} \end{cases} \tag{5}$$

- x: particle
- i: current particle index
- y: personal best
- $\hat{y}$: neighborhood best
- $f(x)$: the fitness of x
- $rand()$: random number between 0 and 1

According to equations 4 and 5, the cognitive and social components will only stochastically influence velocity update if there is an improvement in fitness, following a hill climbing fashion.

In [19], we introduced a pruning procedure in an attempt to arrive at shorter solutions by actively removing signal generating components whose weight falls below a specific threshold at specific checkpoints through the algorithm's run. Experimentation with this pruning procedure did not prove to be fruitful. This was further confirmed when IRace found the best setting for that pruning procedure is to be turned off. This suggests that the pruning procedure is too destructive. Any components that fall below the pruning threshold is removed from all solutions in the swarm and there is no mechanism to reintroduce the pruned components at a later stage. Therefore, components that do not have a good configuration at the moment, and thus not contributing to the solution in a beneficial manner, get removed without having the opportunity to explore other configurations. Shorter solutions have an advantage of being faster to compute and easier to comprehend by the user, and so are still desirable. The challenge is to find a good balance between the quality and size of a solution. In an another attempt to arrive at the *least* sufficing subset of components that optimizes a financial metric, we introduce a novel approach to pruning that was not present in [20]. In this novel pruning approach, components with weights falling below a threshold will have their weights updated to zero without physically removing them from the solution as was the case in [19]. Components who have weight of zero are effectively excluded from contributing to the aggregate signal produced by the candidate solution and thus can be disregarded. By not permanently removing them from the solutions, we allow their reintroduction in later iterations if they *learn* of a useful configuration through the interaction of the particles in the swarm. The pruning procedure is triggered at frequent points throughout the algorithm's run, and the deadline (the number of iterations that pass before it

Table 1: IRace discovered configurations for each of the algorithms tested.

PSO		**PSO^S**		**PSO^P**		**GA**	
Population	45	Population	59	Population	30	Population	53
Iterations	28	Iterations	261	Iterations	90	Generations	266
Neighbors	26	Neighbors	59	Neighbors	30	Mutation Probability	0.6306
c1	2.4291	c1	3.2761	c1	2.908	Crossover Probability	0.455
c2	3.4185	c2	2.363	c2	3.417	Tournament Size	22
Clamp	Scaling	Clamp	Scaling	Clamp	Scaling		
Scaling Factor	0.8974	Scaling Factor	0.551	Scaling Factor	0.5988		
		Pruning	False	Pruning	True		
				Pruning Threshold	0.1799		
				Pruning Deadline	15		

is triggered) and threshold are all user defined parameters. This pruning procedure is added on top of PSO^S resulting in a new variant we will refer to as PSO^P.

6 Experimental Setup

In order to evaluate the efficacy of the proposed methods in building competent market timing strategies, we tested the proposed GA, PSO, PSO^S and PSO^P algorithms. Since all four algorithms have parameters, testing was preceded with hyper-parameter optimization performed using the iterated racing procedure (IRace) [18]. The IRace procedure was run with a budget of 300 iterations and a survivor limit of one, in order to arrive at a single configuration for each algorithm. The results of the IRace procedure can be seen in Table 1.

Looking at the PSO variants, we can see that the variant with the smallest swarm size is PSO^P, followed by regular PSO then PSO^S which is almost twice the size of PSO^P. When it comes to the total number of fitness evaluations to be performed by the algorithm, regular PSO comes in with the least number of evaluations at 1260, followed by PSO^P and PSO^S with 2700 and 15399 respectively. The regular PSO has adopted an l-best neighborhood structure, while both PSO^S and PSO^P adopted g-best neighborhood structures. All three PSO variants favored scaling over Clerc's constriction, with both PSO variants that use the stochastic state update procedure using a scaling factor of around 0.5 while regular PSO adopted a more aggressive scaling profile with a scaling factor of about 0.9. Both regular PSO and PSO^P leaned slightly more towards depending on the social component during velocity update, while PSO^S showed the exact opposite. In essence, after hyper-parameter optimization using IRace, we ended up with a fast acting l-best PSO, an almost equally fast g-best PSO^P and a relatively slow PSO^S. As for GA, the hyper-parameter optimization procedure resulted in a relatively slow GA (with a population size and generation count close to PSO^S) with a high mutation rate and low crossover rate. This suggests that the solution landscape is on the rugged side, with sharp peaks and deep valleys that require radical moves to traverse.

All algorithms are trained and tested using trend representative testing. The data used contains 30 strands, representing 10 upwards, 10 downwards and 10 sideways trends at various intensities. The 30 strands are divided into triplets, resulting in 10 distinct triplets for use in training and testing. The details of the trend dataset can be

Table 2: Data strands used for training and testing. Reproduced from [20]

Id	Symbol	Begin Date	End Date	Length	Trend
BSX1	BSX	2012-10-10	2013-07-09	185	↑
LUV1	LUV	2008-08-22	2010-05-07	430	↔
KFY1	KFY	2007-05-16	2007-10-12	105	↓
EXC1	EXC	2003-04-14	2003-08-20	90	↑
LUV2	LUV	2004-12-03	2005-05-04	105	↔
KFY2	KFY	2007-03-20	2007-09-21	130	↓
AVNW1	AVNW	2005-07-18	2006-01-12	125	↑
PUK1	PUK	2010-08-12	2012-04-03	415	↔
LUV3	LUV	2008-09-02	2009-01-30	105	↓
KFY3	KFY	2003-03-13	2003-08-04	100	↑
EXC2	EXC	2002-10-03	2003-08-04	210	↔
LUV4	LUV	2003-11-21	2004-04-01	90	↓
EXC3	EXC	2003-05-12	2003-10-15	110	↑
PUK2	PUK	2005-05-12	2006-03-13	210	↔
MGA1	MGA	1996-02-29	1996-07-08	90	↓
ED1	ED	1997-07-02	1997-11-20	100	↑
EXC4	EXC	1999-08-20	2000-03-30	155	↔
PUK3	PUK	2002-03-19	2002-07-25	90	↓
BSX2	BSX	2009-04-22	2009-09-18	105	↑
ED2	ED	2011-12-15	2012-05-16	105	↔
JBLU1	JBLU	2003-05-15	2003-11-10	125	↓
MGA2	MGA	2012-12-28	2013-10-14	200	↑
MGA3	MGA	1995-09-19	1996-12-13	315	↔
ATRO1	ATRO	1997-06-04	1997-11-28	125	↓
AVNW2	AVNW	2003-03-07	2003-08-05	105	↑
EXC5	EXC	2015-03-12	2016-09-02	375	↔
AVNW3	AVNW	2013-06-11	2013-11-20	115	↓
IAG1	IAG	2015-11-09	2016-08-24	200	↑
MGA4	MGA	1995-10-17	1996-04-22	130	↔
IAG2	IAG	2012-01-19	2012-06-04	95	↓

seen in Table 2. The columns in Table 2 describe the symbol of the source stock data, the beginning date, the ending date and the trend of every strand in the dataset. The data has then been split into 10 datasets, where each dataset would contain one triplet reserved for testing and the remaining 9 triplets form the training set. Each step in the training and testing procedure is repeated 10 times to cater for the effects of stochasticity.

When it comes to the signal generating components available for the algorithms, we used 63 technical analysis indicators for our experiments. These indicators contain an assortment of momentum indicators, oscillators, accumulation/distribution indicators, candlestick continuation pattern detectors and candlestick reversal pattern indicators. If any of the indicators had parameters that were concerned with the length of data processed, an upper limit of 45 days was set, so that we were guaranteed a minimum of 5 trading signals within a single trading year [3]. Any other parameters are initialized to

[3] A typical US trading year is compromized of 252 days.

random values as long as comply with the value constraints defined by every indicator, and the best setting for these parameters in terms of performance is discovered by the algorithms as they traverse the solution landscape.

7 Results

Table 3 shows the minimum, median, mean and maximum fitness achieved by the four algorithms per dataset and trend. Regular PSO shows the most wins based on mean performance with 15 wins out of a possible 30, followed by PSO^S with 6 wins, then PSO^P with 5 wins and finally GA with least wins scoring only 4 out of a possible 30. Positive values in Table 3 indicates profits were made on the initial investment, negative values indicate that losses were incurred and a value of zero indicates a break-even situation. By looking at overall averages in Table 4, we can see that all four algorithms performed considerably better with downtrends when compared to the other two trend types. We can also see that GA is the least performing algorithm in all of the trend types, and that a PSO variant has a slight edge over GA in all three cases. The clear difference in fitness between downtrends and the other two types of trends indicates that the algorithms generate market timing strategies that are unbalanced. Nevertheless, performing better in downtrends is positive when compared with buy-and-hold strategies which would fail under such conditions. The issue of unbalanced performance with the various trend types can perhaps be remedied by tackling market timing as a multi-objective optimization problem, where we try to pursue a Pareto front that maximizes performance for all three trend types. By observing the performance of the market strategies generated by all four algorithms under downtrends, uptrends and sideways movements, we have a better approximation of the performance of these strategies under varying market conditions, and therein lies the value of Trend Representative Testing. With step-forward testing, we are confined to the underlying trends represented in the training data. This can easily lead to overfitting, where strategies only perform well when exposed to trends that are similar to those encountered in training and poor otherwise. With Trend Representative Testing, we explicitly avoid this issue by exposing our algorithms to a variety of trends during both training and testing.

Table 5 shows the rankings of the algorithms after performing the non-parametric Friedman test with the Holm's post-hoc test by trend type on the mean results [10]. The first column shows the trend type; the second column shows the algorithm name; the third column shows the average rank, where the lower the rank the better the algorithm's performance; the fourth column shows the p-value of the statistical test when the average rank is compared to the average rank of the algorithm with the best rank (control algorithm); and the fifth shows the Holm's critical value. Statistically significant differences at the 5% level between the average ranks of an algorithm and the control algorithm are determined by the fact that the p-value is lower than the critical value, indicating that the control algorithm is significantly better than the algorithm in that row. The non-parametric Friedman test was chosen as it does not make assumptions that the data is normally distributed, a requirement for equivalent parametric tests.

Table 3: Computational results for each algorithm over the 10 datasets. The min, median, mean and max values are determined by running each algorithm 10 times on each dataset. The best result for each dataset and trend combination is shown in bold.

			GA				**PSO**				**PSO^S**				**PSO^P**			
Dataset	**Trend**	**Test Strand**	**Min**	**Median**	**Mean**	**Max**	**Min**	**Median**	**Mean**	**Max**	**Min**	**Median**	**Mean**	**Max**	**Min**	**Median**	**Mean**	**Max**
0	↑	IAG1	-10.35	-5.99	-6.25	-1.39	-4.41	-4.26	-4.01	-3.04	-9.71	-5.79	-6.19	-3.42	-4.41	-4.00	**-3.79**	-2.08
	↔	MGA4	0.39	0.88	0.91	1.60	0.66	1.78	**1.60**	2.09	0.93	1.19	1.25	1.70	0.91	1.53	1.45	1.93
	↓	IAG2	0.80	1.95	1.71	2.16	2.17	2.17	**2.17**	2.17	0.69	0.93	1.22	2.17	2.17	2.17	**2.17**	2.18
1	↑	BSX1	-0.67	-0.35	-0.36	-0.13	-5.91	-0.31	-0.85	-0.02	-0.85	-0.34	-0.39	-0.11	-0.36	-0.31	**-0.28**	-0.11
	↔	LUV1	-1.33	-0.10	**-0.28**	-0.01	-1.71	-0.11	-0.45	-0.04	-1.10	-0.34	-0.42	-0.08	-4.53	-1.06	-1.58	-0.01
	↓	KFY1	2.01	2.08	**2.22**	2.67	1.86	2.09	2.14	2.66	2.00	2.07	2.07	2.17	1.89	2.05	2.05	2.12
2	↑	EXC1	2.80	2.82	2.83	2.90	2.80	2.80	2.80	2.80	2.80	2.83	**2.85**	2.92	2.80	2.80	2.80	2.80
	↔	LUV2	2.09	2.27	2.31	2.64	2.21	2.43	**2.44**	2.62	2.17	2.31	2.30	2.46	2.20	2.37	2.37	2.51
	↓	KFY2	1.61	1.75	1.77	2.05	1.56	1.98	**2.15**	3.61	1.65	2.00	2.04	2.85	1.76	1.87	1.90	2.20
3	↑	AVNW1	-3.83	-2.01	-1.59	1.29	-1.97	0.70	**0.10**	1.26	-3.58	-1.84	-1.21	1.22	-1.70	-0.48	-0.17	1.28
	↔	PUK1	-2.84	-1.44	-1.32	0.00	-2.37	-0.05	**-0.39**	0.00	-2.45	-1.25	-1.00	0.38	-2.53	-0.03	-0.59	0.09
	↓	LUV3	-0.85	2.91	2.03	4.63	1.08	3.06	**3.17**	4.70	1.58	2.67	3.02	6.12	2.04	2.80	2.89	4.11
4	↑	KFY3	2.08	2.49	2.45	2.67	2.67	2.72	**2.74**	2.80	1.39	2.42	2.40	2.94	2.58	2.67	2.64	2.67
	↔	EXC2	0.16	0.91	0.84	1.55	-0.53	1.31	**1.03**	1.60	0.08	1.13	0.99	1.62	0.06	1.08	0.94	1.68
	↓	LUV4	2.80	2.83	2.84	2.96	2.80	2.80	2.80	2.80	2.77	2.85	**2.85**	2.95	2.79	2.80	2.80	2.80
5	↑	EXC3	1.39	1.64	1.71	2.14	1.96	2.03	**2.12**	2.38	1.52	2.02	2.00	2.39	1.96	1.96	1.98	2.14
	↔	PUK2	-4.10	-0.10	-0.88	0.51	0.06	0.59	**0.58**	1.07	-1.51	-0.23	-0.28	0.76	-2.39	0.12	-0.37	0.54
	↓	MGA1	2.80	2.99	2.98	3.15	2.80	2.80	2.80	2.80	2.80	3.14	**3.15**	3.80	2.80	2.80	2.80	2.80
6	↑	ED1	0.38	1.14	1.10	1.98	1.75	1.75	1.88	2.44	-0.03	1.26	1.21	2.32	0.86	1.85	**1.91**	2.46
	↔	EXC4	1.53	2.27	2.33	3.72	0.95	2.07	2.33	3.47	1.20	1.97	2.35	3.96	0.92	2.65	**2.84**	5.20
	↓	PUK3	2.38	2.45	2.57	3.66	2.80	2.80	**2.80**	2.80	2.25	2.43	2.60	3.66	2.28	2.80	2.75	2.80
7	↑	BSX2	3.13	3.42	**3.51**	4.03	2.28	3.22	3.13	3.32	3.25	3.42	3.48	3.76	3.13	3.25	3.24	3.32
	↔	ED2	2.46	2.57	**2.58**	2.70	2.36	2.44	2.45	2.63	2.35	2.52	2.51	2.67	2.27	2.45	2.44	2.54
	↓	JBLU1	-0.07	9.41	7.99	12.06	0.62	8.08	7.56	11.95	6.59	10.68	**10.28**	13.09	6.13	9.90	10.00	11.95
8	↑	MGA2	-6.38	-4.34	-4.50	-3.39	-4.69	-4.33	**-3.28**	-0.04	-4.87	-4.63	-3.29	0.00	-4.74	-4.61	-4.12	-1.47
	↔	MGA3	-1.09	-0.08	-0.13	0.51	-0.17	0.27	**0.23**	0.48	-1.96	-0.10	-0.08	1.34	-1.66	-0.21	-0.28	0.60
	↓	ATRO1	-8.86	9.75	8.49	16.08	5.29	9.10	8.68	11.31	4.76	9.51	9.17	11.51	6.32	9.51	**9.64**	12.18
9	↑	AVNW2	2.09	4.14	4.20	5.65	2.68	3.67	3.48	3.99	3.94	4.77	**4.63**	5.67	2.98	3.84	3.77	3.99
	↔	EXC5	-1.18	-0.40	-0.27	0.96	-0.61	0.44	**0.27**	0.87	-0.59	-0.35	-0.19	0.56	-0.95	0.40	0.31	1.80
	↓	AVNW3	1.75	1.96	1.97	2.14	1.75	1.92	1.94	2.10	1.86	2.01	**2.02**	2.20	1.78	1.95	1.92	2.01

Table 4: Overall average fitness by trend for each algorithm.

Trend	Algorithm GA	PSO	PSO^S	PSO^P
Downtrend	3.46	3.62	3.84	3.89
Sideways	0.61	1.01	0.74	0.76
Uptrend	0.31	0.81	0.55	0.80

Table 5: Average rankings of each algorithm according to the Friedman non-parametric test with the Holm post-hoc test over the mean performance. No statistical differences at the significance level 5% were observed.

Trend	Algorithm	Ranking	p-value	Holm
Downtrend	PSO^S (control)	2.0	–	–
	PSO	2.35	0.5444	0.05
	PSO^P	2.75	0.1939	0.025
	GA	2.9	0.119	0.0167
Sideways	PSO^P (control)	2.1	–	–
	PSO	2.2	0.8625	0.05
	GA	2.7	0.2987	0.025
	PSO^S	3.0	0.119	0.0167
Uptrend	PSO^S (control)	2.2	–	–
	GA	2.2	1.0	0.05
	PSO	2.75	0.3408	0.025
	PSO^P	2.85	0.2602	0.0167

We can see from Table 5 that PSO variants that employed the stochastic state update procedure (namely PSO^S and PSO^P) ranked highest across all three trend types, albeit not at a statistically significant level. This leads to two interesting observations. The first of these observations is that all three PSO variants experimented with here are competitive with GA in terms of performance when it comes to the domain of market timing. PSO can also produce these competitive results at a lower cost in terms of total number of fitness evaluations as can be seen from the algorithm configurations in Table 1. The second observation is that PSO^P, the PSO variant with pruning, is also competitive with the PSO variants without the pruning procedure as no statistical significance was observed in the results. Figure 5 shows a histogram of the solution lengths returned by PSO^P during testing. We can see that the majority of solution lengths were between 29 and 46 components, with only a single solution employing all 63 components. This suggests that PSO is capable of discovering shorter solutions without adversely affecting performance in a significant manner. This presents the opportunity of pursuing shorter and shorter solution lengths with the aim of finding the *least* satisfying subset of components that maximize our financial metrics. By finding shorter solutions, we will be capable of producing market timing strategies that execute faster and are more easily comprehensible.

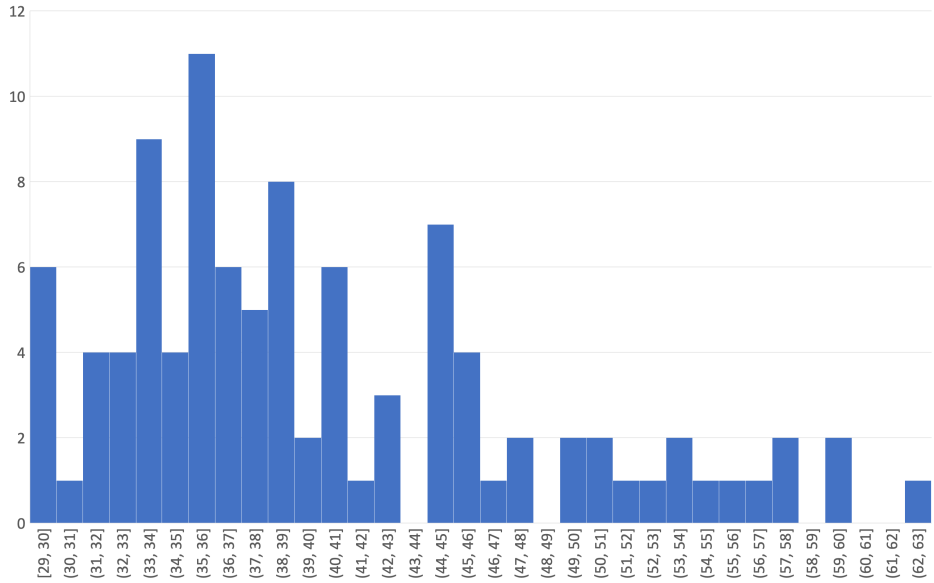

Fig. 5: A histogram showing the solution lengths of the solutions from PSO^P. We can clearly see that the majority of solutions range in length between 29 and 46. Only a single solution employed all 63 technical indicators.

8 Conclusion

In this chapter we revisited and extended the work presented in [20] by improving the details of trend representative testing and introducing a new PSO variant (PSO^P) with a novel pruning procedure. The results show that all three PSO variants are competitive to GA in terms of performance, and one particular variant was capable of achieving such a performance at a fraction of the number of fitness evaluations required. The results also show that the newly introduced PSO variant, PSO^P, was capable of producing competitive results in comparison to the other algorithm while returning solutions that are considerably shorter in length.

We suggest the following avenues of future research. First, use a more sophisticated measure of financial fitness. This would allow us to simulate hidden costs of trading such as slippage. Second, approach the problem of market timing as a multi-objective one by trying to maximize performance across the three types of trends and against multiple financial objectives. Third, pursuing shorter solution lengths by considering it as one of multiple objectives in a multi-objective approach to market timing. The validity of this pursuit is based on the evidence presented in the results returned by PSO^P, where the majority of the solutions returned did not use the full set of signal generating components available and yet remained competitive in terms of performance to the solutions generated by the other algorithms. Finally, adapt more metaheuristics to tackle market timing and compare their performance against the currently proposed ones in significantly larger datasets. We could then use meta-learning to understand if and when metaheuristics perform significantly better than others under particular

conditions and use that information to build hybrid approaches that use more than one metaheuristic to build strategies for market timing.

References

1. Abdelbar, A.: Stubborn ants. In: IEEE Swarm Intelligence Symposium, SIS 2008. pp. 1 – 5 (10 2008)
2. Allen, F., Karjalainen, R.: Using genetic algorithms to find technical trading rules. Journal of Financial Economics **51**(2), 245–271 (1999)
3. Bera, A., Sychel, D., Sacharski, B.: Improved Particle Swarm Optimization method for investment strategies parameters computing. Journal of Theoretical and Applied Computer Science **8**(4), 45–55 (2014)
4. Briza, A.C., Naval Jr., P.C.: Stock trading system based on the multi-objective particle swarm optimization of technical indicators on end-of-day market data. Applied Soft Computing **11**(1), 1191–1201 (2011)
5. Chakravarty, S., Dash, P.K.: A PSO based integrated functional link net and interval type-2 fuzzy logic system for predicting stock market indices. Applied Soft Computing **12**(2), 931–941 (2012)
6. Chen, S.M., Kao, P.Y.: TAIEX forecasting based on fuzzy time series, particle swarm optimization techniques and support vector machines. Information Sciences **247**, 62–71 (2013)
7. Clerc, M.: Think locally act locally-a framework for adaptive particle swarm optimizers. IEEE Journal of Evolutionary Computation **29**, 1951–1957 (2002)
8. Engelbrecht, A.P.: Fundamentals of Computational Swarm Intelligence. John Wiley & Sons Ltd (2005)
9. de la Fuente, D., Garrido, A., Laviada, J., Gómez, A.: Genetic algorithms to optimise the time to make stock market investment. In: Genetic and Evolutionary Computation Conference. pp. 1857–1858 (2006)
10. García, S., Fernández, A., Luengo, J., Herrera, F.: Advanced nonparametric tests for multiple comparisons in the design of experiments in computational intelligence and data mining: Experimental analysis of power. Inf. Sci. **180**(10), 2044–2064 (May 2010)
11. Hu, Y., Liu, K., Zhang, X., Su, L., Ngai, E.W.T., Liu, M.: Application of evolutionary computation for rule discovery in stock algorithmic trading: A literature review. Applied Soft Computing **36**, 534–551 (2015)
12. Kampouridis, M., Otero, F.E.: Evolving trading strategies using directional changes. Expert Systems with Applications **73**, 145–160 (2017)
13. Karathanasopoulos, A., Dunis, C., Khalil, S.: Modelling, forecasting and trading with a new sliding window approach: the crack spread example. Quantitative Finance **7688**(September), 1–12 (2016)
14. Kaufman, P.J.: Trading Systems and Methods. John Wiley & Sons, Inc, 5th edn. (2013)
15. Kim, Y., Ahn, W., Oh, K.J., Enke, D.: An intelligent hybrid trading system for discovering trading rules for the futures market using rough sets and genetic algorithms. Applied Soft Computing **55**, 127–140 (2017)
16. Ladyzynski, P., Grzegorzewski, P.: Particle swarm intelligence tunning of fuzzy geometric protoforms for price patterns recognition and stock trading. Expert Systems with Applications **40**(7), 2391–2397 (2013)
17. Liu, C.F., Yeh, C.Y., Lee, S.J.: Application of type-2 neuro-fuzzy modeling in stock price prediction. Applied Soft Computing **12**(4), 1348–1358 (2012)
18. López-Ibáñez, M., Dubois-Lacoste, J., Pérez Cáceres, L., Stützle, T., Birattari, M.: The irace package: Iterated racing for automatic algorithm configuration. Operations Research Perspectives **3**, 43–58 (2016)

19. Mohamed, I., Otero, F.E.: Using Particle Swarms to Build Strategies for Market Timing: A Comparative Study. In: Swarm Intelligence: 11th International Conference, ANTS 2018, Rome, Italy, October 29–31, 2018, Proceedings. pp. 435–436. Springer International Publishing (2018)
20. Mohamed., I., Otero., F.E.B.: Using population-based metaheuristics and trend representative testing to compose strategies for market timing. In: Proceedings of the 11th International Joint Conference on Computational Intelligence - Volume 1: ECTA, (IJCCI 2019). pp. 59–69. INSTICC, SciTePress (2019)
21. Patterson, S.: Dark Pools: The Rise of A.I. Trading Machines and the Looming Threat to Wall Street. Random House Business Books
22. Penman, S.H.: Financial Statement Analysis and Security Valuation. McGraw-Hill (2013)
23. Pring, M.: Technical Analysis Explained. McGraw-Hill (2002)
24. Soler-Dominguez, A., Juan, A.A., Kizys, R.: A Survey on Financial Applications of Metaheuristics. ACM Computing Surveys **50**(1), 1–23 (2017)
25. Subramanian, H., Ramamoorthy, S., Stone, P., Kuipers, B.: Designing Safe, Profitable Automated Stock Trading Agents Using Evolutionary Algorithms. In: Genetic and Evolutionary Computation Conference. vol. 2, p. 1777 (2006)
26. Sun, Y., Gao, Y.: An Improved Hybrid Algorithm Based on PSO and BP for Stock Price Forecasting. The Open Cybernetics & Systemics Journal (2015)
27. Wang, F., Yu, P.L., Cheung, D.W.: Combining technical trading rules using particle swarm optimization. Expert Systems with Applications **41**(6), 3016–3026 (2014)