USING PARTICLE SWARM OPTIMIZATION FOR MARKET TIMING STRATEGIES

a thesis submitted to
The University of Kent
in the subject of computer science
for the degree
of doctor of philosophy.

By
Ismail Mohamed Anwar Abdelsalam Mohamed
March 2021

Abstract

Market timing is the issue of deciding when to buy or sell a given asset on the market. As one of the core issues of algorithmic trading systems, designers of such system have turned to computational intelligence methods to aid them in this task. In this thesis, we explore the use of Particle Swarm Optimization (PSO) within the domain of market timing.

PSO is a search metaheuristic that was first introduced in 1995 [28] and is based on the behavior of birds in flight. Since its inception, the PSO metaheuristic has seen extensions to adapt it to a variety of problems including single objective optimization, multiobjective optimization, niching and dynamic optimization problems.

Although popular in other domains, PSO has seen limited application to the issue of market timing. The current incumbent algorithm within the market timing domain is Genetic Algorithms (GA), based on the volume of publications as noted in [40] and [84]. In this thesis, we use PSO to compose market timing strategies using technical analysis indicators. Our first contribution is to use a formulation that considers both the selection of components and the tuning of their parameters in a simultaneous manner, and approach market timing as a single objective optimization problem. Current approaches only considers one of those aspects at a time: either selecting from a set of components with fixed values for their parameters or tuning the parameters of a preset selection of components. Our second contribution is proposing a novel training and testing methodology that explicitly exposes candidate market timing strategies to numerous price trends to reduce the likelihood of overfitting to a particular trend and give a better approximation of performance under various market conditions. Our final contribution is to consider market timing as a multiobjective optimization problem, optimizing five

financial metrics and comparing the performance of our PSO variants against a well established multiobjective optimization algorithm. These algorithms address unexplored research areas in the context of PSO algorithms to the best of our knowledge, and are therefore original contributions. The computational results over a range of datasets shows that the proposed PSO algorithms are competitive to GAs using the same formulation. Additionally, the multiobjective variant of our PSO algorithm achieved statistically significant improvements over NSGA-II.

Acknowledgements

I would like to thank Dr. Fernando Otero and Prof. Peter Rodgers for their guidance and mentoring throughout the whole process. I would also like to thank my parents, my wife, my daughter, my family and my friends for supporting me throughout the entire journey. This is for you.

Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

> *"There is nothing more difficult to take in hand, more perilous to conduct, or more uncertain in its success, than to take the lead in the introduction of a new order of things."*
>
> – **Nicolo Machiavelli**

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 investor orders to buy and sell, which further exasperated their losses. One of these measures introduced 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 besides the established exchanges, such as Nasdaq and the New York Stock

Exchange (NYSE), 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 exchange 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 [73].

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 five 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 fourth 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. The final issue a designer has to consider is the *"frequency"* of trading. This can be anywhere in between numerous executions per second (High Frequency Trading) to a few trades being executed every few months or longer (Value Trading).

A large number of computational intelligence techniques were used in the development of algorithmic trading systems, and one of these methods is Particle Swarm Optimization [40][84]. Particle Swarm Optimization (PSO) is a search metaheuristic first proposed by Eberhart and Kennedy in 1995 [28]. Originating from simulations of flocks of birds in flight, PSO uses the emergent properties of a swarm of entities, called particles, as they traverse a solution landscape seeking optimal positions while interacting with the those around them. Each particle in the swarm represents a candidate solution to the problem being solved. The particles follow a relatively simple dynamic in their search. First, particles will consider their previous movement across the solution landscape, and will follow that with a diminishing bias. Second, the particles would consider their last position across the landscape where they fared best and how far away they have wandered. Finally, the particles would consider their neighbors, how far better they are faring in their search for an optimum solution and will try to inch closer to the better performing ones. By continuously factoring in these three aspects, particles start from being randomly scattered across the solution landscape to quickly converging on an optimum, or as close an approximation to it. PSO was first proposed as a nonlinear function optimizer, and counted the optimization of weights on connections in a neural network as one of its first applications. Since then it has been vastly extended to solve multiple classes of optimization problems, including single-objective optimization, multiobjective optimization, constrained optimization and dynamic optimization problems. A lot of extensions have also been introduced that fused PSO with other metaheuristic paradigms to form hybrid approaches [30].

Despite its popularity in other domains, PSO has seen limited use in the financial domain and, in particular, within the market timing space. As mentioned earlier, market timing is the issue of identifying when to buy or sell a given tradable item in a market. This observation is in comparison to other metaheuristics

such as genetic algorithms (GA) and genetic programming (GP)[40][84]. Within a few years of the introduction of electronic exchanges and trading in the mid 1990s, GA has become the most used metaheuristic to guide trading decisions and form the core of market timing strategies based on the sheer volume of publications alone [40][84]. The earliest PSO approach to market timing, on the other hand, was introduced in 2011. Despite showing competitive performance with GA in some applications [49][29], the application of PSO to the domain of market timing has not been thoroughly investigated in comparison to GA.

In this thesis, we explore the use of PSO to tackle market timing. Current approaches to market timing will either consider the tuning of the parameters of a preset selection of components that constitute a market timing strategy, or select from a set of components with predefined parameter values. This has been observed regardless of the algorithm being used as noted by Hu et al. [40]. Being constrained to only performing one of the functions (selection of components or the tuning of their parameters) limits the scope of components a designer can consider for their market timing strategy both in terms of quantity and type. We start by addressing this limitation via the introduction of a new formulation that considers both the selection of components that form a market timing strategy and the tuning of their parameters in a simultaneous fashion. We then adapt PSO to tackle market timing based on this formulation while only optimizing a single financial metric. Another limitation to current approaches in market timing pertains to how the algorithms are trained and tested, leaving them liable to overfitting to particular market trends. This leads to a suboptimal performance when these algorithms are deployed in live trading and encounter market trends that are different from the ones they have overfit to during development. We address this limitation by adopting a novel approach for training and testing that we call Trend Representative Testing. This proposed methodology is then evaluated by comparing the performance of our PSO algorithms against a Genetic Algorithm (GA) that was adapted to tackle market timing using the same formulation as our PSO models in an extensive test. So far, we have considered market timing as a single objective optimization problem whereby a single financial metric is being optimized. This is not the case in real trading and designers of algorithmic trading systems would seek to optimize a number of financial metrics representing various

aspects of profits, losses and exposure to risk. As a result, our final contribution in this thesis is considering market timing as a multiobjective optimization problem and adapting our PSO and GA algorithms to tackle it accordingly.

1.1 Thesis Structure

The remainder of this thesis is structured as follows. We begin by introducing the reader to the issue of market timing and discuss current approaches in **Chapter 2**. We also consider how market timing performance is evaluated and introduce the financial metrics that are to be optimized by our algorithms. In **Chapter 3** we discuss the basis of the algorithms to be used in building market timing strategies and discuss the concepts of single and multi- objective optimization. This is followed by a look at the related work to PSO, GA and multiobjective optimization in **Chapter 4**. **Chapter 5** describes how we adapted PSO to tackle market timing while using a formulation that considers both the selection of the components that constitute a strategy as well as tune their parameters in a simultaneous fashion. **Chapter 6** introduces a novel training and testing approach for market timing strategies as well provide an extensive comparison between the performance of PSO and GA in composing such strategies. **Chapter 7** shows the transition from considering market timing as a single objective optimization problem to a multiobjective one, and shows the adaptations that were needed in terms of the algorithms to make this transition possible. Finally, in **Chapter 8** we conclude by providing a summary of our contributions and suggestions for future research.

1.2 Publications

During work on this thesis, a number of publications were produced and these form the basis of some of the material discussed in the following chapters. These publications are listed below along with which chapters they influenced.

- Ismail Mohamed and Fernando E. B. Otero (2018). 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, Springer International Publishing, pp. 435–436. This was the first publication and took the form of an extended abstract. It describes the first attempt at adapting PSO to tackle market timing based on a formulation that considers both the selection of components that make up a strategy as well as the tuning of their parameters. This work influenced Chapter 5.

- Ismail Mohamed and Fernando E. B. Otero (2019) Using population-based metaheuristcs 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)*, INSTICC, SciTePress, pp. 59-69. This is a full paper that introduces a novel training and testing approach known as trend representative testing as well as an extensive test comparing the performance of PSO and GA in producing market timing strategies. This influenced the material seen in Chapter 6. This paper was shortlisted for the best student paper award.

- Ismail Mohamed and Fernando E. B. Otero (2021) Building Market Timing Strategies Using Trend Representative Testing and Computational Intelligence Metaheuristics. In Computational Intelligence, IJCCI 2019, Revised Selected Papers (Book Chapter). This is an extended version of the ECTA 2019 paper and influences Chapter 6 as well.

- Ismail Mohamed and Fernando E. B. Otero. (2020) A multiobjective optimization approach for market timing. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference (GECCO '20).* Association for Computing Machinery, New York, NY, USA, 22–30. This is a full paper that sees the transition to considering market timing as a multiobjective optimization problem. This paper influences the material in Chapter 7. This paper was shortlisted for a best paper award in its track within the conference.

- Ismail Mohamed and Fernando E.B. Otero (2021). A Novel Multiobjective Particle Swarm Optimization Algorithm for Market Timing. In Transactions on Evolutionary Computation (Submitted). This is an extended version of

the GECCO '20 paper and includes comparisons with two additional benchmarks. This journal paper also influences the material in Chapter 7.

Chapter 2

Market Timing and Financial Metrics

"The public, as a whole, buys at the wrong time and sells at the wrong time."

– **Charles Dow**

As alluded to in the previous chapter, market timing is concerned with deciding when to buy or sell a given security on the market and forms the central problem being tackled by the algorithms in this thesis. In this chapter, we take a deeper look at what market timing means, how market timing strategies are formed and how their performance can be evaluated.

2.1 Market timing

Market timing can be formally defined as the problem of deciding when to buy or sell a given security on the market based on expectations of future price movements. Expectations of where prices could be in the near future can be used to guide a trader's actions. For example, if the expectation on a given asset on the market is that prices will go up, a trader can choose to buy the given asset and sell at a future date when the prices have gone up to expectation, profiting from the difference in prices. If the expectation for the price is to go down, a trader can choose to sell any held assets to avoid incurring a potential loss. If the trader does not hold any

assets that are affected by an expected downturn in prices, the trader might choose to sell the affected assets by loaning them from another trading entity and then redeeming them at a lower price, thus profiting from the downturn. This practice is known as short selling. The issue of market timing is one of the core issues faced by the designer of a trading system along with defining system objectives, securities traded (also known as portfolio composition) and execution optimization as mentioned in the previous chapter.

A common strategy to tackle market timing is to use a collection of components, where every component t consumes information regarding a security and returns a signal indicating whether to buy or sell [44]. A component's signal is limited to three values: 1 for a buy recommendation, -1 for a sell recommendation and 0 for a hold recommendation. Every component has a weight and a set of parameters. The parameters control the behavior of a component and they are unique to each component type. The weight controls how much influence the component has on the overall signal produced by the candidate market timing strategy or solution. The overall signal of the candidate solution is taken as the aggregation of weighted components, and interpreted as follows: buy when positive, sell when negative or hold otherwise. Formally, we can present this formulation as follows:

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

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

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. The varying combinations of components, along with varying values for component weights and parameters, produce a rich landscape of candidate solutions that return different signal values for the same market conditions.

Traders in financial markets have continuously looked for ways to best decide when to take action with a given security, and over time developed numerous components that can digest market context and produce signals [44]. Two distinct

schools of thought have emerged over time in terms of how market context should be processed, and hence how signals are produced. The components we have available today can be categorized to belonging to either one of these schools. These schools are technical analysis and fundamental analysis. Over the next few subsections, we discuss the reasoning behind each of these schools of thought, how traders would build market timing strategies based on components from each school and our choice in how we use components in this thesis.

2.1.1 Technical Analysis

Technical analysis can be defined as the analysis of a security's historical price and volume movements, along with current buy and sell offers, for the purposes of forecasting its future price [44][74]. 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. Charles Dow helped set the foundations of modern technical analysis with his Dow Theory, introduced in the 1920s. The philosophy behind technical analysis is built upon three pillars: price discounts all the information we need to know about the traded security, prices move in trends and history has a likelihood of repeating itself. The first pillar assumes that all the forces that can affect the price of a security have been accounted for and already exerted their influence when the actual trade took place. This includes the psychological state of the market participants, the expectations of the various entities trading in that particular security, the forces of supply and demand and the current state of the entity which the stock represents amongst other factors. It is therefore sufficient to only consider price movements and their history, as they are a reflection of all these forces and their influence. The second pillar assumes that prices move in trends based on the actions of traders currently dealing in that security and their expectations. For example, if traders expect that demand will increase for a particular stock they would buy that particular stock in order to resell 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 to 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 pillar assumes that markets, presented with an almost similar set of stimuli and circumstances, have a tendency of reacting in the same fashion as it had in previous exposures. This was proven empirically in the histories of various securities across many markets over time as traders react in the same consistent fashion to shifts in price [44].

Techniques employing technical analysis will often take the form of functions known as indicators. Indicators will take in price history, along with a set of parameters that govern various aspects of an indicator's behavior, and return a signal – an indication of whether it is favorable to buy or sell at the current moment. An example of a technical analysis indicator is the Moving Average Converge Diverge (MACD) indicator [67]. MACD depends on studying the exponential moving averages of the price data in an attempt to identify the momentum behind the underlying trend in the price. The MACD indicator uses two exponential moving averages, a fast one and another slow one, and the period of both being user-defined parameters. By subtracting the fast moving average from its slow counterpart, we arrive at a new series called the MACD series. We then further smooth the MACD series by a user defined parameter to generate what is known as the Signal line. Crossover points between the Signal line and the MACD series indicate an imminent change in momentum and a subsequent change in the direction of the current price trend. Depending on the direction of the crossover, we either buy (when the signal crosses up) or sell (when the signal line crosses down). An example of this can be seen in Figure 1.

As an exhaustive list of all indicators currently available to the modern trader would easily fill multiple volumes, it is beyond the scope of this thesis. Instead, the reader is directed to the works of Pring [76] and Kaufman [44] for a more detailed look at the world of technical analysis.

Figure 1: An example of the MACD indicator[1]. The top chart shows the raw price data of the Nasdaq 100 ETF (QQQ). The bottom chart shows the MACD technical indicator applied on the QQQ data. The blue and red lines represent the MACD series and the Signal line respectively. Two crossover events were detected and are marked by the arrows. The first event shows the Signal line crossing down over the MACD series, indicating that the momentum for the upwards movement has ended and we are now in a downtrend, hence generating a sell signal. The opposite of these circumstances are detected in the subsequent event, generating a buy signal.
[1] Image from Yahoo! Finance

2.1.2 Fundamental Analysis

Fundamental analysis is the process of deriving the value of a security by analysis of the financial state of the company it represents [74]. This will include analyzing current and previous financial documents and accounting records for the company, considering current management personnel and their performance in the past, sales performance history, earning history, current market sentiment towards the company and macroeconomic conditions amongst many other factors. After considering these factors, analysts can arrive at a fair value for the security and a projection for it moving forward. Fundamental analysis is built on the core assumption that a discrepancy occurs between a security's fair price and market price as the market moves to close that gap. Recommendations for buying or selling the security are therefore based on identifying this discrepancy and how best to utilize it to achieve returns.

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 [68][93]. 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.

2.1.3 Discussion

The existence of these two methods leaves us with a choice: should a trader build their strategy on technical methods or fundamental ones? Although experts on each side would stand by their purest approaches to use one school of thought over the other, a large number of traders would base their strategies on techniques from both sides. It is also quite common for traders to use fundamental analysis for portfolio composition, then use technical analysis for market timing. It would actually be prudent to use methods from both schools as that would hedge the trader's risk in one or more of the components being mistaken or fed false data[1], and thus produce signals that might result in losses. At the end of the day, it depends on what a trader is comfortable with using and their financial objectives to choose components from either or both schools. For practical aspects, we limit our choice of components in this thesis to those of the technical analysis nature. The reasons behind that is that open-sourced implementations of numerous technical indicators are widely available and the data required to use these components (price histories) is freely available. Fundamental analysis components, on the other hand, require the use of market data that is not as freely available as their technical analysis counterparts.

[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

2.2 Backtesting

Before deploying a market timing strategy, it is important to validate that the concepts behind it are sound and that it is likely to reach the financial goals it was set out to achieve. This is done via backtesting: a process where the strategy's recommendations are executed against historical data with the aims of simulating a real market. During this process of emulated market trading, we keep track of a number of metrics, discussed in the next section, that represent attained profits and losses incurred. We start with a stream of price action data pertaining to a particular security traded on the stock market. The data points on this stream could represent the prices at the end of every second, minute, day or other time interval depending on the type of trading we want to emulate. A window of the previous $(n-1)$ prices is then fed to all the components that constitute a strategy in order to arrive at a recommendation or signal for the current time interval (n). To get the overall signal for time interval n, we simply aggregate the individual signals produced by each component of the strategy multiplied by its respective weight. If the overall signal is to buy, then a buy order is generated and executed against the price data, and vice versa. During order execution, we keep track of the impact that has on the capital alloted for investment and the shares currently held. This process is played forward until the price stream is consumed. For the purposes of this thesis, we emulate a day trading process whereby the data points on our price streams represent the prices achieved by a particular security at the end of the day. As for the types of orders, we use a simple market order type with immediate execution. When a buy signal is attained, the backtesting procedure would use all available capital to buy shares at current market price. When a sell signal is received, all current shares are liquidated at current market price. If no shares are held, the available capital is used to sell the maximum amount of shares possible on credit only to buy them back and repay the creditor (the system in this case) at the next buy signal. This is also known as short selling. Any shares held at the end of the backtesting procedure are liquidated at the last market price observed on the last time interval in the testing price stream.

---

documentation were false and misleading, leading to investments by market participants that were built on inaccurate assumptions.

This leads us to the questions: *Which market data to use to simulate the market? How much data should be used?* The typical process is to start with a strand of historical prices at the time interval to be used for backtesting. The strand of history is then partitioned into two: training and testing, with the training partition taking a chronologically earlier section of the strand than testing. The training section is exposed to the strategy and trades are executed against it. The performance of the system is then assessed and if found unsatisfactory, the strategy is tuned and adjusted. Once satisfactory, the system is then run against the testing section of the strand. If at this point the system fails to achieve its goals, then there is something fundamentally amiss with the core strategy and the trader should rethink it. This style of backtesting is known as Step Forward Testing [44], and it is the incumbent method in training and testing when it comes to publications using computational intelligence techniques for market timing [84, 40]. The problem with Step Forward testing is that the system might not be exposed to a variety of price movement trends. It might only be exposed to say, an uptrend, that persists throughout training and testing. Although performing well in backtesting, the system is likely to suffer when encountering a downtrend or a sideways movement of prices during live trading. An example of this can be seen in Figure 2. As we can see here, the underlying trend persistent throughout both the training and testing sections of the data is an upwards movement. We can therefore expect the market timing strategy to perform well when encountering an upwards trend in live trading, but the performance will suffer if we encounter a downwards trend for example. This liability of overfitting has been noted in surveys on the use of computational intelligence techniques for market timing [84, 40] and in publications regarding the synthesis of trading systems [44]. To remedy this problem, a trader might opt to backtest against a longer strand of history, where multiple trends can be seen across time. But what if the history of the security selected for backtesting does not have a variety of trends? An alternative approach would be to backtest against multiple strands of history, from multiple securities, that explicitly undergo various price movement trends and the average performance across these various strands is taken as indicator of the strategy's quality. The latter approach is seen as the superior of the two, and it is the current recommendation by market professionals such as Kaufman [44]. We explore

Figure 2: 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 from Yahoo! Finance

the use of backtesting using Step Forward testing in Chapter 5 and the transition to a more robust training and testing methodology that attempts to remedy the shortcomings of Step Forward testing in Chapter 6.

2.3 Financial Performance Metrics

As mentioned in the previous section, backtesting is used to assess the performance of a candidate market timing strategy based on one or more financial metrics. These financial metrics measure the effects the transactions produced by the market timing strategy on the initial capital committed to investment in terms of profits gained, losses incurred, exposure to risk and confidence in these measures. An underlying assumption adopted by all the metrics presented in this section is that they follow a Normal or Gaussian distribution. This assumption affects how

the metrics are scaled to annual levels, the default values of confidence factors selected and the impact of data sample size on error. In the following subsections, we discuss the financial metrics we use during backtesting in one or more of the experiments in this thesis. These metrics are therefore considered as viable objectives to optimize either individually or simultaneously.

2.3.1 Annualized Rate of Returns (AROR)

The Annualized Rate of Returns (AROR) is defined as the amount of returns on initial capital invested adjusted to reflect an annual rate. AROR can be calculated as follows:

$$\text{AROR}_{\text{simple}} = \frac{E_n}{E_0} \times \frac{252}{n} \tag{3}$$

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 backtesting period. AROR values are interpreted based on initial capital, final capital and the length of the backtesting period. For example, given an initial and final capital of 1000000, and a backtesting period of 252, a value of one would indicate breakeven. Values above one would indicate gains, while values below one indicate losses. When used as a metric to be optimized, AROR values are maximized.

2.3.2 Annualized Portfolio Risk

Annualized portfolio risk is defined as the volatility of returns encountered during trading within a specific time period. Annualized portfolio risk is calculated as follows:

$$\text{Risk} = \sigma(\text{returns}) \times \sqrt{252} \tag{4}$$

where $\sigma(returns)$ is the standard deviation of the returns on the trades performed during backtesting. The standard deviation is multiplied by the square root of 252 (the typical number of trading days in an American calendar) to return an annualized figure. Since the annualized portfolio risk is presented in units of currency, ideally you would want to minimize this variance in returns as much as possible. A strategy with low variance means that the gains obtained were evenly distributed

across the transactions that were made throughout the lifetime of the strategy. This would indicate that the strategy is stable – more or less of such transactions would result in gains or losses that can be easily estimated. On the other hand, a strategy with a relatively high variance means that some transactions resulted in low gains or losses, while others resulted in much higher gains or losses. A change in the volume of transactions would result in an unknown number of gains or losses, as we would be unable to estimate the impact of a single transaction based on previous results. This would indicate that the strategy we have at hand is an unstable one. It is, therefore, important that we reduce Portfolio Risk as much as possible. When used as a metric to be optimized, Portfolio Risk values are minimized.

2.3.3 Value at Risk (VaR)

Value at risk represents the likely value of the initial capital in terms of units of currency we are liable to lose based on a given confidence level. This is calculated as follows:

$$\text{VaR} = \mu(\text{daily returns}) - c\sigma(\text{daily returns}) \tag{5}$$

where *daily returns* represents the returns achieved day by day during backtesting, μ represents the mean of those returns and σ represents the standard deviation of the daily returns. The confidence level c is a user controlled parameter that represents the statistical confidence of losses the user would like to see. A default value of 1.65 (representing 95% confidence) is the most used one. When used as a metric to be optimized, VaR values are minimized. Since VaR values are negative in nature, they are usually multiplied by -1 as a matter of convenience during optimization.

2.3.4 Solution Length

Solutions that are longer in length imply the involvement of a large number of signal generating components. The solution length is determined by counting the number of components within the solution that have a weight above zero, i.e. components that have a positive contribution towards the aggregate signal that is

generated. Any components that have a weight of zero have their contributions to the aggregate signal nullified, and are therefore not considered for solution length. When used as a metric to be optimized, solution length is usually minimized. This is based on the fact that shorter solutions imply the use of a fewer number of components and thus would incur less of a computational cost compared to longer solutions. Shorter solutions are also more comprehensible by the end user when compared to their longer counterparts, and thus more desirable. It is for these two reasons that solution length are usually minimized when used as a metric.

2.3.5 Transactions Count

The number of transactions produced by a market timing strategy is significant in that it gives an indication of how stable that strategy is, and that is based on the concept of sample error. Given a set of returns, the approximate error in that set can be gauged using standard deviation. Higher values of standard deviation would indicate that the returns are erratic or volatile, and that measure is calculated in a form under the Annualized Portfolio Risk. Our confidence in that measure of volatility increases as the number of available data points increases, which in turn reduces the sample error. Sample error can be calculated as follows:

$$\text{Sample Error} = \frac{1}{\sqrt{N}} \tag{6}$$

where N is the number of data points, where a data point represents a single transaction. It is therefore advantageous to maximize the number of data points available, and hence maximize the number of transactions generated by the market timing strategy when using transaction count as a metric to be optimized. As transactions have a cost in real trading, a system that generates a disproportionate number of transactions would result into a hefty cost that could obliterate any returns gained. In order to avoid such situations, and reach reasonable trade-offs, we simulated transaction cost while backtesting based on a fixed commission model. To calculate this value, we first calculate a working commission, which is a fixed value charged per share being bought or sold in the transaction. If the

working commission does not drop below a minimum threshold or exceed a maximum threshold, then the working commission is considered the transaction cost; otherwise, the minimum or maximum threshold is considered the transaction cost depending on whether the working commission was below the former or exceeded the latter. This fixed commission model is based on Interactive Brokers (IB) fixed commission model [2]. The values used for the parameters of the commission model are:

- Per share: \$0.005
- Minimum Threshold: \$1.0
- Maximum Threshold: 1% of total transaction value.

2.3.6 Sharpe Ratio

Described by William F. Sharpe in [81], the Sharpe Ratio is a composite metric that combines both the rate of returns (AROR) and portfolio risk. The Sharpe Ratio is calculated as follows:

$$\text{Sharpe Ratio} = \frac{\text{AROR} - \text{Benchmark Returns}}{\text{Annualized Portfolio Risk}} \tag{7}$$

The benchmark returns in the above equation represents risk free returns that would have been obtained for the same trading period had no trading action have taken place. A usual benchmark used to model this would be investments in fixed rate government bonds over the same period of trading in question. A variation of the Sharpe Ratio exists that uses a benchmark with some inherit risk, such as investing in an Exchange Traded Fund (ETF), and is known as the Information Ratio.

[2] https://www.interactivebrokers.com/en/index.php?f=1590&p=stocks1

Chapter 3

Particle Swarm Optimization (PSO)

"How does nature amplify the intelligence of groups? It forms swarms."

– **Louis B. Rosenberg**

In this chapter, we take a deep look into the Particle Swarm Optimization (PSO) metaheuristic. We start by defining what a metaheuristic is in general, and the two types of optimization problems metaheuristics are used to solve. This is followed by a discussion of the inception of the PSO algorithm and the practical issues involved in its implementation. We then end our look into PSO by presenting the various extensions to the algorithm introduced in the literature to either improve its performance or adapt it to solve particular categories of problems.

3.1 What is a Metaheuristic?

While heuristics are problem specific strategies to find solutions, metaheuristics are algorithms that exist at a higher level of abstraction. Metaheuristics are strategies that guide a search over a solution landscape, agnostic of the actual fitness function [10]. Metaheuristics will often strike a balance between exploring the overall landscape, and exploiting regions of high quality solutions. Research in metaheuristics

can be traced as far back as the 1980s, with the introduction of the Simulated Annealing algorithm. Since then, more metaheuristics were introduced to the scene, including Genetic Algorithms, Genetic Programming, Greedy Randomized Adaptive Search Procedure (GRASP), Tabu Search, Memetic Algorithms, Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO).

Although many taxonomies were introduced in literature to categorize metaheuristics, the one proposed by Glover and Sörensen in [34] was found to be quite succinct. In it, a metaheuristic can be categorized as belonging to one of the following categories:

- Local search metaheuristics: These algorithms iteratively make small changes to the solution representation in an attempt to improve solution quality. In effect, the algorithm searches the immediate neighbors of the solution at hand for a better candidate, and hence the moniker local search. Simulated Annealing and Tabu search are examples of such metaheuristics.

- Constructive metaheuristics: These algorithms start with a single component and progressively add other components till a full candidate solution is attained. Greedy Randomised Adaptive Search Procedure (GRASP) is an example of such a metaheuristic [38].

- Population based metaheuristics: These algorithms rely on a collection of entities, where each entity represents a candidate solution. The entities then interact in order to arrive at new candidates, and the process continues till improvement in the candidates produced ceases, or some other stopping criteria is reached. Genetic Programming [53] and Genetic Algorithm [37] were among the first population based metaheuristics .

- Hybrids: These metaheuristics combine the dynamics of one or more of the aforementioned categories. Particle Swarm Optimization (PSO) [28] can be considered a hybrid metaheuristic as it combines elements from both local search and population based approaches.

3.2 Optimization

When considering search over a solution landscape, metaheuristics are designed to seek areas of optimality. These areas are defined by providing optimum values for one or more objective functions – a quantity that is either minimized or maximized in order to find a solution to the problem being tackled by the metaheuristic. We can, therefore, look at search metaheuristics as tools for solving optimization problems. Optimization problems are generally categorized into either being single objective or multiobjective optimization problems. In the following two subsections we will look into what is entailed by each type of optimization problem in terms of requirements from a metaheuristic to tackle each type.

3.2.1 Single Objective Optimization

Single objective optimization problems are defined as those with a single point of optimality on their solution landscapes. If the problem is of the minimization variety, then that point would the one that produces the least value, compared to all other points, from the fitness function related to that problem. Let x^* denote the point of most optimality, then the previous statement can be presented formally as:

$$f(x^*) < f(x), \forall x \in S \tag{8}$$

where x represents all valid points on the search landscape S. If the problem is of the maximization variety, then the less than sign is exchanged for a greater than sign in the previous equation. The aim of a metaheuristic tackling a single objective optimization problem is to locate x^*, or come as close to it as it can.

3.2.2 Multiobjective Optimization

Multiobjective Optimization Problems (MOPs) are defined as problems that entail the optimization of more than one fitness metric [30]. Assuming minimization, this can be represented as:

$$
\begin{aligned}
&\textit{minimize} && F(x) = [f_1(x), f_2(x), ..., f_k(x)] \\
&\textit{subject to} && g_i(x) \leq 0, \ \textit{for } i = 1, 2, ..., n \\
& && h_i(x) = 0, \ \textit{for } i = 1, 2, ..., m
\end{aligned}
$$

where

- x represents a candidate solution to the problem being solved.
- $F(x)$ represents the set of k objectives being optimized.
- $g_i(x)$ represents the set of n inequality constraints.
- $h_i(x)$ represents the set of m equality constraints.

As multiple fitness metrics are being optimized, it is likely that scenarios would arise where some of the objectives will be in contradiction to one another. This means that optimizing one metric could be detrimental to another in the set of metrics being optimized. This complicates the notion of which solution would be considered *optimal*. The notion of optimality in the context of multiobjective optimization problems is discussed next.

3.2.2.1 Optimality, Dominance and Pareto Fronts

Hypothetically, let us assume that we are tackling a multiobjective optimization problem to minimize two fitness metrics. We are presented with three candidate solutions to the problem: $x_1(3, 4)$, $x_2(4, 3)$ and $x_3(5, 5)$. Which solution would be considered the more *optimal* one? This leads us to the notion of dominance. Dominance between solutions can be defined as:

Definition 1. ***Dominance:*** *Given two solutions, x and y, we can say that solution x dominates solution y ($x \prec y$) if:*

- *x is not worse than y across all fitness metrics being optimized.*
- *x performs better than y in at least one fitness metric.*

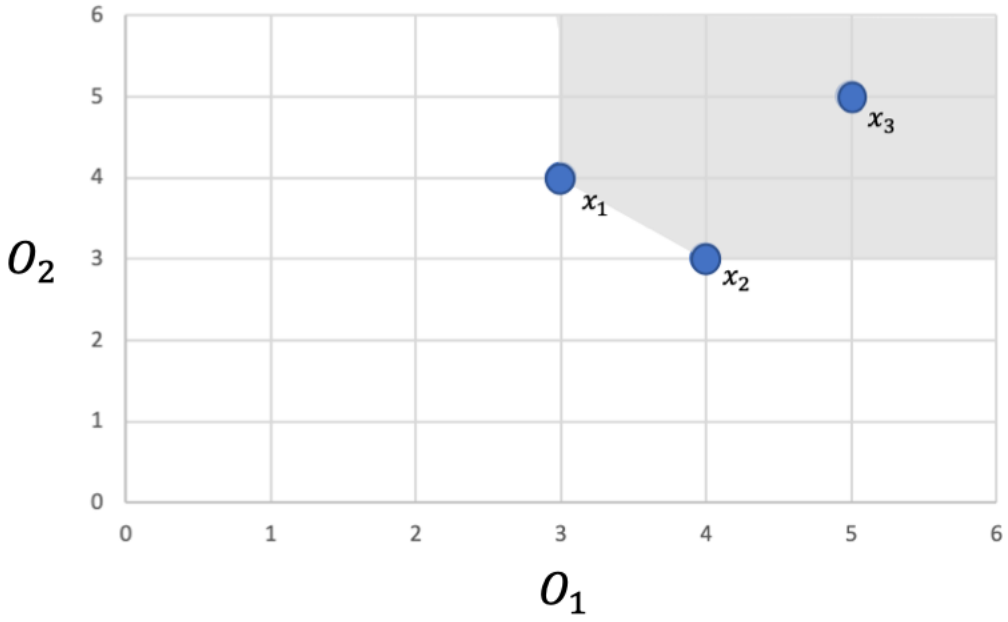

Figure 3: A graph showing an example of dominance. Both solutions x_1 and x_2 dominate solution x_3 when it comes to minimizing both the O_1 and O_2 fitness metrics. The shaded region shows the area dominated by x_1 and x_2.

From our example, we can say that both x_1 and x_2 are better than x_3 in terms of fitness, as both have lower values for both metrics being optimized. They also show dominance to solution x_3 according to definition 1. This can be seen in Figure 3. Dominance is therefore the primary measure of optimality in multiobjective domains. This is in contrast with single objective optimization that have a simpler measure of optimality: the value of the metric itself. Then, what about x_1 and x_2, which would be considered the more dominant, and hence more optimal? Both x_1 and x_2 have an advantage over the other in one of the metrics, and thus both do not comply to definition 1. Both solutions are considered *non-dominated*, and hence both are considered as viable solutions to the problem. In this sense, solving multiobjective optimization problems becomes a search for all non-dominated solutions within the search space. This leads us to the notion of Pareto Optimality: a solution is considered Pareto optimal if there exists no other solution within the search space that dominates it. The set of all Pareto optimal solutions is known as a Pareto set, and their corresponding fitness values is known as the Pareto front.

The discovery of this set, and hence its corresponding front, is then the goal of all metaheuristics tackling multiobjective optimization problems.

3.2.2.2 Schemes for Solving Multiobjective Optimization Problems

Although discovering the Pareto Front within a multiobjective optimization problem's search space is the theoretical answer, it is not the only practical way in tackling such problems. In this section, we list the types of schemes used to tackle MOPs across the literature. These definitions are based on [30, 79].

- **Aggregation Schemes** – One of the simplest ways in tackling MOPs is to combine the fitness metrics into a new single metric that is optimized. This recasts the problem as a single optimization problem, and allows metaheuristics that are designed to solve single optimization problems to tackle MOPs. The various fitness metrics can be assigned either static weights in the new unified metric that do not change over the course of the search, or dynamic ones whose values do change throughout the search process. The issue with using the aggregation scheme is that metaheuristics that employ such a scheme will be geared towards single objective optimization, and thus will return a single answer. Measures will have to be taken to return an approximation of the Pareto front. This could include running the metaheuristic a number of times instead of just once, or using a niching variant of the metaheuristic to return solutions from different regions of the search space.

- **Criterion or Lexicographic Schemes** – In this scheme, the fitness metrics involved in the MOPs are first ranked in terms of priority. The metaheuristics then optimize the objectives according to the rank, from highest to lowest. The optimization process might tackle the objectives separately in order of priority, or use different objectives in different stages of the metaheuristic based on priority.

- **Dominance Based Schemes** – In this scheme, metaheuristics tackling MOPs seek to discover as much of the Pareto Front as possible based on the

concepts of dominance and Pareto Optimality as discussed earlier. Metaheuristics using such a scheme would usually employ an archive to keep track of the non-dominated solutions discovered. The occupants of the archive at the end of the metaheuristic's run would then be considered as the Pareto set discovered.

3.3 Particle Swarm Optimization (PSO)

Eberhart and Kennedy first described the Particle Swarm Optimization (PSO) metaheuristic in 1995 [28, 47]. Tracing its inspiration to the behavior of a flock of birds in flight, PSO utilizes the emergent behavior of a swarm of entities, called particles, as they traverse a solution landscape in search of a global optimum. Each particle in the swarm continuously goes through a process of evaluation, comparison and imitation, and the emergent property of this behavior is that the swarm as a collective converges on areas in the search landscape of optimal quality, or as close as possible to them. Given a swarm of n particles, each particle maintains a position x over the solution landscape which represents a candidate solution to the problem being solved. The representation, or particle state, takes the form of a vector of values, where each values stands in for one component making up the candidate solution. The quality of a candidate solution, and hence a particle, is determined by a fitness function $f(x)$ that is problem dependent. With every time step t, a particle would update its position using the following equation:

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{9}$$

where v is velocity, the direction the particle is currently pursuing across the landscape and its speed. Velocity for the next time step is calculated as follows:

$$v_{ij}(t+1) = \alpha v_{ij}(t) + c_1 r_1(y_{ij}(t) - x_{ij}(t)) + c_2 r_2(\hat{y}_{ij}(t) - x_{ij}(t)) \tag{10}$$

where

- i: the current particle
- j: the current component within the vector

- α : Inertia or Bias. A diminishing function of a particle's previous velocity.

- $(y_{ij}(t) - x_{ij}(t))$: Cognitive Component. The difference between a particle's current candidate solution and its previous personal best, determined by $f(x)$. This is applied across every component of the state vectors.

- $(\hat{y}_t - x_t)$: Social Component. The difference between a particle's current candidate solution and its best performing neighbor, determined by $f(x)$. As with the cognitive component, this is also applied across every component of the state vectors.

- c_1 and c_2: Cognitive and Social coefficients respectively. This controls whether the swarm favors one component over the other in influence by assigning them weights.

- r_1 and r_2: random coefficients assigned to Cognitive and Social components. These add a stochastic element to the swarm's behavior.

Depending on the representation of the problem being solved, and hence the particle state or position, the velocity equation may be applied as stated, or an extra modifier step would be required. If the problem being solved is from a continuous domain, and thus the particle's state is represented as real values, then equation 10 is applied directly to update velocity. If, however, the problem being solved is from a binary domain, and thus the particle's state is represented using boolean values, then the next state of x is determined using a probabilistic function of $v_{ij}(t+1)$:

$$Pr(x_{ij}(t+1)) \rightarrow 1 : \frac{1}{1 + e^{-v_{ij}(t+1)}} \tag{11}$$

By following this simple dynamic, the particles start from being scattered across the search landscape, to converging on a global optima, or as close an approximation of it. The pseudocode basic PSO can be seen Algorithm 1.

As things are rarely this simple, there are some nuances in implementing PSO, and these are discussed next.

Algorithm 1 Basic PSO

1. initialize swarm S
2. **repeat**
3. **for** every particle x_i in S **do**
4. **if** $f(x_i) > personal\_best(x_i)$ **then**
5. $personal\_best(x_i) \leftarrow f(x_i)$
6. **end if**
7. **for** every component j in particle i **do**
8. $bias \leftarrow \alpha v_{ij}(t)$
9. $cognitive \leftarrow c_1 r_1(y_{ij}(t) - x_{ij}(t))$
10. $social \leftarrow c_2 r_2(\hat{y}_{ij}(t) - x_{ij}(t))$
11. $v_{ij}(t+1) \leftarrow bias + cognitive + social$
12. **if** $j \in R$ **then**
13. $x_{ij}(t+1) \leftarrow x_{ij} + v_{ij}(t+1)$
14. **else if** $j \in [0, 1]$ **then**
15. $Pr(x_{ij}(t+1)) \rightarrow 1 : sigmoid(v_{ij}(t+1))$
16. **end if**
17. **end for**
18. **end for**
19. **until** stopping criteria met
20. return fittest particle

3.3.1 Nuances of Implementing PSO

3.3.1.1 Neighborhoods

The social component of the velocity update equation relies on measuring the distance between a current particle's solution and that of its fittest neighbor, but how are neighbors selected? What defines a neighborhood? The very first implementations of PSO considered two simple neighborhood structures: the entirety of the swarm is considered as a single neighborhood or the l adjacent particles to the one at hand could be considered its neighborhood. The implementations using those two neighborhood models became known as *gbest* and *lbest* PSO respectively [48]. *gbest* and *lbest* are also known as star and ring neighborhoods based on the

topology the particle at hand forms with the rest of the swarm. *gbest* implementations favors exploitation over exploration, as information regarding the currently located regions of quality is disseminated quickly. This can lead to premature convergence or the swarm getting stuck on local optima. On the other hand, *lbest*, with a relatively low valued l, spreads information regarding high quality regions of the solution landscape explored so far more slowly, and as such favors exploration over exploitation. This, however, can have the side effect that convergence can take longer than *gbest* implementations. A remedy for that would be to progressively increase the size of an *lbest* implementation's neighborhood, in order to favor exploitation in the later stages of the algorithm's runtime.

Besides *gbest* and *lbest*, other neighborhoods models have been devised. These include static neighborhood structures, such as the wheel [48], pyramid [30] and hyper-cube [1], and more sophisticated, dynamic structures that rely on fitness and spatial location [30]. Although some neighborhood structures have outperformed others when applied to certain problems, no one model has shown complete supremacy over the rest, and it falls to the implementer to fine tune the neighborhood structure used to best suit the problem being tackled. A discussion of the various neighborhood structures and their performance can be found in [30].

3.3.1.2 Particle Movement Bounds

Left to their own devices, particles in the swarm have been noticed to quickly move to the edges of the search space, and sometimes worse, beyond it. The particles would continue pushing the edges, so to speak, never converging. This is especially true when the particles have been initialized with significant initial velocities. In order to curb this behavior, and to promote convergence, the following measures were introduced over time:

- Initializing velocity to near-zero: By initializing velocity to zero or a very small value, particles are not liable to explosively shoot off in various directions in the beginning of the algorithm's runtime. This will prevent the particles from jumping to the edges of the search space within the first few iterations.

- Velocity Clamping: This measure imposes a limit to the value $v_{ij}(t+1)$ can

take by clamping it to be between V_{min} and V_{max}, which are usually defined as fractions of the domain of valid values x_{ij} can take. This has the effect of limiting the step a particle can take with every iteration. Though effectively preventing the swarm from seeking the edges of the search space and aiding in its convergence, using too tight a clamp can result in the particles getting stuck in wells of local optima. This risk can be mitigated by loosening the clamps on velocity when stagnation arising from this issue occurs.

- Decaying α: By relying less on previous velocity vectors, particles are more influenced by the cognitive and social components in calculating the velocity for their next position update. This is implemented as a decaying function weighing previous velocity, designated as α, and can be seen in equation 10 [82, 83]. This has the effect of promoting convergence as the algorithm proceeds, striking a good balance between exploration and exploitation, and preventing the particles from wandering beyond the viable search space.

- Clerk's Constriction: Devised as an alternative to velocity clamping and decaying α, Clerc devised a coefficient that is multiplied by the velocity update equation after removing α, r_1 and r_2. These parameters are instead used in the equation used to calculate the constriction coefficient. Under certain parameter constraints, PSO implementations that use Clerc's Constriction Coefficient are guaranteed to converge. More on Clerc's Constriction Coefficient can be found in [48].

3.3.1.3 Cognitive Versus Social

In order to establish the effect the cognitive and social components have on controlling a particle's behavior, and the swarm's in general, Kennedy studied the effect each component has by testing models that solely relied on one over the other [45]. Tuning how much each component influences velocity update can be achieved via the c_1 and c_2 coefficients. In the study, Kennedy observed that relying on the cognitive component alone resulted in swarms that essentially collapsed into multiple local search and displayed worse performance. On the other hand, relying on the social component alone resulted in swarms with a rapid convergence

profile. This was also confirmed by studies done by Carlisle and Dozier [19]. The effects of balancing influence between the cognitive and social components, and their effect on swarm behavior, is further discussed in [30].

3.3.1.4 Initialization, Stopping Criteria, Convergence and Stagnation

Sufficiently sampling the search landscape is essential to locating areas of optimality by a metaheuristic. This also holds true for PSO. A swarm's particles need to be evenly scattered across the solution landscape in the early stages of the algorithm's runtime to ensure that enough of the landscape has been sampled before the swarm starts to converge. This reduces the chances of the swarm getting stuck on a local optima and increases the likelihood that a global optima is found.

As for the issue of defining stopping criteria, the simplest method would be to set a quota of iterations that once met results in the algorithm stopping and the fittest particle returned. The quota of iterations needs not to be too low that results in the algorithm terminating prematurely before convergence is attained, or too high that needless computational complexity is added without commensurate improvements in quality. Another popular stopping criteria is convergence. This occurs when all particles within the swarm share the same state, in effect occupying the same space on the solution landscape. This is not to be confused with stagnation, which occurs when the particles show no improvement in solution quality over the last few iterations, and yet this quality is no where near the acceptable minimum levels. This can occur when swarms are attracted to wells of local optimality which they are unable to escape. While convergence is a valid exit for the algorithm, stagnation is not. In order to recover from a stagnation event, one could pursue one or more of the following strategies:

- Restart the algorithm by reinitializing the swarm. This rescatters the particles across the solution landscape, and as the algorithm progresses, it is probable that the swarm might avoid the area where it last stagnated due to the stochastic properties of the algorithm.

- Increase the size of the particle step. This can be done by either adopting a growing α or loosening the velocity clamps, as discussed earlier. Widening a particle's step increases its chances to escape the area of stagnation and puts

it on a trajectory towards other regions of the solution landscape. If enough particles escape the stagnation area to other areas of promising quality, then the rest of the swarm will follow via the mechanics of the velocity update equation.

- Upon detecting areas of stagnation, a utility function can mark this area as undesirable to the particles in the swarm by giving positions within that region reverse fitness, for example. This will result in the particles within the swarm to be repulsed by this region and seek other locations on the solution landscape.

- Move the current fittest particle in the swarm into a new random position within its current locus. This is the basis of a variation of PSO known as Guaranteed Convergence PSO (GCPSO), first described by Ven den Bergh in [89]. The idea behind this strategy is that by moving the current global best, then the mechanics of the velocity update equation would attract the rest of the swarm out of the area of stagnation.

The measures outlined above to recover from stagnation can be used on their own or in combination. If measures to recover from a stagnation fail repeatedly, then and only then, can we consider that the swarm has converged. In that case, the metaheuristic has found the closest approximation it can of the global optima. For a more detailed argument on convergence and stagnation, the reader is directed to the work of Ven der Bergh in [89] and its elaboration by Engelbrecht in [30].

3.3.2 Variations and Extensions

Having covered the dynamics of the basic PSO metaheuristic, we now turn our attention to extensions and variations developed over time in order to improve performance or address different classes of optimization problems. We first look into extensions and variations to improve the performance of single objective PSO. We then look into extensions to PSO that allow it to tackle niching problems. This is followed by looking into some hybrid approaches that fuse concepts from PSO with other metaheuristic paradigms. Finally, we close our discussion of extensions and variations by looking at approaches to tackling multiobjective optimization

problems. The variations and extensions discussed in the following subsections are not meant to be an exhaustive list of all developments in PSO since its inception. Instead they are meant to be a sampling to illuminate how PSO can (and has been) adapted to tackle a variety of problem types.

3.3.2.1 Single Objective Optimization

The basic PSO model discussed in the previous section is designed to tackle single optimization problems. Though some extensions and variations were mentioned, they mainly addressed shortcomings in the original algorithm design, controlled particle behavior and promoted convergence. The extensions and variations discussed in the next few paragraphs are more aimed at improving the performance of the PSO metaheuristic by returning better quality solutions.

Fully-Informed PSO (FIPS) was introduced by Kennedy *et al.* in [46, 60]. Instead of only being influenced by one neighbor in the velocity update equation, particles in FIPS are influenced by all of their neighbors. Particles in FIPS could either be influenced equally by all of their neighbors, or the influence can be weighted by the fitness of each neighbor. Kennedy *et al.* observed that both modes of operation outperformed the basic PSO model, with the latter model, where influence is weighted by fitness, performing better than the former.

In an attempt to strike a better balance between exploitation and exploration, Blackwell and Bentley imbued the particles in the swarm with a charge, analogous to electrostatic charges [11] [59]. Naming their approach Charged PSO, the idea was that particles carrying a charge would repel each other, yet the entire swarm would be attracted to its center of mass. The repulsion between the particles would aid in exploration, while the attraction towards the swarm's center of mass would promote exploitation and convergence. The repulsive force is encoded as an extra component in the velocity update equation. Blackwell and Bentley proposed three models within their approach: a neutral swarm, where non of the particles carried a charge; a charged swarm where all of the particles in the swarm carried a charge and an atomic swarm, where only half of the particles in the swarm carried a charge. A neutral swarm is essentially a regular particle swarm, with behavior identical to the one described in the previous section. Among the three models,

atomic swarms performed better than both charged and neutral swarms. The explanation behind that is that although explorative of the solution landscape, a fully charged swarm will have difficulty converging as the particles would continue to repel each other. An atomic swarm on the other hand will have the non-charged particles fall to the center of the swarm, on the area of optimality, while the charged particles continue to roam the regions on the periphery. This particular property also makes the Charged PSO model suitable for dynamic optimization problems, where the region of optimality shifts over time. A swarm with particles continuously roaming the landscape, as in Charged PSO, would be faster to pick up on such a shift and respond by migrating the center of the swarm's mass.

Another interesting class of extensions to PSO is one that uses sub-swarming. Instead of using one single, contiguous body of particles, sub-swarming utilized multiple collections of particles, each collection considered a smaller swarm forming part of the whole. These collections, or sub-swarms, could then work together, cooperatively towards locating a global optima, or they could compete. An example of a cooperative, sub-swarming technique is the Multi-phase PSO introduced by Al-Kazemi and Mohan in [3]. In Multi-Phase PSO, the swarm is divided into two equally sized sub-swarms, with the particles being randomly assigned to one of the two swarms. The sub-swarms then alternate between two phases of operation: attraction and repulsion. During the attraction phase, particles in a sub-swarm will be attracted towards the global best solution found by both sub-swarms. On the other hand, during repulsion, particles move away from the global best. Sharing the global best amongst the sub-swarms is the mechanism that allows cooperation. Controlling the particles' behavior during the phases is done by manipulating the cognitive and social coefficients, c_1 and c_2 respectively. During attraction, $c_1 = 1$ and $c_2 = -1$, while in repulsion the values are reversed. Multi-Phase PSO particles also do not rely on memory of a personal best position in their cognitive component. Instead, they only move when v_{t+1} takes them to a better place on the solution landscape. Multi-Phase PSO has shown better performance in training feed forward neural networks when compared with backpropagation and basic PSO.

3.3.2.2 Niching

Niching optimization problems are defined as those with multiple regions of optimality in their solution landscapes, and it is a requirement to locate as many of them as possible. Since basic PSO is capable of locating global optima, then it is feasible that by running it numerous times it would be able to locate all of the optima dotted across the solution landscape. There is no guarantee, however, that the swarm will not continuously converge on a previously discovered optimum. In order to avert this behavior, the fitness function can be modified after every optimum discovered to penalize the locus of that optimum on the solution landscape. A similar approach is used by Kassabalidis *et al.* [43] in inverting neural networks after training with success.

Another approach for adapting PSO to niching problems is that of Parsopoulos *et al.* [70, 72]. Dubbed Deflating PSO, Parsopoulos *et al.* modify the fitness function so that all points on the solution landscape in the vicinity of a recently discovered optimum have a fitness equal to, or more than, that of points outside the well of optimality. This has the effect of removing that particular well of optimality without affecting the rest of the solution landscape. This process is repeated until all wells, and thus optima, are discovered and removed.

So far, the methods of adapting PSO to solve niching problems has been sequential in nature. Optima are located one after the other, with the algorithms resetting their search between discoveries, till the solution landscape is consumed. In contrast, NichePSO, developed by Brits *et al.* in [14, 15], is one of the first PSO-based approaches that operates in parallel. Particles in NichePSO starting their life belonging to a singular swarm. The swarm roams the solution landscape searching for optima. When one possible optimum is located, a group of particles breaks off from the main swarm to form a sub-swarm to further discover the recently discovered well of optimality while the rest of the swarm moves on. This continues until the main swarm is completely divided into sub-swarms, and the sub-swarms operate in parallel to find the optima within their respective wells. Convergence occurs when all the sub-swarms are no longer capable of finding better solutions within their assigned regions of the solution landscape.

3.3.2.3 Hybrid Approaches

Hybrid approaches, those combining PSO with aspects from other metaheuristic paradigms, were developed soon after the introduction of PSO. The aims of hybridization is to improve the performance of the basic PSO algorithm and strike a better balance between exploration and exploitation. In this subsection, we will discuss hybrids of PSO with Genetic Algorithms (GA), Genetic Programming (GP) and Estimation of Distribution Algorithms (EDA).

One of the first approaches to combine PSO with aspects from GA, and in fact one of the first PSO hybrids, is that of Angeline in [5]. In attempts to improve the accuracy of the solutions discovered, Angeline proposed to use a tactic similar to selection that occurs in GA algortihms. During the algorithms runtime, a random subset of particles from the swarm would be selected as a benchmark of performance. The rest of the particles in the swarm are then compared against this benchmark and ranked accordingly. The bottom half of the particles according to rank, i.e. particles that are performing poorly when compared to the benchmark, is replaced with the top half. The personal best positions of the replacing particles is maintained throughout this process. Angeline's adaptation showed that it had better local search capabilities when compared to the basic PSO algorithm. It also had the undesirable side effect of vastly reducing diversity with the swarm, as with every time this process was applied, the diversity in the swarm fell by 50%. In an attempt to remedy this side effect, Koay and Srinivasan [51] proposed to replace the bottom half of the particles with mutated copies of the top half. Koay and Srinvasan also added the further constraint that replacement will only occur if the replacing particle has a higher fitness than the one being replaced. Another approach that utilizes concepts from GA is Cheap-PSO by Clerc [23]. In Cheap-PSO, the particles are capable of adjusting their step size, spawning new particles or terminating their existence based on the current status of the swarm as a whole. Particles would spawn a new particle and add it to the swarm when it senses a stagnation in its immediate neighborhood and that not enough improvement in the quality of solutions being detected is being achieved. On the other hand, if enough improvement is being achieved from the immediate neighborhood, then the particle with the poorest performance is terminated. The step size is adjusted

according the current performance of the particle, with larger steps being taken when not enough improvement is being achieved and vice versa. Cheap-PSO also does not use the cognitive component in its velocity update model, and relies solely on bias and the social component.

Moraglio *et al* developed hybrids of PSO and GP culminating a framework coined Particle Swarm Programming (PSP) [64, 87]. The aim of PSP is to imbue PSO with the capacity of working with GP expression trees over combinatorial search spaces. In order to do so, Moraglio *et al.* introduced new cross over operators to be used instead of the regular cognitive and social components in the velocity and position update equations. The performance of PSP was compared to GP benchmarks and was found to be competitive.

Closing out our discussions of PSO hybrids is those that combine aspects of Estimation of Distribution Algorithms (EDA). EDA progressively samples solutions from the vicinity of the best ones in a population of candidates on the search landscape in order to arrive at a global optima. Zhou *et al.* use PSO to enhance the search process of EDA by allowing the swarm to learn the distribution function [95]. Zhou *et al* applied their approach to a benchmark of discreet optimization problems and found their approach to perform better than other PSO models for discreet optimization. Bengoetxea and Larrañaga also developed an EDO-PSO hybrid in [7]. Their approach relies on two sub-populations: an EDA populaiton and a PSO population. The EDA population is then subdivided into chunks, were each chunk applies the EDA on its own. The chunks are then considered as particles in a PSO swarm, and PSO is used to optimize the chunk centers of mass. The process then repeats until convergence is attained or the stopping criteria are fulfilled. The performance of Bengoetxea and Larrañaga's model was evaluated on a number of benchmarks against prototypical EDA models and the hybrid's performance has shown to be competitive in general and better on some of the benchmarks.

3.3.2.4 Multiobjective Optimization

The earliest approach to adapt PSO to tackle multiobjective optimization problems is the one by Moore and Chapman in 1999 [63]. This approach followed

a dominance based scheme to attaining a Pareto front and employed the use of archives at two levels. The first archive is maintained per particle in the swarm and is used to track the non-dominated solutions discovered by each particle. The second (global) archive is used to track globally non-dominated solutions discovered by the swarm and, at the end of the algorithm run, it is used to represent the Pareto set discovered. When updating the state of particles via the PSO velocity equation, the authors would use a randomly selected solution from a particle's individual archive to stand in for a personal best. As for the neighborhood best, the selection is based on random selection amongst all the individual archives maintained by the neighbors of a given particle.

The year 2002 saw a flurry of activity in applying PSO to multiobjective optimization problems. Hu and Eberhart proposed a lexicographical approach for PSO to multiobjective optimization problems in [39]. Here, the authors set out to solve a MOP that utilized two fitness metrics. The authors used the simpler fitness metric to define the neighbors in a dynamic neighborhood of any given particle, and use the more complex fitness metric to select the neighborhood best. Personal bests in this scenario are the latest non-dominated solutions discovered by the particles as they traverse the search landscape. As this approach is limited to only being capable of optimizing two objectives, the authors revisited this scheme in [91] and adopted a dominance based scheme. An archive was added to keep track of non-dominated solutions discovered during the run of the algorithm, while measures were taken to improve the exploration aspect of the algorithm and allow it reach regions of the search landscape that were unreachable by the earlier algorithm.

Within the same year, Pasopoulos and Vrahatis also presented another contribution to the domain in the form of the Vector Evaluated PSO in [71]. This was later extended in [52]. In this dominance based scheme, the authors would use two subswarms to optimize a multiobjective optimization problem consisting of two objectives. Each subswarm would specialize in optimizing a single objective. Particles within each subswarm would use the global best of the other subswarm as the neighborhood best in a co-evolutionary fashion, and this is how the authors proposed tackling the MOP. The obvious shortcoming of this scheme is that it is limited to only tackling MOPs that consist of two objectives. Coello Coello and

Lechga proposed Multiobjective PSO (MOPSO), which is a dominance based algorithm to tackle MOPs [17]. The main feature of MOPSO is that it maintained a truncated archive, where priority in admission is given to new non-dominated solutions that occupy less densely populated regions of the Pareto front discovered so far. Personal bests are defined as the latest non-dominated solutions discovered by a given particle so far, while neighborhood bests are probabilistically selected from sparsely populated regions of the Pareto front. In an attempt to reduce the costs of maintaining an unbounded archive in a dominance based scheme, Fieldsend and Singh proposed a novel data structure to represent the archive known as a "dominated tree" in [31]. This new data structure defines how a neighborhood best is selected based on a composite point from the tree and the closest individual in the search space. Personal bests are managed as mini-archives maintained per particle for its discovered non-dominated solutions from which a random selection is made.

In 2003, Zhang *et al* proposed a lexicographical based approach for tackling MOPs using PSO [54]. Here the authors maintained separate global and personal bests for every objective being optimized. Neighborhood bests are a synthetic average of all the objectives begin optimized. Personal bests could either be constructed in the same fashion or selected at random from the set of tracked personal bests across all objectives. Within the same year, Mostaghim and Teich attempted to improve the performance of MOPSO proposed by Coello Coello and Lechuga [65]. Their contribution can be summed up as the introduction of new measure to improve the selection of neighborhood bests that lead to a faster convergence and improved diversity in the Pareto front returned. Another PSO variant proposed in 2003 is that of Zhang and Huang [94]. The main contribution of the authors here is that selection of a neighborhood best is based on probabilistic choice amongst solutions maintained in the archive that considers their distance from the particle at hand, favoring closer solutions. Personal bests in this scenario are the last non-dominated solutions discovered by the particles throughout the run of the algorithm.

Yen and Lu proposed a dominance based approach to tackling MOPs in 2003 [92]. The authors in this paper divided the search space into a grid of cells, where

each cell is represented by a centroid (a calculated coordinate representing the central point of the grid). Particles are assigned to the cells they are closest to based on their distance from the centroid. The cells are ranked based on dominance and occupancy of particles, with densely populated cells that contain non-dominated solutions receiving higher ranks. Neighborhood bests are then selected from the top ranked cells, while personal bests are selected at random from the occupants of the cell to which a particle belongs.

An interesting aggregate based approach was proposed by Baumgartner *et al* in 2004 [88]. The authors divided the main swarm into a number of subswarms, where each subswarm represented a different configuration for the weights used in the weighted sum composing all the fitness metrics being optimized. Each subswarm is then allowed to traverse the search landscape as it sees fit, while the main swarm kept track of the global bests discovered by each subswarm.

In 2009, Abido proposed using a multiobjective PSO variant to tackle the environmental/economic dispatch problem [2]. Here, the author's aim was to minimize two competing metrics: fuel cost and emissions, whilst complying to a set of constraints. The author introduces three types of archives to the basic PSO algorithm: an archive maintained at the particle level to maintain the non-dominated solutions encountered by each particle during the search process, a global archive used in selecting neighborhood bests and external archive to produce a Pareto set at the end of the algorithm's run. The global archive is formed by merging all the particle archives and filtering out dominated solutions. To select the personal and neighborhood bests for velocity update, the distances between the members of the particle's archive and the global archive are measured, and the pair with the shortest distance are selected as the personal and neighborhood bests respectively. After performing state update, the new solution represented by the particle is considered for admittance to the particle and external archives based on the concept of dominance. All the aforementioned archives are bounded, and when the archives exceed a certain size limit, solutions in the archives are removed based on the results of a clustering procedure. The results of using this approach showed to be superior to other algorithms in terms of both quality and diversity of the solutions in the returned Pareto fronts.

In 2012, Mousa *et al* proposed a hybrid approach to tackling multiobjective

optimization problems using aspects of both PSO and Genetic Algorithms (GA) [66]. The authors here use a phased approach to arrive at a final Pareto front. In the first phase of algorithm, PSO and GA are used in alternating succession to explore the search space and locate non-dominated solutions. The PSO operates to a manner similar to [2] in selecting local and neighborhood bests during velocity update in that it considers the Euclidean distances between the constituents of a particle's non-dominated archive and the constituents of a global non-dominated archive, and then selecting the pair with shortest distance as the local and neighborhood bests respectively. After the PSO updates the positions of the particles, the swarm is then considered as the population of individuals for the GA algorithm. The GA algorithm aggregates the fitness metrics into a weighted sum, and evolves a second generation from the current individuals. After GA evolves the subsequent generation, the individuals are treated again as particles in a swarm for the PSO to operate on. This process is repeated, alternating between PSO and GA, until the stopping criteria are satisfied. The surviving non-dominated solutions are then refined by a local search metaheuristic in the second and final phase of the algorithm to produce the Pareto front.

More recent approaches include [40], [6], [61], [77], [69], [80] and [57]. In an attempt to improve the diversity of solutions in the Pareto front and prevent premature convergence, the author in [40] propose a multiobjective PSO with novel measures to address these challenges. Based on a dominance-based approach, the authors utilize a bounded external archive to maintain the non-dominated solutions discovered during the search. Each particle also maintains a personal archive of non-dominated solutions, from a which a personal best is selected to participate in velocity and state update. With each iteration, a subset of the members in the external archive is selected to form candidates from which a neighborhood best is selected to participate in velocity update. These candidates are selected based on density and potential metrics, where density estimates how close a given solution is to other solutions on the current Pareto front and potential estimates how close a given solution would be to the true Pareto front. The candidates in this subset are then scored based on entropy, and one is probabilistically chosen to represent the neighborhood best per particle. As for selecting a personal best, one is chosen from the personal archive that minimizes the hyperbox formed between

the current particle's state, its previous velocity, its neighborhood best and the current contents of its archive. With the influence of density, potential and entropy, the multiobjective PSO traverses the solution landscape and returns the contents of the external archive at the end of the search as the discovered Pareto front. Another unique feature about this approach is that it is capable of adjusting its intertia, cognitive and social biases dynamically based on the delta of entropy displayed by the positions of the particles in the swarm between every iteration.

In [6], the authors proposed a PSO algorithm to perform clustering in a multi-objective fashion. The swarm's goal is to minimize the distance between two data points within a cluster (cohesion) and maximize the number of clusters within the dataset (connectivity). A particle here represents a possible assignment of a data point to a cluster. Using a dominance-based approach, the authors maintain an archive per particle that keeps track of the last discovered non-dominated solution discovered by each particle. The neighborhood best is selected at random from an archive formed by the union of all the particle archives. An external archive is used to keep track of all the non-dominated solutions discovered by the swarm, and at the end of algorithm's run, is used to represent the discovered Pareto front.

In [61], authors proposed dubbed Vortex Multi-Objective Particle Swarm Optimization (MOVPSO). The main shortcoming identified by the authors in typical dominance-based approaches of PSO (mainly based on the model defined in [63]) is the lack of diversity in the Pareto front. To address this shortcoming, the proposed algorithm traverses the search space using two alternating behaviors: convergence and dispersion. During convergence, the positions of the particles in the swarm is evaluated, and the swarm is attracted to positions that are furthest away from the swarm's center of mass based on Euclidean distance. The trajectories taken during convergence are linear. After convergence, the particles undergo dispersion from that previous convergence point, in trajectories that are circular in motion, with an ever increasing radius. With the constant alternation between convergence and dispersion, the authors hoped that the algorithm would be better in discovering the search space and returning a Pareto front with a higher diversity than the typical approaches.

Using a dominance-based hybrid approach, Rahimi *et al* proposed a multi-objective PSO to improve the detection of communities within complex networks

[77]. In their approach, the authors incorporate concepts from Genetic Algorithms (GA) to assist in their goal to minimize two measures of community quality. During initialization, the particles in the swarm are set as their own personal bests. With each iteration, the particles update their personal bests, perform the velocity update and subsequent state updates and a final mutation operator as a refinement. To update their personal state, the current state of a particle is crossed over with its last personal best in a fashion similar to Genetic Algorithms (GA) to produce two offspring. The non-dominated solution in the offspring is considered the new personal best. In the case of a tie, the offspring are scored using a community quality based metric, and the state with the highest score is considered the new personal best. To get the global best, and thus the neighborhood best that is to be used in the velocity update of the swarm particles, first all the current states are added to a non-dominated archive, and then the particle with the highest score based on the metric that was used in the personal state updates is chosen as the global best. Having newly defined personal and global (neighborhood) bests, the particles in the swarm perform velocity and state updates. As a final refinement step in the iteration, each particle will undergo a mutation procedure where by it replaces a component of its state with a possible value from any of the neighboring particles. The new resultant state is compared with the particle's personal best in terms of dominance and if found dominant replaces the particle's current personal best. This process repeats until a preset number of iterations is exhausted, and the Pareto front returned is the non-dominated archive formed from the particles states at the end of the search.

In [69] the authors attempt to simplify the state update mechanism and improve diversity of solutions discovered with their Diversity Enhanced Multiobjective PSO (DEMPSO). In DEMPSO, the velocity update equation drops the cognitive component and relies solely on the bias and social components. A global archive is used to maintain all the non-dominated solutions discovered in DEMPSO. The neighborhood best for each particle is based on the member of the global archive that has the farthest cosine distance from its current position. The algorithm also balances exploration and exploitation by dynamically shifting between the two behaviors based on the current particles' velocities.

In [80] the authors address the shortcomings of the VEPSO model first introduced in [71] by introducing a new model called Multi-guide PSO (MGPSO). MGPSO uses multiple swarms, each optimizing a single objective from the set of objectives being optimized. Instead of relying on a neighborhood guide from an alternate swarm as in VEPSO, the MGPSO model keeps the neighborhood guide from within the same subswarm as the current particle and adds a third component to the velocity update function that represents a guide from an archive of all the non-dominated solutions discovered. The authors present a stability study on their newly introduced model within the same publication and compare its performance both to existing VEPSO variants and non-PSO multiobjective optimization algorithms including NSGA-II, MOEA/D and PESA-II. The results show that the MGPSO model is highly competitive with all the algorithms it was compared against, and in some circumstances supersedes them in terms of performance.

Finally, in [57], the authors tackle multiobjective problems with a relatively high number of objectives using a proposed PSO approach that promotes diversity in the Pareto front by utilizing multiple subswarms. Considered as a hybrid between dominance-based and lexicographical approaches, the proposed algorithm uses a set of subswarms in its search process, one for each objective function to be optimized. Each subswarm specializes in optimizing a single objective function. An external, bounded archive is used to keep track of all non-dominated solutions discovered across all the particles in all of the subswarms. When selecting a neighborhood best for velocity update, a particle considers a non-dominated solution from the archive based on only two objective functions. The first is the objective being optimized by the current subswarm. For the second, we consider the remaining objectives being optimized and normalize their values based on the solutions in the archive. We then compare the performance of the current particle against the normalized values of these objectives and select the objective where the current particle is doing the worst. The neighborhood best is then chosen out of the archive where it dominates the current particle based on these two objectives. Personal bests are considered to be the last best solution encountered in terms of the objective function being optimized by the particle's subswarm. This coevolutionary particle swarm based approached showed promising performance when compared against a number of other multiobjective optimization algorithms

while testing them using two standard testing suites.

A summary of the multiobjective particle swarm optimization techniques discussed in this chapter can be seen in Table 1.

Table 1: Publications of work done multiobjective optimization using PSO.

Citation	**Authors**	**Year**	**Approach**
[63]	J. Moore and R. Chapman	1999	Dominance-based
[71]	K. E. Parsopoulos and M. N. Varhatis	2002	Dominance-based
[39]	X. Hu and R.C. Eberhart	2002	Lexicographical
[17]	C.A. Coello Coello, E.H.N. Luna and A.H.N. Aguirre	2002	Dominance-based
[31]	J. E. Fieldsend and S. Singh	2002	Dominance-based
[91]	X. Hu, R.C. Eberhart and Y. Shi	2003	Dominance-based
[54]	L. Zhang, C. Liu, Z. Ma, M. Ma and Y. Liang	2003	Lexicographical
[65]	S. Mostaghim and J. Tiech	2003	Dominance-based
[94]	Y. Zhang and S. Huang	2003	Dominance-based
[92]	G.G. Yen and H. Lu	2003	Dominance-based
[52]	K. E. Parsopoulos, M. N. Varhatis and D.K. Tasoulis	2004	Dominance-based
[88]	U. Baumgartner, C. Magele and W. Renhart	2004	Aggregate
[2]	M.A. Abido	2009	Dominance-based
[66]	A.A. Mousa, M.A. El-Shorbagy and W.F. Abd-El-Wahed	2012	Dominance-based, Hybrid
[40]	W. Hu and G. G. Yen	2015	Dominance-based
[6]	G. Armano and M. R. Farmani	2016	Dominance-based
[61]	J. Meza, H. Espitia, C. Montenegro, E. Gimenez and R. Gonzalez-Crespo	2017	Dominance-based
[77]	S. Rahimi, A. Abdollahpouri and P. Moradi	2018	Dominance-based, Hybrid
[69]	A. Pan, L. Wang, W. Guo and Q. Wu	2018	Dominance-based
[80]	C. Scheepers, A.P. Engelbrecht and C.W. Cleghorn	2019	Dominance-based
[57]	X. Liu, Z. Zhan, Y. Gao, J. Zhang, S. Kwong and J. Zhang	2019	Dominance-Based, Lexicographical, Hybrid

Chapter 4

Related Work in Market Timing

"Examine the present and learn from the past to see how the future will unfold. Too often we just look at the present and base our actions solely on that."

–Shinjo Ito

In this chapter we look at the current state of the involvement of computational intelligence in the market, with a particular focus on market timing. We will briefly consider work done using the current incumbent metaheuristic (Genetic Algorithms) to tackle that issue, before covering the work done using Particle Swarm Optimization (PSO). We will then briefly cover other uses of PSO within the financial domain before concluding with a critique of the current research and identifying rooms for improvement.

4.1 Market Timing

As you may recall form Chapter 2, market timing is the issue of when to buy and sell a given security on a stock exchange. In two recent and comprehensive studies, Hu *et al.* [40] and Soler-Dominguez *et al.* [84] investigate the use of computational intelligence techniques in finance. While the study by Soler-Dominguez *et al.* was more holistic in its coverage, the study done by Hu *et al.* was focused on the use of computational intelligence in the discovery of trading strategies. Both studies considered a large number metaheuristics to belong under the computational

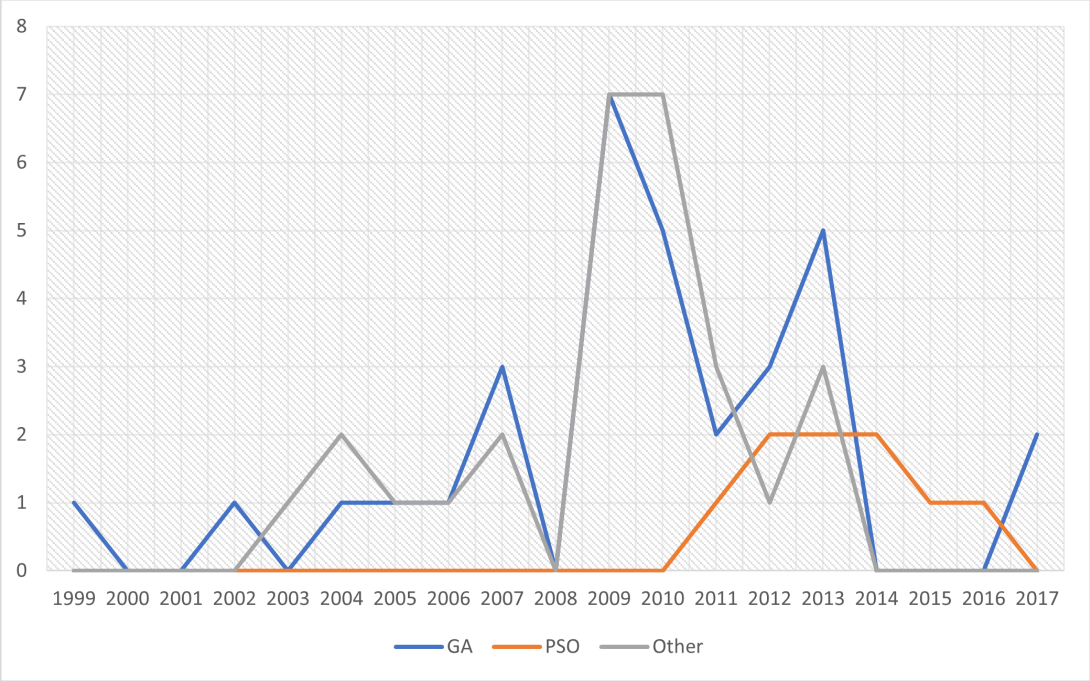

Figure 4: Publications by Algorithm and Year based on the [40] and [84] surveys of literature on the use of computational intelligence in finance. These surveys cover a time span between 1999 and 2017.

intelligence umbrella, and that included evolutionary algorithms (Genetic Algorithms, Genetic Programming, Differential Evolution), swarm intelligence (Particle Swarm Optimization, Ant Colony Optimization, Artificial Bee Colony Optimization), stochastic local search (Simulated Annealing, Iterated Local Search, Tabu Search, GRASP), fuzzy systems and neural networks amongst others. Both studies cover a combined time span starting with the early 1990's and ending with current times. Figure 4 shows a graph of publications by metaheuristic and year.

By surveying the techniques covered in both studies, we can see that generic algorithms (GA), and to a slightly lesser extent genetic programming (GP), are the most applied metaheuristics when it comes to the issue of market timing based on volume of publications alone. In the next two subsections we will look into applications of genetic approaches to market timing, with particular attention to Genetic Algorithms, and compare them to PSO based approaches to market timing. Since applications using GA are much more numerous compared to their PSO

counterparts, the related work for GA described next is meant to be a sampling of the different approaches taken in using GA to tackle market timing. For a more exhaustive look on the use of GA in market timing, we direct the reader to [40, 84]. As for PSO, we cover all the work done in relation to market timing up to and including 2017, where work on this thesis began.

4.1.0.1 Genetic Algorithms

One of the earliest work done based on GA was by Allen and Karjalanien in 1999 [4]. Here, the authors opted to use Genetic Programming (GP), an extension of GA, to discover both the structure and the parameters of possible trading rules that optimize a single financial fitness metric. The genetic structure of the individuals in the population represents possible trading rules modeled as trees. The roots of these trees are always Boolean functions that generate a buy signal when evaluating to true, and sell otherwise. The trees are built from a set of building blocks that include functions to calculate moving averages of past prices, functions that return the last trading prices, functions that return the maxima and minima of past prices, functions returning the absolute value of a difference between two numbers, various arithmetic operators (addition, subtraction, multiplication and division), logical operators (if-then-else, and, or, not), comparison operators ($<$, $>$) as well as various numerical and Boolean constants. Standard GP crossover and mutation operators are used to manipulate the individuals in the population in a continuous evolution scheme to evolve trading rules over the run of the algorithm. The performance of candidate solutions in the populace are evaluated against a reserved subset of the training data (which the authors call a "selection" period) and an elitist strategy is used to keep track of the best performing candidates. The training and testing procedure is a Step Forward one based on the data of the S&P500 index between January 1928 and December 1995 The metric optimized was excess returns in relation to a buy-and-hold strategy on the S&P500 index. The results show that the majority of the rules discovered had negative returns after accounting for transaction costs. The authors do however note that the trading rules discovered exhibit low volatility which might make them of interest to risk averse traders. The components available to the algorithm are rather simple,

and the authors suggest that one possible avenue of improvement is to provide the algorithm with more complex components to build the trees from including components based on fundamental analysis.

In [12], the authors use grammatical evolution to evolve trading rules for foreign exchange currency pairings. Related to GA from a genetic perspective, grammatical evolution (GE) relies on a linear representation of a genome that encodes the information required to generate rules from a Backus Naur Form (BNF) grammar. This is opposed to the tree syntax utilized by GP in evolving rules, as can be seen in [4]. The inspiration for GE comes from how proteins are synthesized from DNA genetic material. The grammar from which the trading rules can be evolved is composed of three technical indicators, along with standard arithmetic operators, binary operators, unary operators and the current price. The values generated by the rules are translated into sell, hold or buy based on predefined bands. An initial population of randomly generated individuals represents possible means of building trading rules using the grammar after decoding them from their nature as integer sequences. The algorithm proposed by the authors then repeatedly applies the mutation, crossover and duplication operators of GA to continuously improve upon the population and converge on a solution of optimal fitness. The authors tested their solution using Step Forward testing on data from three currency pairs and benchmarked the results against a buy-and-hold strategy. The results showed that the proposed system was superior to the benchmark in five out of six testing scenarios.

Another approach is to use GA to directly optimize the parameters of one or more financial analysis indicators, be they fundamental or technical in nature. Examples of such an approach can be seen in the work of Garrido *et al.* [24] and Subramanian *et al.* [85]. In [24], the authors encode the parameters of three technical indicators into a chromosome, and use GA to find the best parameter configuration that maximizes profit. In Subramanian *et al.* [85], the authors experimented with two approaches to arrive at market timing trading rules. Their first approach was to use a GA to optimize the weights of four technical indicators in order to maximize one of two financial metrics: either the Sharpe ratio or a modified version of the Sortino ratio. Their second approach was to use a GP to combine the aforementioned four indicators to form trading rules based on the

AND, OR and XOR Boolean operators. These approaches were evaluated in a trading simulation and results show that the GA approach yielded significantly better returns when compared to their GP counterpart. The authors also noted that the choice of financial metric used while training has a significant effect on the efficacy of the generated trading rules.

Other approaches since then use GA to improve the fitness of another primary metaheuristic in charge of producing the trading signals by optimizing its parameters. These primary, signal-producing metaheuristics included fuzzy systems, neural networks, self-organizing maps (SOM) and a variety of classification algorithms. One such example is [18]. In [18] the authors aim to improve he forecasting of price movements using a hybrid fuzzy time series (FTS) – GA algorithm. FTS was devised as a method to overcome the limitations of using traditional time series analysis methods via the incorporation of fuzzy mathematics. An important aspect for the performance of FTS is how the data is partitioned into intervals to formulate the universe of discourse. The authors used a GA model to optimize the intervals to be used in the universe of discourse for FTS by using an initial randomly generated population representing potential interval definitions, then using selection, crossover and mutation to continuously push the population to reach a convergence point. The fitness of candidate solutions from GA was based on the root mean square error of the forecasted series in relation to the actual data. The authors tested their hybrid FTS-GA system on data from TAIEX stock market and reported favorable results when compared to other FTS systems. A thorough breakdown of such synergistic approaches can be seen in [40].

More recent approaches for the use of GA to tackle market timing can be seen in the work of Kampouridis and Otero [41] and Kim *et al.* [50].

4.1.0.2 Particle Swarm Optimization

On the other hand, Particle Swarm Optimization (PSO) has not seen the popularity of GA and GP in the space of market timing. Though introduced much later than GA, PSO has started seeing some adoption in the space, and over the next few paragraphs we examine the work done by PSO to tackle market timing in chronological order.

The earliest PSO approach to tackle market timing was that of Briza and Naval, Jr. [16] in 2011. Inspired by [85], the authors optimized the weights of five technical indicators: the Directional Movement Index, Linear Regression, Moving Average Converge Diverge (MACD), Moving Averages and Parabolic Stop and Reverse. The values used for the parameters of each of the technical indicators involved was preset to industry standard values commonly used in literature. The authors approached market timing as a multiobjective optimization problem, and aimed to maximize two metrics of fitness: percent profit and the Sharpe Ratio. The signal used to decide when to buy or sell is then based on aggregating the individual signals produced by the individual indicators multiplied by their respective weights. If the aggregate value is positive and exceeds half the sum of the weights then a buy signal is generated, otherwise a sell signal is generated. Accrued profits from trades are not allowed to be used in reinvestment and transaction costs were not considered in the trading simulations. An individual particle in the swarm represents a candidate weight configuration for the five technical indicators used. The algorithm governing the interaction between the particles is based on MOPSO-CD [78], a multiobjective particle swarm that adopts the crowding distance metric from NSGA-II to maintain diversity in the Pareto set discovered. The authors used a Step Forward procedure for training and testing their model using daily data from the Dow Jones Industrial Average index (DJIA). To select the data for training and testing, the authors resorted to using three overlapping periods of the DJIA index with an ever increasing size in an attempt to capture varying market conditions, with the training and testing portions used to be of equal size. The authors then used these training and testing windows of data in experiments with various values for population size and iterations to be used in order to find the best values to use for the market timing system. The performance of their model was compared to that of NSGA-II and a Buy-and-hold strategy, and showed better performance than either of the benchmark strategies.

In 2012, PSO was used as a secondary metaheuristic to optimize the parameters of a primary signal generating one for the purposes of market timing. Chakravarty and Dash [20] approach market timing as a financial time series issue: if we are capable of forecasting movements in prices as a time series, then we can leverage that knowledge into arriving at decisions of when to buy or sell a given security

on the market. The authors note that statistical techniques of modeling, such as Autoregressive Moving Average (ARMA); Autoregressive Integrated Moving Avergae (ARIMA); Autoregressive Conditional Heteroscedasticity (ARCH) and Generalized Autoregressive Conditional Heteroscedasticity (GARCH), do not perform well unless supplied with relatively large datasets, which comes with a large computational cost. On the other hand, the authors note that computational intelligence techniques such as neural networks, fuzzy set theory, genetic algorithms and particle swarm optimization are better equipped in terms of performance to tackle forecasting time series of a such a chaotic nature as price data in the stock market. The authors in this paper compare the forecasting capacity of three hybrid neural models: Local Linear Wavelet Neural Network (LLWNN), Functional Link Artificial Neural Network (FLANN) and Functional Link and Interval Type 2 Fuzzy Neural System (FLIT2FNS). In order to train and optimize the weights used on the neural networks for the three models, the authors compared the performance of backpropagation versus Particle Swarm Optimization. The forecasting capability of the three models with either backpropagation or particle swarm optimization was then evaluated using data from the Standard's and Poor's 500 index (S&P500), Down Jones Industrial Average index (DJIA) and Bombay Stock Exchange (BSE). The experiments showed that Particle Swarm Optimization was the superior training algorithm, with the winning combination of model and training algorithm to be FLIT2NFS with PSO as shown by the experiment results. A similar approach can be seen in the work of Liu *et al.* [56]. Here, the authors used PSO to optimize the antecedent and consequent parameters in the rules of a type-2 neuro-fuzzy model. The model was evaluated using data from American and Taiwanese stock exchanges, and the accuracy of the proposed model shows to be favorable when compared with fuzzy time series variants based on root mean squared error.

The year 2013 brought us two attempts to use PSO for market timing. The first attempt, by Chen and Kao [21], uses PSO to optimize a system that relied on fuzzy time series and support vector machines (SVM) to forecast the prices of an index for the purposes of market timing. This approach showed an edge when compared against other contemporary methods for forecasting the index price that was used in that research. The second attempt, by Ladyzynski and Grzegorzewski

[55], uses a combination fuzzy logic and classification trees to identify price chart patterns, while PSO is used to optimize the parameters of the aforementioned hybrid approach. In their results, the authors have noted that use of PSO vastly improved the predictive capacity of the fuzzy logic and classification tree hybrid, and that the overall system proved to be promising.

In 2014, another two attempts were made to use PSO for market timing. Wang *et al.* [90] use a combination of a reward scheme and PSO to optimize the weights of a two technical indicators. The Sharpe ratio was used to measure the performance of the hybrid, and in their results, the authors note that their system outperformed other methods such as GARCH. Bera *et al* [8] used PSO to only optimize a single technical indicator. Although trading on the foreign exchange instead of the stock exchange, the authors note that their system has shown to be profitable in testing.

Finally, 2015 and 2016 showed only a single attempt per year. Sun and Gao [86] used PSO to optimize the weights on a neural network that predicted the prices of securities on an exchange. The authors note that their system was able to predict the price with an error rate of around 30% when compared to the actual prices. Karathanasopoulos, Dunis and Khalil [42] use PSO to optimize the weights on a radial basis function neural network (RBF-NN) that is capable of predicting the price of crude oil, a commodity. Though not on the stock exchange, the trading of commodities occurs on similar exchanges and uses many of the same market timing techniques. Compared to two classical neural network models, the authors note that their PSO-augmented approach significantly outperformed them in prediction capacity.

4.2 Other Uses of PSO

Besides market timing, PSO has also shown utilization in other areas within the financial domain. One of the most popular uses of PSO has been and still is portfolio optimization. A related problem to market timing, portfolio optimization is about finding the best combination of securities to build up trading portfolio in order to maximize returns while using a fixed trading strategy. Portfolio optimization remains to be one of the most popular uses of PSO in finance, where PSO has shown a strong competitive edge against other metaheuristics. Other uses of

PSO include option pricing, feature selection for classification in financial domain applications and credit risk optimization. For a good breakdown of the uses of PSO in those spaces, refer to the study by Soler-Dominguez *et al.* [84].

4.3 Critique

Our critique of the work done in market timing using PSO will be based on the following factors:

- Volume: The quantity of publications using the metaheuristic.
- Methodology: We discuss the categories of approaches used to utilize PSO in market timing.
- Limitations: We discuss the limitations identified by the authors of the publications themselves, as well limitation trends identified by the authors of computational intelligence studies mentioned earlier.

The aspects of critique are discussed in the next few subsections.

4.3.1 Volume of PSO Publications

The amount of publications that utilize PSO in one form of the other for the purposes of market timing is 9 publications, with the earliest being in 2011 and the latest in 2016 as of the time of when research in this thesis began. This largely pales in comparison with the market incumbent, GA, with 46 publications and ranging in time from the late 1990's to 2017 as of the time when research in this thesis began. One possible factor that can help us explain this massive difference is that genetic algorithms were introduced far earlier than particle swarm optimization, late 1970's versus mid 1990's respectively. Using PSO for other applications in finance is a different story however, with the most popular application being portfolio optimization. We can perhaps attribute this to the fact that the publications on PSO as a subject matter by its original authors [48] contained a lot of explanations and examples of the binary variant. This might have led to the

popularization of PSO as a combinatorial optimizer, leading to the natural application of it on portfolio optimization, a combinatorial optimization problem at its core.

4.3.2 PSO Methodology

From the 9 publications on the use of PSO in market timing, we can see a salient trend: PSO is used in a secondary role to optimize a primary metaheuristic or computational intelligence technique that is responsible for signal generation. The only three exemptions to that are the works of Briza and Naval, Jr. [16], Wang *et al.* [90] and Bera *et al* [8]. In these three publications, the authors used PSO as the only metaheuristic.

We also note that Step Forward backtesting is the only backtesting method used, and that buy-and-hold is the most common benchmark. No publication on market timing, whether involving PSO or otherwise, uses the style of backtesting recommended by Kaufman [44] and other market professionals. It was also noted by Hu *et al.* in their study [40] that a small percentage of all the publications surveyed (including GA) only compare the performance of their algorithm against a simpler version of the same metaheuristic, versus comparing its performance to another kind of metaheuristic. And while we are on the topic of testing, we also note that no PSO publication tests their algorithm in a live trading environment.

4.3.3 Limitations of Previous PSO Works

In the study by Hu *et al.* [40], the authors observe that most of publications displayed a performance that favored only one type of price trend, with the majority showing good performance in downtrends and suffering in uptrends. In both studies [40, 84], the authors find that the number of signal generating components used was kept rather low with rarely any publication using more than five unique components. It was also found that the majority of publications used technical analysis, with only two publications using fundamental analysis and four using a components from both schools. None of the PSO publications found used fundamental analysis. The authors speculate that perhaps using a more balanced selection of technical and fundamental components could lead to a more steady

performance across the various price trends. The authors of both studies also note that publication authors underestimate the impact of transaction cost on performance. Transaction cost in this context is used to refer to transaction fees, commissions, slippage and the availability of liquidity. This underestimation manifests itself as vague estimations of transaction fees, and no explicit account of the other contributing factors mentioned. As transaction costs can have a sizable impact on performance, turning a winning strategy in the lab to a losing one on the market, the authors of the studies lament that not enough attention is paid to transaction costs and its impact.

Chapter 5

Using Particle Swarms to Compose Strategies for Market Timing

"If we wait for the moment when everything, absolutely everything is ready, we shall never begin."

–Ivan Turgenev

In this chapter, we will introduce our first PSO algorithm to tackle market timing using a formulation that considers both the selection of trading signal generation components and the tuning of their parameters in a simultaneous fashion. We approach market timing as a single objective optimization problem, and the work in this chapter serves as a proof of concept: can we use PSO to tackle market timing in the aforementioned formulation? We begin by proposing an encoding strategy that enables this formulation, then proceed in describing the necessary modifications to basic PSO to use this encoding and optimize a single financial metric. We also describe two novel PSO algorithms that are meant to address limitations of the current PSO in exploring the search landscape and returning the least sufficing subsets to meet a particular objective. This is then followed by a description of the experimental setup used to test these PSO algorithms and a discussion of the results obtained and their implications.

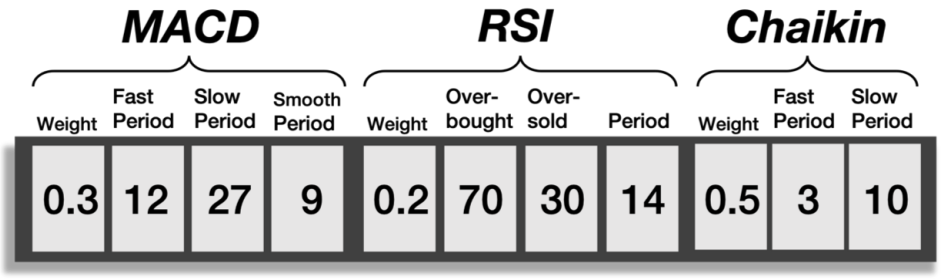

Figure 5: An example of a strategy encoded as an associative array. This example has instances of three technical indicators: MACD, RSI and the Chaikin oscillator. Each indicator has a weight and a set of parameters associated with it. For more on how the technical indicators generate signals, please refer to [76], [67] and [44]

5.1 Encoding Strategy

The first step in adapting a metaheuristic to tackle a particular problem is to find a suitable encoding for its candidate solutions. A candidate market timing strategy, and hence a candidate solution, is composed of one or more signal generating components, where each component has a weight and a set of parameters associated with it. Since components used in this thesis are of the technical analysis variety, all components encoded here represent instances of technical analysis indicators. We chose to encode our candidate strategies as associative arrays, sometimes implemented as dictionaries in particular programming languages. The first associate level identifies a certain indicator type, while the second identifies the weight and parameters associated with a given instance of that indicator. An example of this can be seen in Figure 5. In this example, our candidate strategy is composed of three technical indicators: the Moving Average Converge Diverge (MACD), the Relative Strength Indicator (RSI) and the Chaikin oscillator. Each of these indicators has a weight associated with it, and a set of unique parameters. This encoding strategy is used by our PSO algorithms in the representation of particles.

5.2 PSO to Tackle Market Timing

In order to adapt the basic PSO algorithm, presented in Chapter 3 (Algorithm 1), to tackle market timing in such a manner that considers both the selection

of components and the tuning of their parameters in a simultaneous fashion, we performed a number of modifications in its implementation. In order to be agnostic to the types of signal generating components and their parameters, the first modification was to push down the implementation of the addition, subtraction and multiplication operators required by the velocity update mechanism (lines 8 – 15) to be at the component level and not the metaheuristic level. This would no longer limit the parameter types to either be in the binary or real domains. A trader is now free to include a signal generating component of an arbitrary number of parameters and parameter types, as long as the implementation of that component overrides the necessary operators. Secondly, we also adopted a number of measures to promote convergence within the swarm and prevent velocity explosion. The first of these measures is that we used a decreasing inertia schedule. This means that for every step of the algorithm, inertia for every particle decreases by an amount defined by a function based on the number of iterations left. The second measure we adopted was that we scaled down $v_{ij}(t+1)$ before updating a particle's state by a user defined factor. As an alternative to velocity scaling, we also considered Clerc's Constriction as defined in [22]. The PSO algorithm would now be able to optimize both the parameters and weights of these components in relation to a financial fitness metric. The pseudocode for the adapted PSO can be seen in Algorithm 2.

The PSO algorithm described so far might not provide the best balance between exploration and exploitation. The presence of a local optima in the vicinity of the swarm can quickly pull the swarm's center of mass towards it. This is especially evident when the PSO algorithm uses a network structure that allows for the broad dissemination of information regarding current best discovered solutions, as is the case with the star or g-best topology. This can lead to the swarm of particles converging on suboptimal regions within the search space. Multiple measures have been devised since the introduction of the basic model to remedy that problem with varying degrees of success. We have discussed some of the measures in Chapter 3, and for further insight on the breadth of these measures, the reader is directed to [30]. Furthermore, the current PSO algorithm cannot grow or shrink the size of its candidate solution. This limits the market timing strategy to a preset number of signal generating components that form up the solution. This

can lead to a scenario where some of the components in a candidate solution being a load by not contributing significantly to the overall fitness yet still incurring a computational cost. In an attempt to address these two shortcomings, we introduce two modifications aiming to strike a better balance between exploration and exploitation while progressively shrinking the solutions represented by the swarm until a least sufficing subset of components is attained. We have dubbed these new algorithms as PSO with Stochastic State Update (PSO^{S}) and PSO^{S} with State Reduction (PSO^{SR}) [1].

Every particle x in the swarm S represents a candidate solution. From our discussion earlier, this means that a particle 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 algorithm, in PSO^{S}, the cognitive and social components of the velocity update equation are modified to be calculated as follows:

$$v_{t+1}(Cognitive) = \begin{cases} y_{ij}(t) - x_{ij}(t) & \text{, if rand()} < |\frac{f(y_{ij}(t))}{f(x_{ij}(t))+f(y_{ij}(t))}| \\ 0 & \text{, otherwise} \end{cases} \tag{12}$$

$$v_{t+1}(Social) = \begin{cases} \hat{y}_{ij}(t) - x_{ij}(t) & \text{, if rand()} < |\frac{f(\hat{y}_{ij}(t))}{f(x_{ij}(t))+f(\hat{y}_{ij}(t))}| \\ 0 & \text{, otherwise} \end{cases} \tag{13}$$

where:

- x: particle
- i: current particle index
- j: trading component index within current particle
- y: personal best
- $\hat{y}$: neighborhood best

[1] In [62], the newly introduced models were referred to as PSO-FInSSUP, with PSO^{S} being referred to as F- and PSO^{SR} referred to as F+

- $f(x)$: the fitness of x

That means the cognitive and social components will only stochastically influence velocity update if there is an improvement in fitness, in a hill climbing fashion. The PSO algorithm implementing the Stochastic State Update procedure is referred to as PSO^{S}.

The second modification we introduced to the basic algorithm is that with every preset number of iterations, we prune the least effective components based on their contribution to fitness. We start by aggregating the weighted fitness for every component across all particles in the swarm. This list of contributions by component is then normalized, and any components falling below a threshold Γ is removed from the swarm, resulting in a reduction in the size of the state represented by the particles. After a pruning event, the weights of the surviving components within the particles are renormalized between 0 and 1. The algorithm for this pruning procedure can be seen in Algorithm 5. The idea behind this procedure is that we want to shrink the solutions in search for the least sufficing set of components that maximizes our fitness metric, without largely impacting fitness. The PSO algorithm implementing both the Stochastic State Update procedure and this state reduction is referred to as PSO^{SR}. Pseudocode for both PSO^{S} and PSO^{SR} can be seen in Algorithm 3.

5.3 Experimental Setup

In order to measure the effectiveness of PSO in discovering market timing strategies, we tested the variations identified in Table 2. For variants using the ring neighborhood structure, 5% of the swarm lying on either side of the particle is considered to be within its neighborhood. PSO algorithms employing velocity clamping as part of their convergence mechanism use a factor of 0.5, a value deduced empirically. For all variants of PSO, we used a swarm size of 100 particles. Each experiment was allowed to run for a 100 iterations, and this was repeated 20 times to gather data for statistical significance tests. The metric optimized by all PSO variants was the Sharpe Ratio. We follow an elitist approach, and the algorithms return the best solution they discovered over their run.

Algorithm 2 PSO algorithm adapted to tackle market timing.

1. *archive*: an archive used to keep track of elite particles
2. S: swarm
3. N: swarm size
4. x: particle state
5. y: particle personal best
6. $calculate\_fitness$: a function used to perform backtesting using particle states and returns associated fitnesses
7. initialize swarm S
8. **repeat**
9. **for** every particle x_i in S **do**
10. **for** every component j in particle i **do**
11. $bias \leftarrow \alpha v_{ij}(t)$
12. $cognitive \leftarrow c_1 r_1(y_{ij}(t) - x_{ij}(t))$
13. $social \leftarrow c_2 r_2(\hat{y}_{ij}(t) - x_{ij}(t))$
14. $v_{ij}(t+1) \leftarrow cognitive + social$
15. **if** $clamp = Clerc$ **then**
16. $v_{ij}(t+1) = v_{ij}(t+1) \times clerc\_coefficient$
17. **else if** $clamp = Factor$ **then**
18. $v_{ij}(t+1) \leftarrow v_{ij}(t+1) + bias$
19. $v_{ij}(t+1) = v_{ij}(t+1) \times velocity\_clamp\_factor$
20. **end if**
21. $v_{ij}(t) = v_{ij}(t+1)$
22. $x_{ij} = x_{ij} + v_{ij}(t)$
23. **end for**
24. **end for**
25. $f(S) \leftarrow calculate\_fitness(S)$
26. **for** i: 1 to N **do**
27. **if** $f(x_i) \geq f(y_i)$ **then**
28. $y_i \leftarrow x_i$
29. **end if**
30. **end for**
31. $archive \leftarrow archive \cup$ fittest particle in S
32. **until** stopping criteria met
33. **return** fittest particle in *archive*

Algorithm 3 PSO^{S} and PSO^{SR} algorithms. If Pruning is enabled, then we have a PSO^{SR} algorithm. Otherwise, it is considered a PSO^{S} algorithm. Pruning is a user controlled parameter.

1. *archive*: an archive used to keep track of elite particles
2. S: swarm
3. N: swarm size
4. x: particle state
5. y: particle personal best
6. $calculate\_fitness$: a function used to perform backtesting using particle states and returns associated fitnesses
7. initialize swarm S
8. **repeat**
9. **for** every particle x_i in S **do**
10. **for** every component j in particle i **do**
11. $bias \leftarrow \alpha v_{ij}(t)$
12. **if** $random() < \frac{f(y_{ij}(t))}{f(x_{ij}(t))+f(y_{ij}(t))}$ **then**
13. $cognitive \leftarrow c_1 r_1(y_{ij}(t) - x_{ij}(t))$
14. **else**
15. $cognitive \leftarrow 0$
16. **end if**
17. **if** $random() < \frac{f(\hat{y}_{ij}(t))}{f(x_{ij}(t))+f(\hat{y}_{ij}(t))}$ **then**
18. $social \leftarrow c_2 r_2(\hat{y}_{ij}(t) - x_{ij}(t))$
19. **else**
20. $social \leftarrow 0$
21. **end if**
22. $v_{ij}(t+1) \leftarrow cognitive + social$
23. **if** $clamp = Clerc$ **then**
24. $v_{ij}(t+1) = v_{ij}(t+1) \times clerc\_coefficient$
25. **else if** $clamp = Factor$ **then**
26. $v_{ij}(t+1) \leftarrow v_{ij}(t+1) + bias$
27. $v_{ij}(t+1) = v_{ij}(t+1) \times velocity\_clamp\_factor$
28. **end if**
29. $v_{ij}(t) = v_{ij}(t+1)$
30. $x_{ij} = x_{ij} + v_{ij}(t)$
31. **end for**
32. **end for**

Algorithm 4 PSO^{S} and PSO^{SR} algorithms continued.

```
33.     f(S) ← calculate_fitness(S)
34.     for i: 1 to N do
35.         if f(i) ≥ f(y_i) then
36.             y_i ← x_i
37.         end if
38.     end for
39.     archive ← archive∪ fittest particle in S
40.     if Pruning is Enabled then
41.         if current iteration meets pruning deadline then
42.             prune()                          ▷ Refer to Algorithm 5
43.         end if
44.     end if
45. until stopping criteria met
46. return fittest particle in archive
```

Algorithm 5 Pruning Procedure for PSO^{SR}

```
1. for every particle x_i in S do
2.     for every component j in x_i do
3.         C_j ← C_j + w_ij(t) f(x_ij(t))
4.     end for
5. end for
6. normalize(C)
7. for every contribution c_j in C do
8.     if c_j < Γ then
9.         remove component j from all x in S
10.    end if
11. end for
```

The daily trading data of four assets from the US stock markets was used for both training and testing. The assets used were: Microsoft Corporation (MSFT), British Petroleum p.l.c (BP), Tesla Incorporated (TSLA) and Alphabet Incorporated (GOOG, formerly Google). The portion used for training is daily prices

Table 2: PSO variants used in the experiments.

Shorthand	Algorithm	Neighborhood Structure	Convergence Mechanisms
LB-V	PSO	l-best	Velocity Clamping, Decreasing Inertia
LB-C	PSO	l-best	Clerc's Constriction
GB	PSO	g-best	Velocity Clamping, Decreasing Inertia
PSO^S	PSO with Stochastic State Update	l-best	Velocity Clamping, Decreasing Inertia, Stochastic State Update
PSO^{SR}	PSO with Stochastic State Update and State Reduction	l-best	Velocity Clamping, Decreasing Inertia, Stochastic State Update

between the beginning of 2015 and the end of 2016, and the portion used for testing was daily prices for the year 2017 in a Step Forward fashion, as discussed in Section 2.2.

Six technical analysis indicators were used to form a gallery of components from which particles could build candidate solutions: Moving Average Converge Diverge (MACD), Aroon, Relative Strength Indicator (RSI), Stochastic Oscillator, Chaikin Oscillator and On Balance Volume (OBV). Where these components took parameters that affected periods of data to look at, a hard upper limit of 45 was set, so that we could get at least 5 trading signals with a single trading year (which is on average comprised of 252 trading days in the US market). Any other parameters were initialized to random values, and the best performing setting is discovered by the particles in the swarm as they traverse the solution landscape.

5.4 Results

Table 3 shows the mean Sharpe Ratio values attained by each PSO variant and asset combination on the testing data. Overall, we can see that algorithms that used ring-based networks fared better on average than star-based neighborhoods,

Table 3: Minimum, Mean and Maximum Sharpe Ratio values for each PSO variant calculated over 20 runs. We also show the Sharpe value for a Buy and Hold strategy applied per each asset's testing period. The highest value achieved per algorithm is highlighted in bold. The mean values observed per algorithm are considerably lower than the Buy and Hold values for the same testing periods, indicating a subpar performance.

Asset	Algorithm	Min	Mean	Max	Std
MSFT	LB-V	0.2043	0.6923	1.2371	0.3875
	LB-C	-0.5977	0.3485	0.6592	0.4592
	GB	**0.4451**	0.6907	1.0406	0.2949
	PSO^{S}	0.2195	0.6630	1.3042	0.3778
	PSO^{SR}	-0.5052	**0.7382**	**1.6543**	**0.5360**
	Buy & Hold	–	*2.36*	–	–
GOOG	LB-V	**0.5413**	**1.1565**	2.1562	0.6913
	LB-C	-0.3693	0.0999	1.1802	0.6582
	GB	0.1669	0.1669	0.1669	0.0000
	PSO^{S}	0.2326	1.0974	1.8891	0.6369
	PSO^{SR}	-0.6660	0.7329	**2.2556**	**0.7975**
	Buy & Hold	–	*2.00*	–	–
BP	LB-V	-0.7375	0.1915	1.1866	**0.6949**
	LB-C	-0.6892	**0.3967**	**1.3689**	0.6449
	GB	**-0.4452**	-0.0980	0.3909	0.2493
	PSO^{S}	-0.6802	-0.0605	1.1502	0.6063
	PSO^{SR}	-1.1122	-0.1977	1.2637	0.5649
	Buy & Hold	–	*1.10*	–	–
TSLA	LB-V	-0.3387	0.3374	**0.7313**	0.3591
	LB-C	**0.8516**	**0.8516**	0.8516	0.000
	GB	0.2190	0.5800	0.8697	0.2961
	PSO^{S}	-0.3599	0.2575	1.1583	**0.4643**
	PSO^{SR}	-0.5190	0.1568	0.6924	0.3103
	Buy & Hold	–	*1.24*	–	–

with GB ranking second to last according to a Friedman test using Holm's post-hoc correction (See Table 4). The fact that there is no statistical difference, as can be seen in Table 4, means that the results from the various PSO algorithms are competitive, suggesting the viability of the newly introduced PSO variants. From Table 3, we can also see that the standard deviation of the all PSO variants

tested are relatively large when compared to their means. This means that none of the algorithms tested so far was unable to identify specific regions of optimality within the search space, and instead solutions obtained when the stopping criteria were achieved are from various locations of the search space with a wide range of fitness values.

When considering the Sharpe Ratio values attained, we can see that the average Sharpe Ratio values rarely reached a value of 1 or more. Sharpe Ratio values of 1 and above are considered satisfactory, and values greater than three are considered excellent when the Sharpe Ratio is used to evaluate trading performance. The fact that the average values rarely achieved satisfactory levels suggests that the solutions discovered by PSO so far would be considered sub par. This is further exacerbated when comparing with a simple buy-and-hold strategy (as seen in Table 3), which represents a passive investor who bought the asset at the beginning of the testing period and held on to it. One can deduce from the Sharpe Ratio values of the buy-and-hold benchmarks that the trends witnessed during testing are either uptrends (MSFT and GOOG) or sideways (BP and TSLA). This highlights the need to be more prudent and subject algorithms to a more varied collection of testing periods that contain up, down and sideways trends to various degrees in order to get a more representative measure of how our approaches would perform in live trading. This will be addressed in the upcoming chapters.

In order to see if the pruning procedure in PSO^{SR} is effective in producing shorter solutions, we analyzed the results of the 80 experiments where this algorithm was involved and counted the number of the components in the solutions returned. Of the 80 returned solutions, only 18 solutions were affected by pruning reducing their length from 6 components to 5. The quality of those solutions was also affected, achieving only a maximum Sharpe ratio of 1.26 compared to a value of 2.26 where the solutions retained all 6 components. This can be seen in Figure 6, where we render the solutions returned by PSO^{SR} as a scatter plot showing fitness against solution length. Based on these observations, it seems that the pruning procedure in PSO^{SR} is ineffective in producing shorter solutions that are competitive to their unpruned counterparts in its current form.

From the results seen so far, we can see that representing the issue of market timing in a form that considers both the weight of the constituent components

and the value of its parameters and uses PSO to optimize their values is a feasible approach.

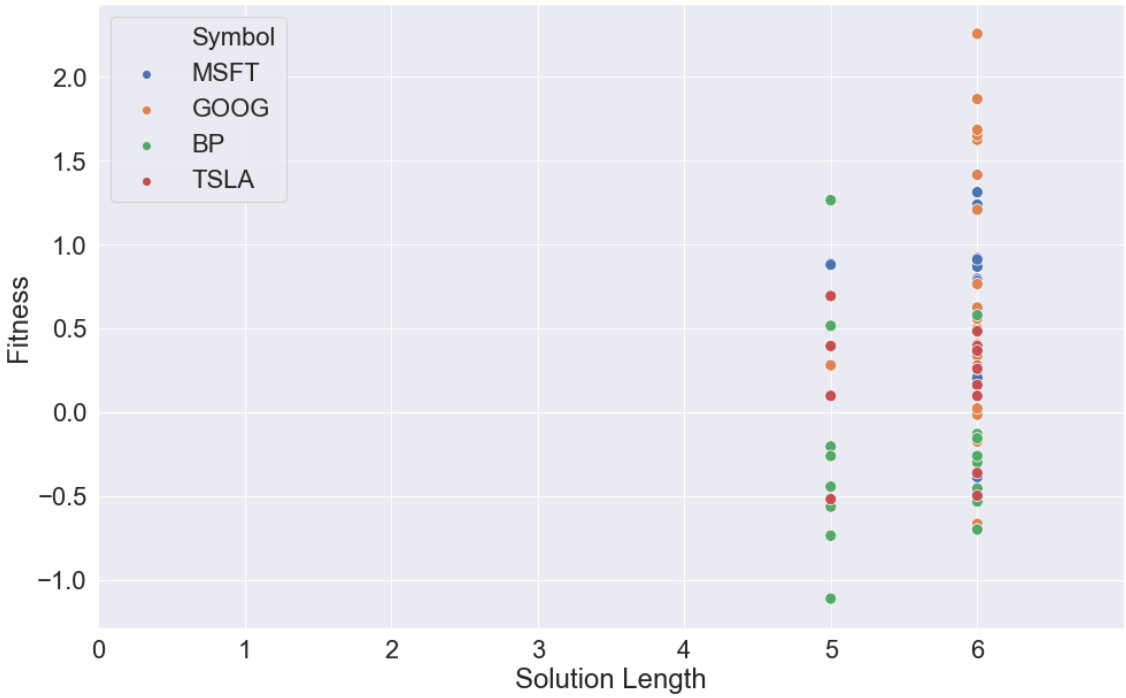

Figure 6: Scatter Plot showing Solution Length and Fitness achieved by PSO^{SR}. Out of 80 experiments, 62 solutions returned were not affected by the pruning, while the remaining 18 only lost one component. The 18 solutions of shorter length also show a reduced fitness when compared to the solutions that were not affected by pruning, indicating that the pruning procedure in PSO^{SR} is ineffective.

5.5 Summary

In this chapter, we presented a strategy to tackle market timing as a single objective optimization problem using PSO in a manner that considers both the selection of signal generating components (by way of adjusting their weights) and the tuning of their parameters simultaneously. We introduced two novel PSO algorithms: PSO^{S} and PSO^{SR}, and tested them along with classical PSO algorithms in optimizing the Sharpe Ratio using 6 technical indicators against data from 4 securities.

Table 4: Rankings of algorithms based on the Friedman non-parametric test with Holm post-hoc correction using mean fitness.

Algorithm	Ranking	p	Holm
LB-V	2.25	–	–
PSO^{SR}	2.75	0.6547	0.05
LB-C	3.0	0.5023	0.025
GB	3.25	0.3711	0.0167
PSO^{S}	3.75	0.1797	0.0125

Although the results showed that the two new algorithms were competitive to classical PSO, none of the algorithms was able to achieve good values for the Sharpe Ratio using the current experimental setup and Step Forward testing. Nevertheless, the work presented in this chapter proves that it is possible to consider the composition of market timing strategies in such a manner that addresses the selection of components and the tuning of their parameters in one fell swoop when compared to previous approaches that attempted to either select from a gallery of components with preset parameters or tune the parameters of a preset selection, but not both at the same time. We also showed that it is possible to do this using PSO, an algorithm that is not as extensively used within the domain of market timing as the current incumbent genetic algorithms. In the next chapters, we address the current limitations, improving the methodology of training and testing then move on to tackling market timing as a multiobjective optimization problem that involves a number of competing financial performance metrics.

Chapter 6

Trend Representative Testing

"You can only fight the way you practice."

–Miyamoto Musashi

As mentioned in Chapter 2, 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 [44, 40, 84]. Step Forward testing starts by acquiring a stream of price data for a particular tradable asset and then arbitrarily spliting this stream into two sections: the chronologically earlier section being used for training, while the later is used for testing. A common practice followed by users of this testing methodology is to ensure that the training section is proportionally twice the size of the testing section in terms of the 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 in terms of both 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, as illustrated in Chapter 2, Figure 2. This shortcoming has been reported in both the studies by Hu *et al.* [40] and Soler-Dominguez *et al.* [84], as well as in literature from the trading domain such as [44]. 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 this chapter we propose a novel training and testing methodology, called Trend Representative Testing, to address these shortcomings.

6.1 Trend Representative Testing

The primary 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 chance 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. This is based on the suggestions of domain experts in [44]. 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.

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 of 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 [1] 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

[1] Data retrieved from https://www.alphavantage.co/

Figure 7: A visual example[1] of the process behind generating a Trend Representative Testing dataset. The data shows 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 from Yahoo! Finance

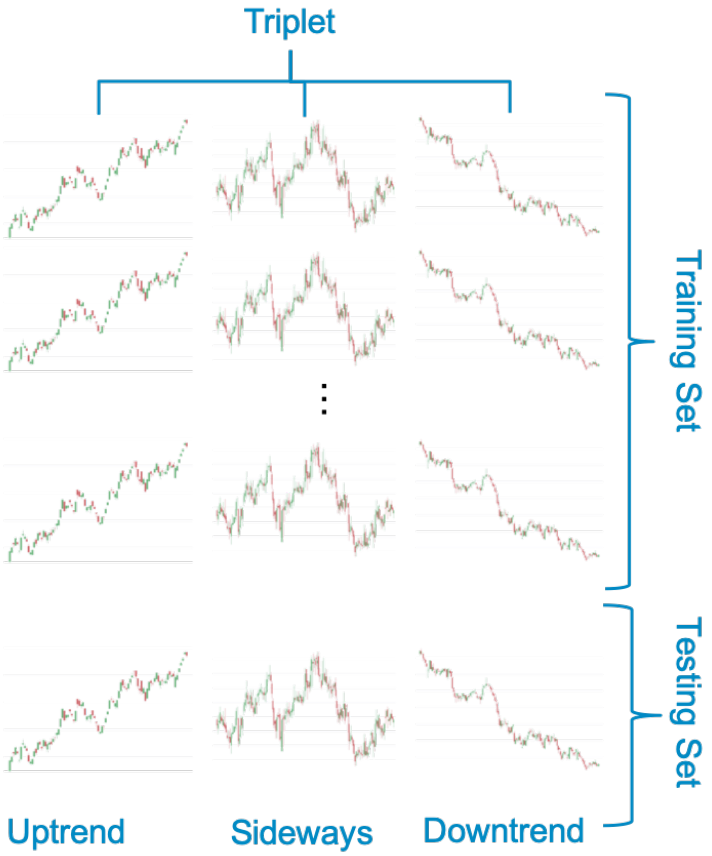

Figure 8: Using triplets of strands for training and testing in Trend Representative Testing. A triplet is composed of three strands: an uptrend, sideways movement and a downtrend. A set of triplets is divided into a training subset and testing subset. During training, an algorithm will pick a triplet at random from the training subset at each iteration, backtest a given candidate solution against each strand within the triplet and report the average performance. A similar process is followed during the testing phase.

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 problem that we are focusing on. 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 [44]. In the second step, 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 [76] and finally added to the library. A visual example of this process can be seen Figure 7.

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 randomly select a triplet from the training set with each iteration, and this triplet is used to assess the performance of all candidate solutions within the iteration. To measure the performance of a candidate solution, we evaluate the performance of the candidate solution at hand against each constituent strand within the triplet designated for the current iteration and report the average fitness. 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 solutions from the algorithm. An illustration of training and test sets can be seen in Figure 8. Pseudocode for how trend representative testing is used to calculate fitness can be seen in Algorithm 6. The idea behind trend representative testing is that we want to discourage niching or specializing in one particular trend type and instead promote discovering market timing strategies that fair well against various market conditions.

6.2 Computational Experiments

Having explored the motivation behind Trend Representative Testing, its methodology and how datasets are generated, we now move on to experimentation. Our first set of experiments compares Trend Representative Testing with the current incumbent method: Step Forward testing. This comparison used a limited set of technical indicators to see if using trend representative testing produces more resilient market timing strategies. Our second set of experiments is designed to see

Algorithm 6 Calculating Fitness using Trend Representative Testing

1. **function** $calculate\_fitness(population)$
2. U: Set of all strands representing upwards trends
3. S: Set of all strands representing sideways trends
4. D: Set of all strands representing downwards trends
5. $triplet \leftarrow U[random(1, length(U))] \cup S[random(1, length(S))] \cup D[random(1, length(D))]$
6. $fitnesses \leftarrow \{\}$
7. **for** $i = 1$ to $length(population)$ **do**
8. $fitness \leftarrow \{\}$
9. **for** *strand in triplet* **do**
10. $fitness \leftarrow fitness \cup backtest(strand, population[i])$
11. **end for**
12. $fitnesses \leftarrow fitnesses \cup mean(fitness)$
13. **end for**
14. **return** $fitnesses$
15. **end function**

if our PSO variants, presented in Chapter 5, are capable of producing competent market timing strategies using trend representative testing. They use a much extended set of technical indicators. To provide more context for the performance of the market timing strategies produced by our PSO variants, we benchmark them in both sets of experiments against those produced by a Genetic Algorithm (GA). The reason we chose GA as the benchmark algorithm is because it is the most used algorithm within the domain of market timing, as discussed in Chapter 4. The design of the GA benchmark is discussed next.

6.2.1 Genetic Algorithm Benchmark

In order to apply genetic algorithms (GA) to tackle market timing using our proposed formalization, we started with a typical implementation of GA and modified its operators in order to accommodate how we encode candidate solutions. First, individuals selected for crossover are chosen using a typical tournament procedure, with the tournament size being a user-defined parameter. A crossover point is then

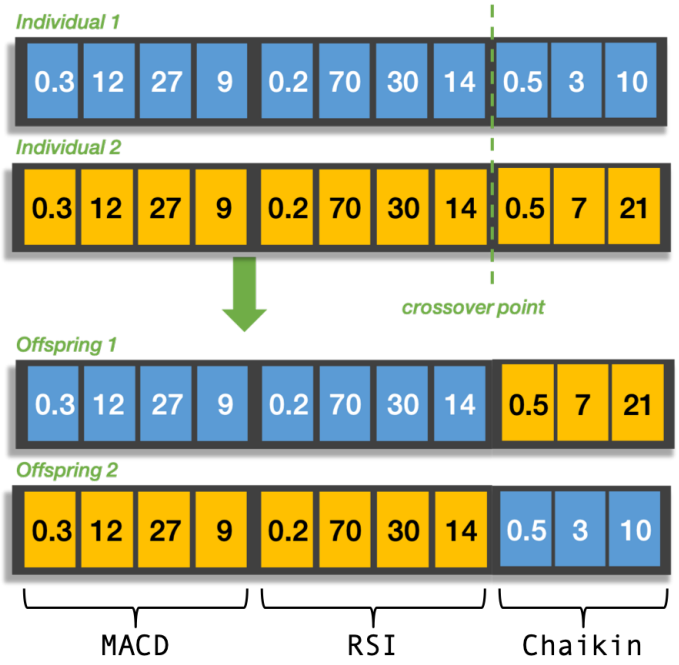

Figure 9: An example of a crossover operation.

selected at random such 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, parameter values are within valid ranges and the constraints on the parameters for each component are maintained. Using the example from Chapter 5, 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 9. When a mutation event is triggered, a random component in an individual's genotype is replaced by a newly instantiated copy of the same component type. This newly instantiated copy would use random values for the constituent parameters within the valid range.

Crossover and mutation are used to generate a new population for the next generation, and this procedure continues until the allocated budget of generations is exhausted. An archive is used to keep track of the elite individuals per generation, with the most fit individual in the archive reported at the end of the GA run as the proposed solution.

Algorithm 7 GA Benchmark Algorithm

1. N: population size
2. M: mutation probability
3. C: crossover probability
4. *population*: a population of candidate solutions
5. $fitnesses$: fitnesses associated with population at time t
6. $fittest()$: function that searches a given population and its associated fitnesses and returns the fittest individual
7. *archive*: an elitist archive used to keep track of fittest solutions per generation
8. $population_t \leftarrow$ populate with candidate solution of size N
9. $fitnesses_t \leftarrow calculate\_fitness(population_t)$ ▷ refer to Algorithm 6
10. **repeat** ▷ main loop
11. $archive \leftarrow archive \cup fittest(population_t, fitnesses_t)$
12. $population_{t+1} \leftarrow \{\}$
13. $i \leftarrow 0$
14. **repeat**
15. **if** $random() < M$ **then**
16. $selected \leftarrow tournament\_selection(population_t)$
17. $mutated \leftarrow mutate(selected)$
18. $population_{t+1} \leftarrow population_{t+1} \cup mutated$
19. $i \leftarrow i + 1$
20. **end if**
21. **if** $random() < C$ **then**
22. $p_1 \leftarrow tournament\_selection(population_t)$
23. $p_2 \leftarrow tournament\_selection(population_t)$
24. $population_{t+1} \leftarrow population_{t+1} \cup crossover(p_1, p_2)$
25. $i \leftarrow i + 1$
26. **end if**
27. **until** i = N
28. $population_{t+1} \leftarrow population_{t+1} \cup fittest(populaiton_t, fitnesses_t)$
29. $population_t \leftarrow population_{t+1}$
30. $fitnesses_t \leftarrow calculate\_fitness(population_t)$ ▷ refer to Algorithm 6
31. **until** stopping criteria met
32. **return** return fittest member in *archive*

6.2.2 Comparison With Step Forward Testing

The first set of experiments are designed to answer the question *Is Trend Representative Testing better than Step Forward testing in producing resilient market timing strategies?* The algorithms used in this set of experiments are GA, PSO, PSO^{S} and PSO^{SR}. All algorithms have access to 15 technical indicators – a list of the indicators can be found in Table 5. Since the algorithms being tested have parameters of their own, the first step in our experimental setup consists of performing hyperparameter optimization. As the pruning parameter is a tunable option, our novel PSO variants introduced in Chapter 5 are translated to either PSO^{S}, when the pruning is turned off, and PSO^{SR}, when pruning is turned on. The IRace algorithm [58] was used to perform hyperparameter optimization for the aforementioned algorithms for both Step Forward and Trend Representative testing, resulting in six distinct configurations. To generate data for IRace to train and test on, we start with the data for the Trend Representative testing method. We selected 30 strands at random from the strand repository created for the Trend Representative dataset. A strand also has a trend direction and an intensity associated with them. These 30 strands are selected such that we end up with 10 uptrends, 10 sideways and 10 downtrends. The 30 strands are then divided into 10 triplets, where each triplet contains a single uptrend, a single sideways movement and a downtrend. For training, the algorithm would use 9 triplets and hold out the last triplet for testing. This is repeated in a round-robin fashion until each triplet has played both a role in training and testing. For the Step Forward set of IRace runs, four distinct datasets were built: one consisting of all uptrends, one consisting of all downtrends, one consisting of all sideways trends and finally one consisting of random trends. The price streams selected for each of these datasets are divided into training and testing data such that training would be the earlier 70% of the strand, while testing would be the latter 30%. The parameters for each algorithm and the results of the IRace run can be seen in Table 6 and Table 7 respectively.

For the training and testing datasets used in experiments we performed a similar exercise as with the hyperparameter optimization datasets. We start with the

Table 5: Technical Indicators used in Step Forward versus Trend Representative Testing. All parameters are of type integer, with the exception of weight, which is of the type double. Information regarding how these indicators generate signals can be found in [67], [76] and [44].

Indicator	Parameters
Moving Average Converge Diverge (MACD)	Weight, Short Period, Long Period, Signal Period
Aroon Oscillator	Weight, Period, Lookback
Relative Strength Indicator	Weight, Overbought Threshold, Oversold Threshold, Period
Stochastic Oscillator	Weight, Fast K Period, Slow K Period, Slow D Period
Chaikin Oscillator	Weight, Fast Period, Slow Period
On Balance Volume (OBV)	Weight, Period
Mat Hold Pattern	Weight, Trend Period, Smooth Period
Rising Falling Three Methods Pattern	Weight, Trend Period, Smooth Period
Separating Lines Pattern	Weight, Trend Period, Smooth Period
Doji Star Pattern	Weight, Trend Period, Smooth Period
Engulfing Pattern	Weight, Trend Period, Smooth Period
Three Outside Pattern	Weight, Trend Period, Smooth Period
Three Black Crows Pattern	Weight, Trend Period, Smooth Period
Three White Soldiers	Weight, Trend Period, Smooth Period
Morning Star Pattern	Weight, Trend Period, Smooth Period

Trend Representative set, and obtain data in a similar fashion but with the stipulation that none of the selected strands were used with IRace. The 30 strands selected for experimentation are also divided into 10 triplets, were each triplet contains a single uptrend, a single sideways trend and a single downtrend. To obtain the data for the Step Forward training set, the 30 strands are considered as the testing section, and we retrieve the chronologically preceding data per strand to complete the training set. The amount of preceding data retrieved per strand is done in such a manner to maintain 70% training to 30% testing in the Step Forward scheme. This leaves us with 30 pairs for training and testing in a Step Forward fashion, and 10 triplets for training and testing in a Trend Representative fashion. The data used by the experiments can be seen in Table **??**. The trends

Table 6: IRace Parameters

Parameter	Algoritm	Range	Condition
Population Size	All	20 - 100	
Iterations	All	20 - 300	
Mutation Probability	GA	0 - 1	
Crossover Probability	GA	0 -1	
Tournament Size	GA	2 - 50	
Neighborhood Size	PSO, PSO^S, PSO^{SR}	2 - 100	
c1	PSO, PSO^S, PSO^{SR}	2 - 4	
c2	PSO, PSO^S, PSO^{SR}	2 - 4	
Clamp	PSO, PSO^S, PSO^{SR}	Clerc or Factor	
Factor	PSO, PSO^S, PSO^{SR}	0 - 1	Only if Clamp = Factor
Pruning	PSO^{SR}	True or False	
Pruning Threshold	PSO^S, PSO^{SR}	0 - 1	Only if Pruning = True
Pruning Deadline	PSO^S, PSO^{SR}	1 - n	Only if Pruning = True and $n \leq$ Iterations

of the Step Forward training datasets were discovered post-hoc, and the fact that they all turned out to be sideways is coincidental. Both testing strategies are performed by the four algorithms as configured by IRace and each experiment is repeated 10 times to cater for effects of stochasticity.

Figures 10 to 14 show the minimum, median, mean and maximum fitnesses achieved for each strand and algorithm, along with standard deviation. From looking at the heatmaps, we can see that the highest minimum fitness was achieved by Trend Representative PSO^{SR}. Median and mean fitnesses for the different algorithms show a clear advantage for Trend Representative PSO^{SR} with the AVNW2 (Refer to Table 8) test strand. Beside these two particular data points, median and mean fitnesses for all algorithms are close in value, and this is further corroborated by the Friedman test performed on mean fitnesses for the algorithms. Maximum fitnesses achieved by the algorithms display two particular hot spots where fitnesses are significantly larger than the rest of the data points. These two hot spots occur with the AVNW2 and AVNW6 test strands. With AVNW2, Trend Representative GA, Trend Representative PSO^{SR}, Step Forward (Random)

Table 7: Algorithmic configurations after IRace for Step Forward versus Trend Representative Testing experiments.

Parameter	Step Forward (Down)			Step Forward (Random)			Step Forward (Side)			Step Forward (Up)			Trend Representative		
	PSO^{SR}	GA	PSO	PSO^{S}	GA	PSO	PSO^{SR}	GA	PSO	PSO^{S}	GA	PSO	PSO^{SR}	GA	PSO
Population Size	65	51	58	45	43	81	31	37	87	70	79	74	45	29	32
Iterations	26	187	274	192	230	142	165	125	255	37	276	33	289	262	185
Mutation Probability	–	0.581	–	–	0.5668	–	–	0.9346	–	–	0.4035	–	–	0.4513	–
Crossover Probability	–	0.7525	–	–	0.0932	–	–	0.5246	–	–	0.701	–	–	0.9218	–
Tournament Size	–	30	–	–	33	–	–	17	–	–	36	–	–	5	–
Neighborhood Size	99	–	58	24	–	63	31	–	40	68	–	74	19	–	32
c1	2.6253	–	2.6319	3.2983	–	3.2853	2.5538	–	3.6963	2.7469	–	2.0227	3.2963	–	2.5893
c2	2.8787	–	3.3577	3.3614	–	3.0859	2.1425	–	2.3687	2.2217	–	2.7579	2.5195	–	2.2139
Clamp	Clerc	–	Clerc	Clerc	–	Clerc	Clerc	–	Clerc	Clerc	–	Factor	Clerc	–	Clerc
Factor	–	–	–	–	–	–	–	–	–	–	–	0.8281	–	–	–
Pruning	True	–	–	False	–	–	True	–	–	False	–	–	True	–	–
Pruning Threshold	0.1543	–	–	–	–	–	0.2222	–	–	–	–	–	0.3152	–	–
Pruning Deadline	117	–	–	–	–	–	85	–	–	–	–	–	285	–	–

Table 8: Experiment Training and Testing Data. The Id is formed by concatenating the asset symbol and a numerical identifier.

Id	Begin Date	End Date	Length	Trend	Is Training?	Is Testing?	Fold
ED2	9/8/14	1/14/15	90	↑	Yes	Yes	0
ETV2	3/18/13	7/24/13	90	↔	Yes	Yes	0
LUV2	7/31/08	1/21/09	120	↓	Yes	Yes	0
JBLU2	8/1/06	12/6/06	90	↑	Yes	Yes	1
LUV4	11/18/08	10/30/09	240	↔	Yes	Yes	1
LUV6	11/4/08	3/30/09	100	↓	Yes	Yes	1
COWN2	2/2/15	7/9/15	110	↑	Yes	Yes	2
JBLU4	4/17/08	6/15/10	545	↔	Yes	Yes	2
AVNW2	4/27/11	12/12/11	160	↓	Yes	Yes	2
KFY2	7/22/16	12/27/16	110	↑	Yes	Yes	3
JBLU6	9/12/08	5/3/11	665	↔	Yes	Yes	3
COWN4	8/1/11	12/28/11	105	↓	Yes	Yes	3
COMT2	1/26/16	6/9/16	95	↑	Yes	Yes	4
EXC2	2/24/14	2/24/16	505	↔	Yes	Yes	4
IAG2	1/31/12	6/7/12	90	↓	Yes	Yes	4
MGA2	3/4/13	8/14/13	115	↑	Yes	Yes	5
IAG4	1/26/11	7/25/11	125	↔	Yes	Yes	5
IAG6	4/18/16	8/30/16	95	↓	Yes	Yes	5
AVNW4	5/26/05	1/5/06	155	↑	Yes	Yes	6
ED4	8/5/16	12/27/16	100	↔	Yes	Yes	6
COWN6	7/7/14	11/10/14	90	↓	Yes	Yes	6
BSX2	9/7/12	6/26/13	200	↑	Yes	Yes	7
COWN8	5/3/12	1/14/13	175	↔	Yes	Yes	7
AVNW6	4/1/11	11/9/11	155	↓	Yes	Yes	7
MGA4	3/26/13	10/18/13	145	↑	Yes	Yes	8
JBLU8	1/11/11	11/7/12	460	↔	Yes	Yes	8
COWN10	7/15/11	12/5/11	100	↓	Yes	Yes	8
IAG8	11/18/15	4/6/16	95	↑	Yes	Yes	9
COWN12	11/15/10	10/5/11	225	↔	Yes	Yes	9
LUV8	7/9/08	11/19/08	95	↓	Yes	Yes	9
ED1	11/5/13	9/5/14	210	↔	Yes	No	NA
ETV1	5/14/12	3/15/13	210	↔	Yes	No	NA
LUV1	6/21/07	7/30/08	280	↔	Yes	No	NA
JBLU1	9/29/05	7/31/06	210	↔	Yes	No	NA
LUV3	8/29/06	11/17/08	560	↔	Yes	No	NA
LUV5	12/3/07	11/3/08	233	↔	Yes	No	NA
COWN1	1/27/14	1/30/15	256	↔	Yes	No	NA
JBLU3	3/31/03	4/16/08	1271	↔	Yes	No	NA
AVNW1	11/2/09	4/26/11	373	↔	Yes	No	NA
KFY1	7/17/15	7/21/16	256	↔	Yes	No	NA
JBLU5	7/17/02	9/11/08	1551	↔	Yes	No	NA
COWN3	8/11/10	7/29/11	245	↔	Yes	No	NA
COMT1	3/11/15	1/25/16	221	↔	Yes	No	NA
EXC1	6/18/09	2/21/14	1178	↔	Yes	No	NA
IAG1	3/31/11	1/30/12	210	↔	Yes	No	NA
MGA1	2/6/12	3/1/13	268	↔	Yes	No	NA
IAG3	11/30/09	1/25/11	291	↔	Yes	No	NA
IAG5	6/2/15	4/15/16	221	↔	Yes	No	NA
AVNW3	12/18/03	5/25/05	361	↔	Yes	No	NA
ED3	9/2/15	8/4/16	233	↔	Yes	No	NA
COWN5	9/4/13	7/3/14	210	↔	Yes	No	NA
BSX1	11/2/10	9/6/12	466	↔	Yes	No	NA
COWN7	9/21/10	5/2/12	408	↔	Yes	No	NA
AVNW5	10/26/09	3/31/11	361	↔	Yes	No	NA
MGA3	11/16/11	3/25/13	338	↔	Yes	No	NA
JBLU7	10/6/06	1/10/11	1073	↔	Yes	No	NA
COWN9	8/12/10	7/14/11	233	↔	Yes	No	NA
IAG7	1/5/15	11/17/15	221	↔	Yes	No	NA
COWN11	10/15/08	11/12/10	525	↔	Yes	No	NA
LUV7	8/22/07	7/8/08	221	↔	Yes	No	NA

GA and Trend Representative PSO show a clear advantage over the other algorithm variants in this order. In AVNW6, the Trend Representative variants show the clear advantage over Step Forward ones.

The results obtained from the experiments were also rendered in the form of box plots of fitnesses for the various algorithms and test strands, and can be seen in Appendix I. Figures 22 to 52 show box plots of fitnesses for the various algorithms and test strands. This helps us get a deeper look at the performance of the algorithms tested. The salient trend that can be seen from the box plots is that the algorithms show stability around a relatively narrow band of fitness, with one or two algorithms outperforming the remaining ones. This can be seen, for example, with BSX2, COWN2, EXC2, IAG4, JBLU4, LUV6, and MGA4. In some of these instances, Trend Representative algorithms are at a clear advantage, such as in the cases of AVNW2, AVNW6, COWN6, ED2, ED4, JBLU4, and JBLU6. In other instances, Trend Representative algorithms are performing worse than their Step Forward counterparts, such as in the cases of COWN12, COWN4, IAG6, KFY2, LUV2, LUV6 and MGA4. It can be observed that algorithms that employ Trend Representative testing have a higher tendency to produce a wider distribution of fitness values when compared to algorithms that employ Step Forward testing. This can be advantageous if a market timing strategy is to be built using maximum fitness and the designer of the strategy can afford the cost of running the algorithm numerous times. No clear correlation between the performance of a particular type of algorithm and the underlying trend type of a test strand was observed.

In order to see whether statistical significance has been attained or not, we conducted a Friedman non-parametric test with the Holm post-hoc correction[27, 33] on minimum, median, mean and maximum fitnesses. The results can be seen in Tables 9 to 12. No statistical differences at the significance level 5% level were observed to identify whether Trend Representative Testing has an edge over Step Forward Testing based on current experimental setup. Given that Step Forward and Trend Representative Testing perform equally from a statistical perspective, we opt to use Trend Representative Testing. The reasoning behind this is that Trend Representative testing explicitly exposes an algorithm to multiple trend types during training and testing, and that allows us to have an estimation of its performance when encountering unforeseen trends during deployment.

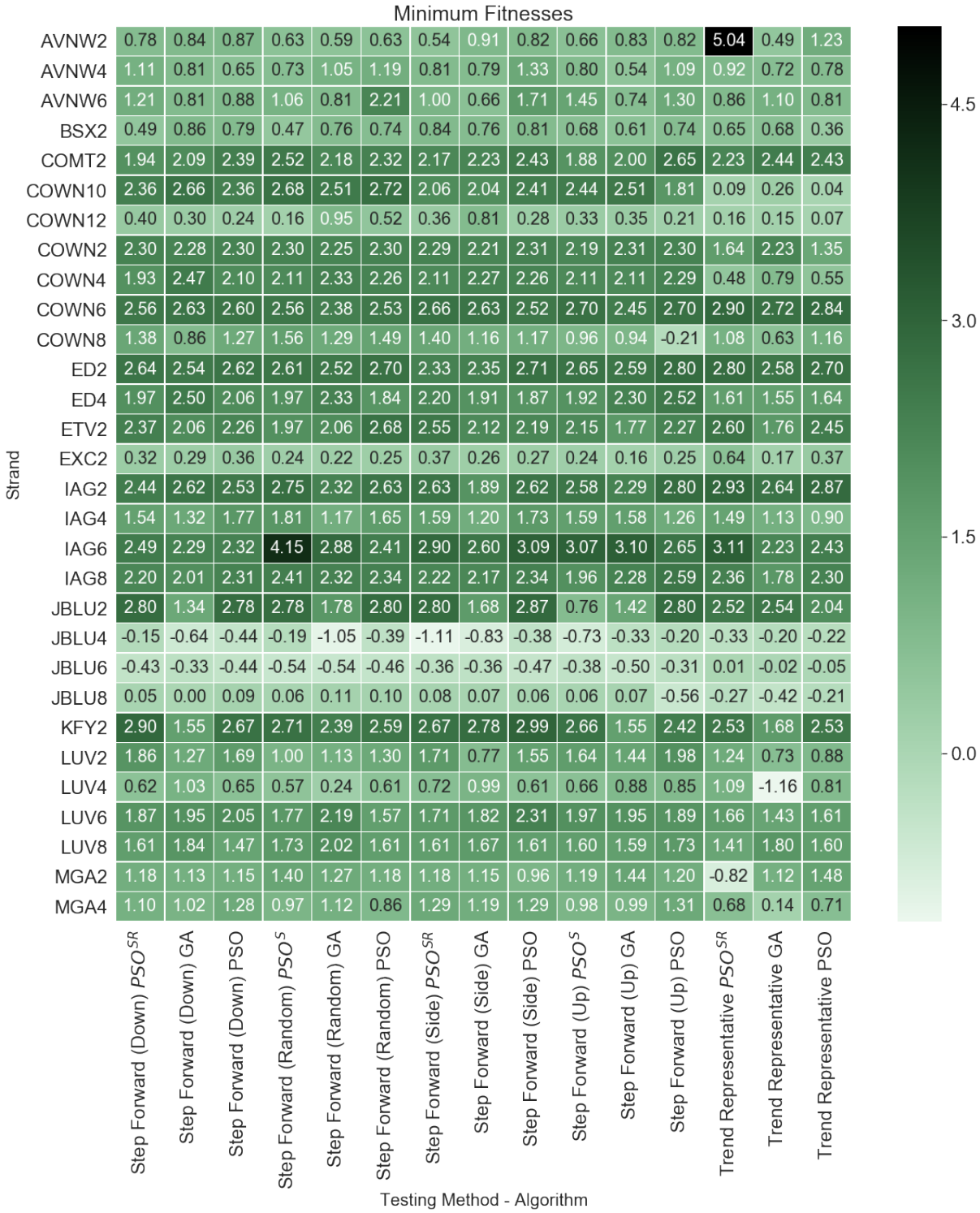

Figure 10: Minimum Fitnesses Heat Map. Darker colors indicate a higher fitness and thus better solutions.

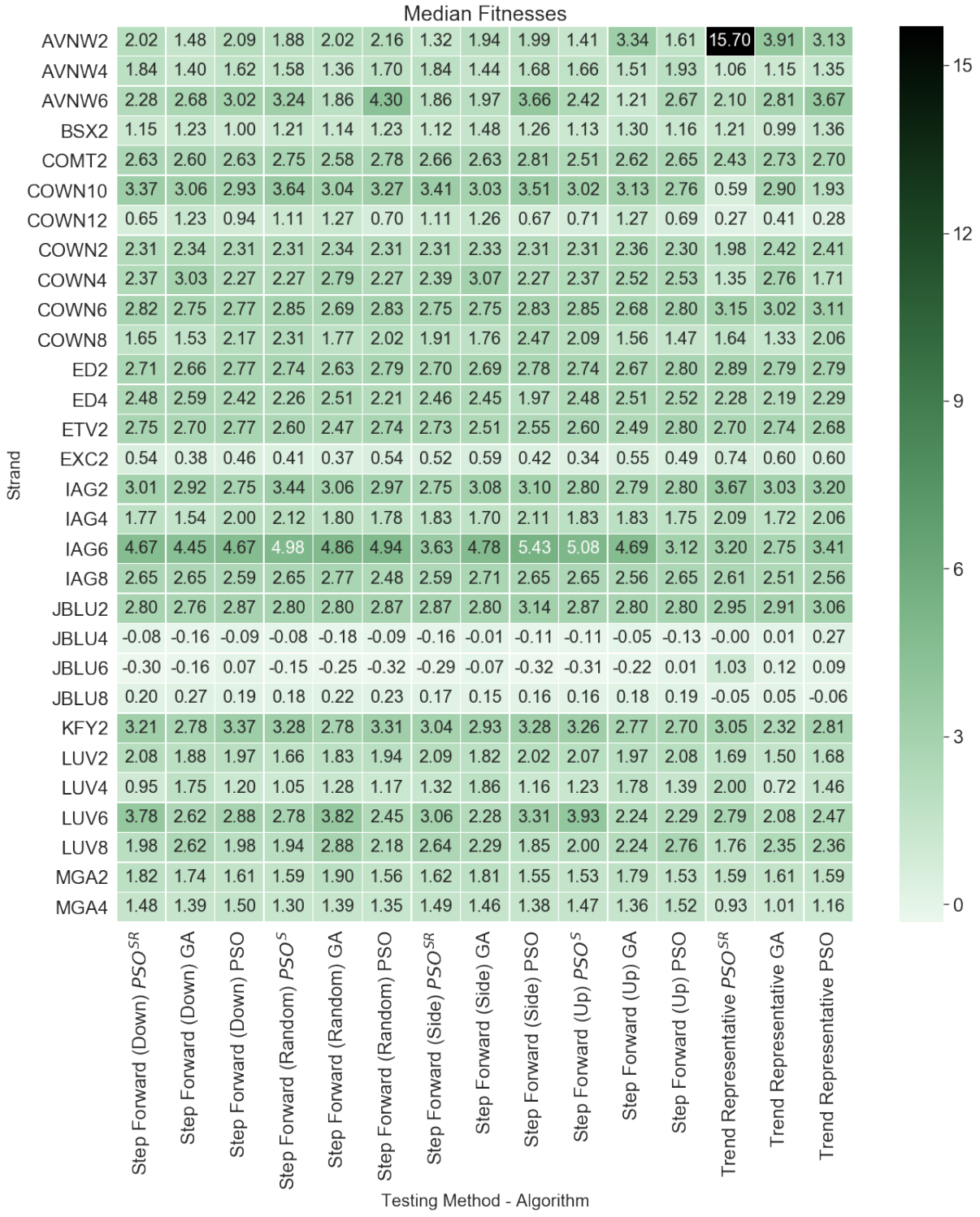

Figure 11: Median Fitnesses Heat Map. Darker colors indicate a higher fitness and thus better solutions.

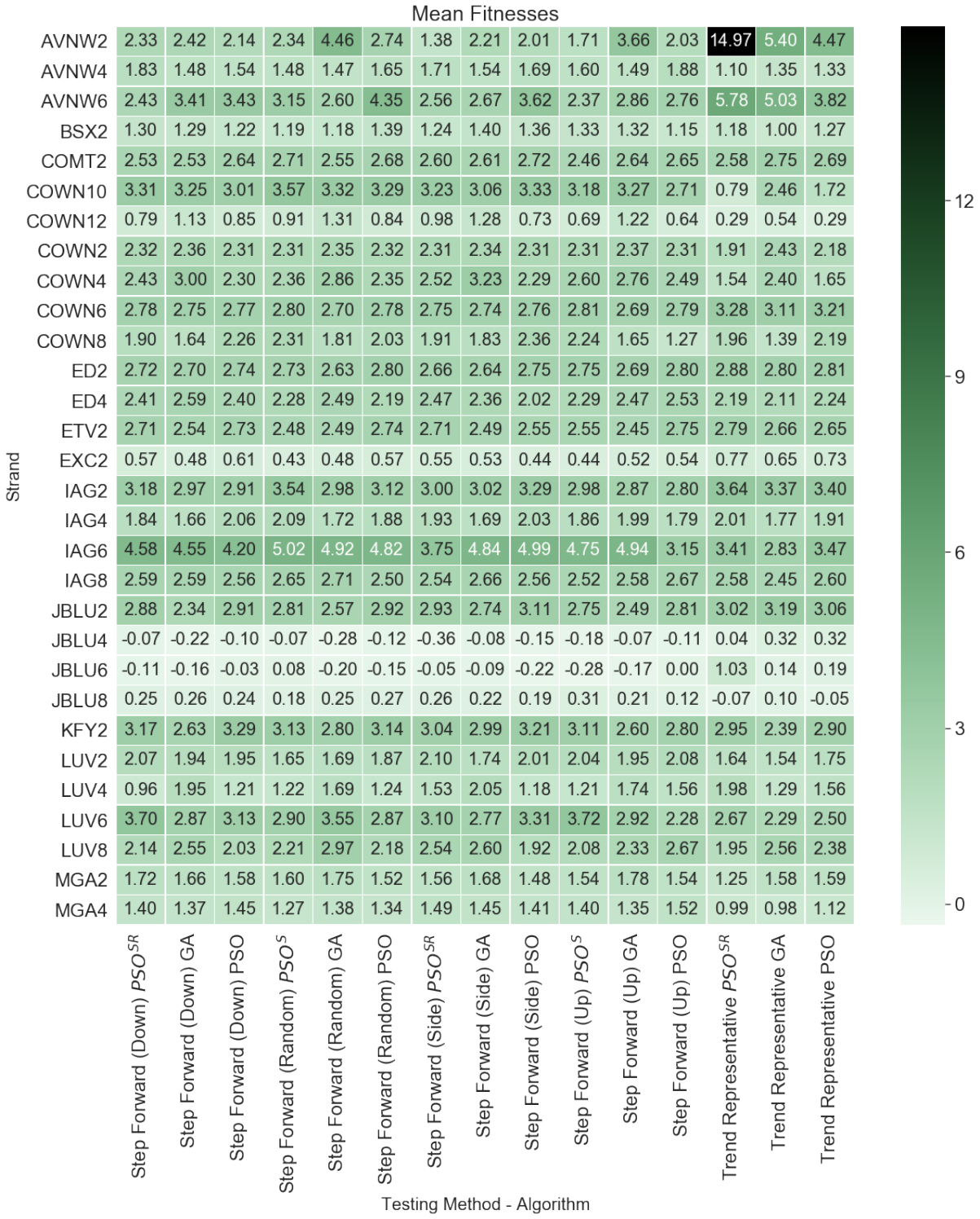

Figure 12: Mean Fitnesses Heat Map. Darker colors indicate a higher fitness and thus better solutions.

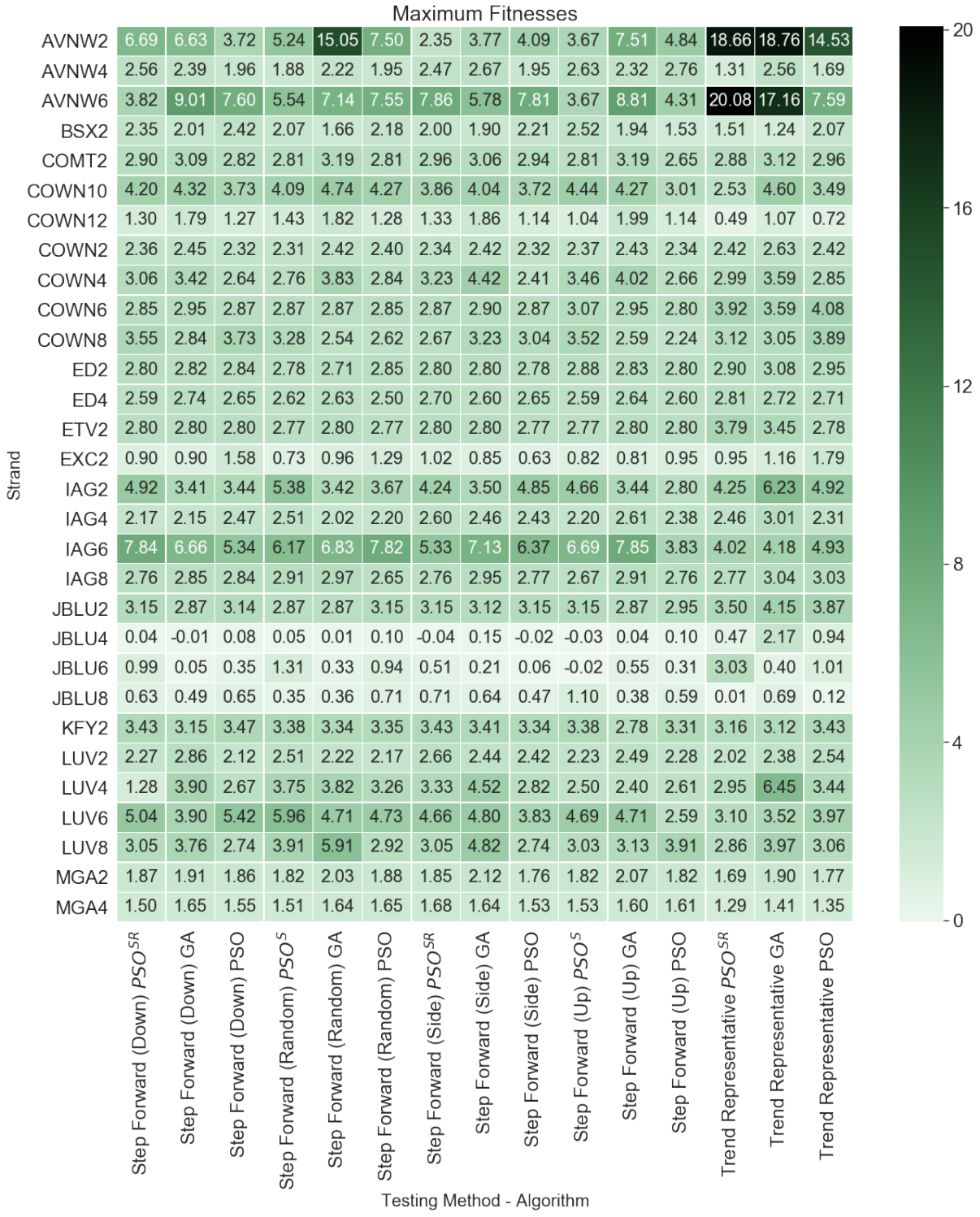

Figure 13: Maximum Fitnesses Heat Map. Darker colors indicate a higher fitness and thus better solutions.

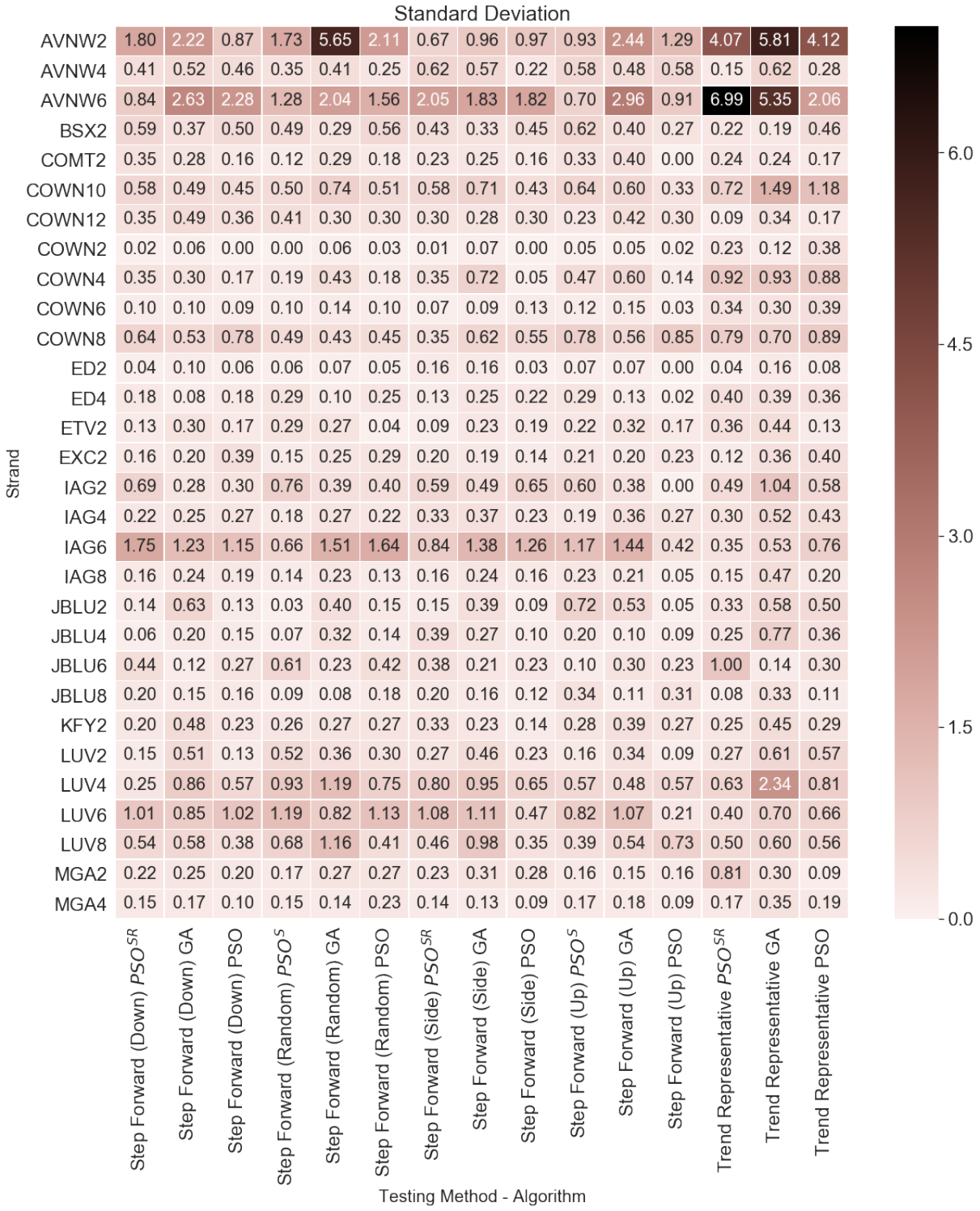

Figure 14: Fitness Standard Deviation Heat Map. Darker colors indicate an increased range implying a decreased stability in an algorithm with a particular testing strand.

Table 9: Average Rankings of each algorithm according to the Friedman non-parametric test with the Holm post-hoc correction over minimum fitness. No statistical differences at the significance level 5% level were observed to identify whether Trend Representative Testing has an edge over Step Forward Testing based on current experimental setup.

Algorithm	Ranking	p	Holm
Step Forward (Up) PSO	5.98333	–	–
Step Forward (Side) PSO	6.0	0.98848	0.05
Step Forward (Random) PSO	6.83333	0.46166	0.025
Step Forward (Side) PSO^{SR}	6.86667	0.44429	0.01667
Step Forward (Down) PSO	7.33333	0.24235	0.0125
Step Forward (Random) PSO^{S}	7.43333	0.20921	0.01
Step Forward (Random) GA	7.8	0.11565	0.00833
Step Forward (Down) PSO^{SR}	8.03333	0.07583	0.00714
Trend Representative PSO^{SR}	8.38333	0.03767	0.00625
Step Forward (Down) GA	8.45	0.03266	0.00556
Step Forward (Up) PSO^{S}	8.6	0.02345	0.005
Step Forward (Side) GA	8.73333	0.01724	0.00455
Step Forward (Up) GA	9.16667	0.00584	0.00417
Trend Representative PSO	**9.38333**	**0.00323**	**0.00385**
Trend Representative GA	**11.0**	**1.39555E-5**	**0.00357**

Table 10: Average Rankings of each algorithm according to the Friedman non-parametric test with the Holm post-hoc correction over median fitness. No statistical differences at the significance level 5% level were observed to identify whether Trend Representative Testing has an edge over Step Forward Testing based on current experimental setup.

Algorithm	Ranking	p	Holm
Step Forward (Side) PSO	6.73333	–	–
Step Forward (Random) PSO	7.11667	0.73991	0.05
Step Forward (Down) PSO^{SR}	7.51667	0.49753	0.025
Trend Representative PSO	7.61667	0.44428	0.01667
Step Forward (Random) GA	7.65	0.42728	0.0125
Step Forward (Side) PSO^{SR}	7.66667	0.41892	0.01
Step Forward (Random) PSO^{S}	7.86667	0.32635	0.00833
Step Forward (Down) PSO	7.9	0.31232	0.00714
Step Forward (Up) PSO^{S}	8.06667	0.24821	0.00625
Step Forward (Down) GA	8.08333	0.24235	0.00556
Step Forward (Side) GA	8.25	0.18902	0.005
Step Forward (Up) PSO	8.3	0.17485	0.00455
Step Forward (Up) GA	8.5	0.12602	0.00417
Trend Representative PSO^{SR}	8.91667	0.05865	0.00385
Trend Representative GA	9.81667	0.00758	0.00357

Table 11: Average Rankings of each algorithm according to the Friedman non-parametric test with the Holm post-hoc correction over mean fitness. No statistical differences at the significance level 5% level were observed to identify whether Trend Representative Testing has an edge over Step Forward Testing based on current experimental setup.

Algorithm	Ranking	p	Holm
Step Forward (Random) PSO	7.23333	–	–
Step Forward (Side) PSO	7.36667	0.90807	0.05
Step Forward (Down) PSO^{SR}	7.5	0.81736	0.025
Step Forward (Random) GA	7.53333	0.79501	0.01667
Step Forward (Side) PSO^{SR}	7.56667	0.77283	0.0125
Step Forward (Side) GA	7.9	0.56370	0.01
Trend Representative PSO	7.96667	0.52537	0.00833
Step Forward (Down) PSO	8.0	0.50672	0.00714
Step Forward (Up) GA	8.06667	0.47049	0.00625
Step Forward (Random) PSO^{S}	8.1	0.45292	0.00556
Step Forward (Down) GA	8.2	0.40250	0.005
Step Forward (Up) PSO^{S}	8.33333	0.34078	0.00455
Step Forward (Up) PSO	8.6	0.23658	0.00417
Trend Representative PSO^{SR}	8.8	0.17485	0.00385
Trend Representative GA	8.83333	0.16586	0.00357

Table 12: Average Rankings of each algorithm according to the Friedman non-parametric test with the Holm post-hoc correction over maximum fitness. No statistical differences at the significance level 5% level were observed to identify whether Trend Representative Testing has an edge over Step Forward Testing based on current experimental setup.

Algorithm	Ranking	p	Holm
Trend Representative GA	5.1	–	–
Step Forward (Side) GA	6.38333	0.26640	0.05
Trend Representative PSO	6.88333	0.12249	0.025
Step Forward (Up) GA	6.96666	0.10597	0.01667
Step Forward (Down) GA	7.35000	0.05135	0.0125
Step Forward (Side) PSO^{SR}	7.5	0.03767	0.01
Step Forward (Random) GA	7.75	0.02173	0.00833
Step Forward (Down) PSO^{SR}	8.08333	0.00978	0.00714
Step Forward (Down) PSO	**8.3**	**0.00558**	**0.00625**
Step Forward (Random) PSO	**8.38333**	**0.00446**	**0.00556**
Step Forward (Random) PSO^{S}	**8.61667**	**0.00232**	**0.005**
Trend Representative PSO^{SR}	**8.66667**	**0.00201**	**0.00455**
Step Forward (Up) PSO^{S}	**8.95**	**8.55458E-4**	**0.00417**
Step Forward (Side) PSO	**10.28333**	**7.15924E-6**	**0.00385**
Step Forward (Up) PSO	**10.78333**	**8.57032E-7**	**0.00357**

6.2.3 Extended Experiments with Trend Representative Testing

Our second set of experiments is designed to see if Trend Representative Testing and PSO can produce competent market timing strategies in a larger scale scenario. The number of technical indicators available to the algorithms was extended from 15 in the previous experiment to 63. In order to evaluate the effectiveness of the algorithms in composing market timing strategies, we tested basic PSO, our novel PSO variants and used GA as a benchmark to compare them against. As all the algorithms have parameters, testing was preceded by hyperparameter optimization performed using the iterated racing procedure (IRace) [58]. Irace is based on three steps: generating candidate configurations based on a sampling of the variables being optimized, racing the configurations and using the results of the racing to adjust the sampling process. During racing, the configurations being considered are evaluated against a dataset reserved for hyperparameter optimization and at configurable checkpoints during the race, configurations that are performing worse than their peers at a statistically significant level are discarded and exit the race. At the end of the race, surviving configurations are used to bias sampling towards the winning values and the process is repeated until the stopping criteria for IRace are satisfied. 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. During hyperparameter optimization, the pruning procedure was found to produce inferior results with the extended library of technical indicators. The IRace algorithm consistently recommended that this feature be turned off during hyperparameter optimization, and so only PSO^S was included in the experiments and PSO^{SR} is excluded. The results of the IRace procedure can be seen in Table 13. In regards to PSO, IRace arrived at swarm sizes that are similar for both variants. The configuration discovered for PSO uses a much lower number of iterations when compared with the configuration for PSO^S. We can also see that the PSO configuration is slightly more reliant on the cognitive component with an l-best neighborhood spanning half the swarm, while the PSO^S configuration is slightly more reliant on the cognitive component with a g-best neighborhood structure. Both PSO configurations favored velocity scaling over Clerc's constriction, with

Table 13: IRace discovered configurations for each of the algorithms tested.

PSO		**PSO^{S}**		**GA**	
Parameter	*Value*	*Parameter*	*Value*	*Parameter*	*Value*
Population	45	Population	59	Population	53
Iterations	28	Iterations	261	Generations	266
Neighbors	26	Neighbors	59	Mutation Probability	0.6306
c1	2.4291	c1	3.2761	Crossover Probability	0.455
c2	3.4185	c2	2.363	Tournament Size	22
Clamp	Scaling	Clamp	Scaling		
Scaling Factor	0.8974	Scaling Factor	0.551		

PSO using relatively larger steps while PSO^{S} uses relatively smaller steps based on the scaling factors. After hyperparameter optimization, we ended up with two PSO configurations: a fast acting l-best PSO and a slower g-best PSO^{S}. As for GA, the findings of IRace show that the configuration had a relatively high mutation rate and a relatively low crossover rate when compared to typical values used for those parameters. The discovered population size and the number of generations are similar in size to those of PSO^{S}.

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 details of the trends can be seen in Table 14. The columns in Table 14 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 of each trend for testing and the remaining 27 are then used for training. Each step in the training and testing procedure is repeated 10 times to cater for the effects of stochasticity.

As mentioned earlier, 63 signal generating components were used in both training and testing. These 63 components are of the technical variety, and contain a selection of momentum indicators, oscillators, accumulation/distribution indicators, candlestick continuation pattern detectors and candlestick reversal pattern

indicators. Where the components took parameters that affected periods of data to look at, an upper limit of 45 days was set, so that we could get at least 5 trading signals within a single trading year (which is on average compromised of 252 days in the US market). Any other parameters for the technical indicators are initialized to random values in a manner that does not break any constraints set on them by each indicator type. A list of the indicators available to the algorithms can be seen in Table 17.

Table 18 shows the minimum, median, mean and maximum fitnesses based on the Annualized Rate of Return (AROR) achieved by each algorithm, dataset and trend. GA showed a slight edge over the PSO variants with all three trends in dataset 1, while PSO^S showed a clear advantage with the downtrend in dataset 7. The basic PSO variant takes most of the wins based on means, when compared to GA and PSO^S. By looking at overall averages in Table 15, we can see that all three algorithms showed a higher overall fitness during a downtrend when compared with the other two trends, leaving us with an unbalanced performance. Nevertheless, performing better in downtrends is positive when compared with buy-and-hold strategies which would fail under such conditions. This issue of unbalanced performance with various trend types can be mitigated once the problem of market timing is approached as a multiobjective one, where the aim is to discover a Pareto front with solutions that maximize fitness across all three trends. By having the performance of the three algorithms explicitly compared across a variety of trends, we have a better approximation of the performance of the strategies produced by these algorithms under live market conditions, and therein lies the advantage of using trend representative testing. With Step Forward testing, we are limited to the price movements in the training section of the data. This can easily lead to strategies that are overfit to one particular type of trend, because that was all they were exposed to during training. With trend representative testing, we explicitly avoid this issue by exposing our algorithms to a variety of trends during both training and testing.

Table 16 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 [33]. 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); 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. We can see from this table that PSO and PSO^{S} were ranked higher than GA in both downtrends and sideways, with a close tie for uptrends, although not at a statistically significant level. This suggests that PSO, both in its basic and modified flavors, is competitive with GA when it comes to the domain of market timing. PSO, in particular, has an advantage over GA in that it achieves these highly competitive results with an order of magnitude fewer number of iterations using a similar population size. The stochastic state update modification has given PSO^{S} a small improvement in ranking when tested in downtrends and uptrends over the PSO, although this is achieved with a greater number of iterations.

6.3 Revisiting Pruning

In Chapter 5, 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. Our PSO variant employing this pruning procedure is known as PSO^{SR}. 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 during our second set of experiments. 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

Table 14: Data strands used for training and testing. Id is composed of the asset symbol and a numerical identifier.

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

Table 15: Overall average fitness by trend for each algorithm.

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

Table 16: 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)	1.7	–	–
	PSO	2.0	0.6708	0.05
	GA	2.3	0.1797	0.025
Sideways	PSO (control)	1.7	–	–
	GA	2.0	0.5023	0.05
	PSO^S	2.3	0.1797	0.025
Uptrend	PSO^S (control)	1.9	–	–
	GA	1.9	0.9999	0.05
	PSO	2.2	0.5023	0.025

to explore other configurations. Shorter solutions have two main advantages over their longer counterparts: they are faster to compute and easier to comprehend. Since shorter solutions are composed of a fewer number of signal generating components, they consume fewer computational resources in order to generate their recommendations than solutions that are relatively longer. Shorter solutions are also more easily comprehensible by human users. It is for these two reasons that shorter solutions are more desirable than longer ones. The challenge is to find a good balance between the quality and size of a solution. In order to arrive at the *least* sufficient subset of components that optimizes a financial metric, we

Table 17: Extended set of technical Indicators used in Trend Representative Testing experiments. All parameters are of type integer, with the exception of weight, which is of the type double.

Indicator	Parameters
Moving Average Converge Diverge (MACD)	Weight, Short Period, Long Period, Signal Period
Aroon Oscillator	Weight, Period, Lookback
Relative Strength Indicator	Weight, Overbought Threshold, Oversold Threshold, Period
Stochastic Oscillator	Weight, Fast K Period, Slow K Period, Slow D Period
Chaikin Oscillator	Weight, Fast Period, Slow Period
On Balance Volume (OBV)	Weight, Period
Two Crows Pattern	Weight, Trend Period, Smooth Period
Three Black Crows Pattern	Weight, Trend Period, Smooth Period
Three Inside Pattern	Weight, Trend Period, Smooth Period
Three Lines Strike Pattern	Weight, Trend Period, Smooth Period
Three Outside Pattern	Weight, Trend Period, Smooth Period
Three Stars In South Pattern	Weight, Trend Period, Smooth Period
Three White Soldiers Pattern	Weight, Trend Period, Smooth Period
Abandoned Baby Pattern	Weight, Trend Period, Smooth Period
Advanced Block Pattern	Weight, Trend Period, Smooth Period
Belt Hold Pattern	Weight, Trend Period, Smooth Period
Break Away Pattern	Weight, Trend Period, Smooth Period
Closing Marbozu Pattern	Weight, Trend Period, Smooth Period
Counter Attack Pattern	Weight, Trend Period, Smooth Period
Dark Cloud Cover Pattern	Weight, Trend Period, Smooth Period
Doji Pattern	Weight, Trend Period, Smooth Period
Doji Star Pattern	Weight, Trend Period, Smooth Period
Dragonfly Doji Pattern	Weight, Trend Period, Smooth Period
Engulfing Pattern	Weight, Trend Period, Smooth Period
Evening Doji Star	Weight, Trend Period, Smooth Period

Table 17: (continued)

Indicator	Parameters
Gap Side by Side White Lines Pattern	Weight, Trend Period, Smooth Period
Gravestone Doji Pattern	Weight, Trend Period, Smooth Period
Hammer Pattern	Weight, Trend Period, Smooth Period
Hanging Man Pattern	Weight, Trend Period, Smooth Period
Harami Pattern	Weight, Trend Period, Smooth Period
Harami Cross Pattern	Weight, Trend Period, Smooth Period
Hikkake Pattern	Weight, Trend Period, Smooth Period
Hikkake Modified Pattern	Weight, Trend Period, Smooth Period
Homing Pigeon Pattern	Weight, Trend Period, Smooth Period
Identical Three Crows Pattern	Weight, Trend Period, Smooth Period
In Neck Pattern	Weight, Trend Period, Smooth Period
Inverted Hammer Pattern	Weight, Trend Period, Smooth Period
Kicking Pattern	Weight, Trend Period, Smooth Period
Kicking by Length Pattern	Weight, Trend Period, Smooth Period
Ladder Bottom Pattern	Weight, Trend Period, Smooth Period
Long Legged Doji Pattern	Weight, Trend Period, Smooth Period
Long Line Pattern	Weight, Trend Period, Smooth Period
Marbozu Pattern	Weight, Trend Period, Smooth Period
Matching Low Pattern	Weight, Trend Period, Smooth Period
Morning Doji Star Pattern	Weight, Trend Period, Smooth Period
Mat Hold Pattern	Weight, Trend Period, Smooth Period
Morning Star Pattern	Weight, Trend Period, Smooth Period
On Neck Pattern	Weight, Trend Period, Smooth Period
Piercing Pattern	Weight, Trend Period, Smooth Period
Rickshaw Man Pattern	Weight, Trend Period, Smooth Period
Rising Falling Three Methods Pattern	Weight, Trend Period, Smooth Period
Separating Lines Pattern	Weight, Trend Period, Smooth Period
Shooting Star Pattern	Weight, Trend Period, Smooth Period
Spinning Top Pattern	Weight, Trend Period, Smooth Period
Stalled Pattern	Weight, Trend Period, Smooth Period

Table 17: (continued)

Indicator	Parameters
Stick Sandwich Pattern	Weight, Trend Period, Smooth Period
Takuri Pattern	Weight, Trend Period, Smooth Period
Tasuki Gap Pattern	Weight, Trend Period, Smooth Period
Thrusting Pattern	Weight, Trend Period, Smooth Period
Tri-Star Pattern	Weight, Trend Period, Smooth Period
Unique Three River Pattern	Weight, Trend Period, Smooth Period
Upside Gap Two Crows Pattern	Weight, Trend Period, Smooth Period
Up/Down Gap Three Methods Pattern	Weight, Trend Period, Smooth Period

Table 18: 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.

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

introduce a novel approach to pruning as an extension to the one presented in Chapter 5. 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 previously. Components who have a 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 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^{SP}. Pseudocode for PSO^{SP} can be seen in Algorithm 8.

As with the other algorithms, PSO^{SP} underwent hyperparameter optimization using IRace before experimentation. The configuration discovered by IRace for the PSO^{SP} algorithm can be seen in Table 19. Table 21 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^{SP} with 5 wins and finally GA with least wins scoring only 4 out of a possible 30. Positive values in Table 21 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 20, 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.

Table 22 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

Table 19: IRace discovered configuration for the PSO^{SP} algorithm.

Parameter	*Value*
Population	30
Iterations	90
Neighborhood Size	30
c1	2.908
c2	3.417
Clamp	Scaling
Scaling Factor	0.5988
Pruning	Enabled
Pruning Threshold	0.1799
Pruning Deadline	15

Table 20: Overall average fitness by trend for each algorithm.

	Algorithm			
Trend	GA	PSO	PSO^{S}	PSO^{SP}
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

[33]. 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.

Algorithm 8 PSO^{SP} with updated pruning procedure. Although this algorithm might seem similar to Algorithm 3, it differs in its use of Trend Representative Testing (line 6) and an updated pruning procedure (line 41).

1. *archive*: an archive used to keep track of elite particles
2. S: swarm
3. N: swarm size
4. x: particle state
5. y: particle personal best
6. $calculate\_fitness$: Refer o Algorithm 6
7. initialize swarm S
8. **repeat**
9. **for** every particle x_i in S **do**
10. **for** every component j in particle i **do**
11. $bias \leftarrow \alpha v_{ij}(t)$
12. **if** $random() < \frac{f(y_{ij}(t))}{f(x_{ij}(t))+f(y_{ij}(t))}$ **then**
13. $cognitive \leftarrow c_1 r_1(y_{ij}(t) - x_{ij}(t))$
14. **else**
15. $cognitive \leftarrow 0$
16. **end if**
17. **if** $random() < \frac{f(\hat{y}_{ij}(t))}{f(x_{ij}(t))+f(\hat{y}_{ij}(t))}$ **then**
18. $social \leftarrow c_2 r_2(\hat{y}_{ij}(t) - x_{ij}(t))$
19. **else**
20. $social \leftarrow 0$
21. **end if**
22. $v_{ij}(t+1) \leftarrow cognitive + social$
23. **if** $clamp = Clerc$ **then**
24. $v_{ij}(t+1) = v_{ij}(t+1) \times clerc\_coefficient$
25. **else if** $clamp = Factor$ **then**
26. $v_{ij}(t+1) \leftarrow v_{ij}(t+1) + bias$
27. $v_{ij}(t+1) = v_{ij}(t+1) \times velocity\_clamp\_factor$
28. **end if**
29. $v_{ij}(t) = v_{ij}(t+1)$
30. $x_{ij} = x_{ij} + v_{ij}(t)$
31. **end for**
32. **end for**

Algorithm 9 PSO^{SP} with updated pruning procedure continued.

```
33.     f(S) ← calculate_fitness(S)
34.     for i: 1 to N do
35.         if f(i) ≥ f(y_i) then
36.             y_i ← x_i
37.         end if
38.     end for
39.     archive ← archive∪ fittest particle in S
40.     if current iteration meets pruning deadline then
41.         updated_pruning()                          ▷ Refer to Algorithm 10
42.     end if
43. until stopping criteria met
44. return fittest particle in archive
```

Algorithm 10 Updated Pruning Procedure

```
function UPDATED_PRUNING( )
    for every particle x_i in S do
        for every component j in x_i do
            if weight(x_ij) < Γ then
                weight(x_ij) ← 0
            end if
        end for
    end for
end function
```

Table 21: 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^{SP}			
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

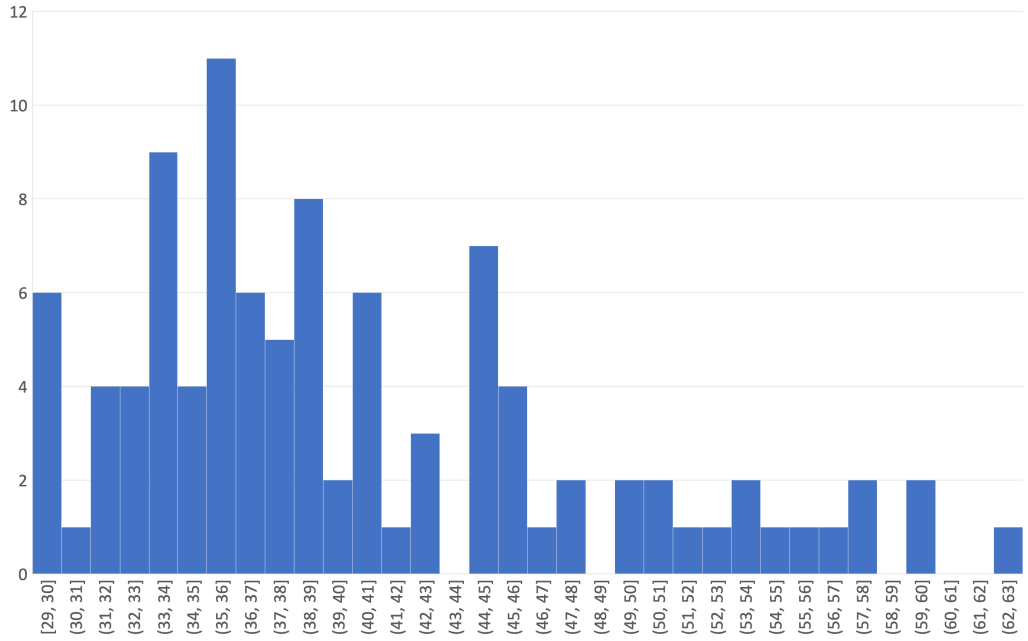

Figure 15: A histogram showing the solution lengths of the solutions from PSO^{SP}. We can clearly see that the majority of solutions range in length between 29 and 46. Only a single solution employed all 63 technical indicators.

We can see from Table 4 that PSO variants that employed the stochastic state update procedure (namely PSO^{S} and PSO^{SP}) 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 24. The second observation is that PSO^{SP}, 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 15 shows a histogram of the solution lengths returned by PSO^{SP} 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

Table 22: 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^{SP}	2.75	0.1939	0.025
	GA	2.9	0.119	0.0167
Sideways	PSO^{SP} (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^{SP}	2.85	0.2602	0.0167

pursuing shorter and shorter solution lengths with the aim of finding the *least* sufficient 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.

6.4 Summary

In this chapter, we addressed the shortcomings of the incumbent testing methodology in market timing (Step Forward Testing) by introducing a novel training and testing methodology known as Trend Representative Testing. This methodology is based on the recommendations in [44] and is based on the concept that by exposing our candidate strategies to a variety of market trends during both training and testing we avoid the possibility of overfitting to any particular type

of trend and have a better estimation of their performance under live trading conditions. Although found to be statistically similar to Step Forward testing after an extensive comparison, we still recommend the usage Trend Representative Testing as it provides a better approximation of performance under a variety of market conditions and reduces the probability of overfitting to any particular trend.

We then used Trend Representative Testing with PSO to evaluate their capacity of producing competent market timing strategies. We tested regular PSO, along with our PSO variants from Chapter 5: PSO^{S} and PSO^{SR}. We also introduce a new PSO variant that has a less destructive approach to pruning: PSO^{SP}. As a benchmark, we compared the strategies produced by our PSO variants with GA, the current incumbent algorithm in the domain of market timing based on volume of publications. Results indicated that the performance of all PSO variants is competitive with our benchmark, as no one algorithm attained a statistically significant edge over another under the three market trends: uptrends, downtrends and sideways movements. This is interesting as some of the PSO variants were able to achieve this at a fraction of the computational cost of GA, and with PSO^{SP} we are able to produce solutions that are shorter in length without a significant sacrifice in solution quality. In the next chapter, we evolve on how we tackle market timing from considering it as a single objective optimization problem to a multiobjective one with competing financial metrics.

Chapter 7

A Multiobjective Approach to Market Timing

"In formal logic, a contradiction is the signal of defeat, but in the evolution of real knowledge it marks the first step in progress toward a victory."

–Alfred North Whitehead

In the previous two chapters, we have seen how PSO can be used to tackle market timing as a single objective optimization problem. In particular, we have seen how the different algorithms can be used to produce market timing strategies that optimize a single metric: profit. This, however, may not be sufficient to produce market timing strategies that are suitable for live trading. Designers of market timing strategies will usually consider multiple objectives at the same time. For example, the designers may seek strategies that maximize profits, minimize losses and exposure to risk, and do so with the least amount of components both for the sake of comprehensibility and speed of execution. In this chapter, we address the limitation of previously proposed algorithms by approaching market timing as a multiobjective optimization problem. We begin by selecting five metrics of financial performance that provide insight not only regarding profit, but also measures of potential loss, risk, solution length and number of transactions generated by candidate market timing strategies. We then move on to describing how we modified the various PSO algorithms introduced in the previous chapters, as well as the GA benchmark, to use our new set of metrics for financial performance

and tackle market timing as a multiobjective optimization problem. To evaluate their performance, we compare the all algorithms to an established multiobjective optimization algorithm (NSGA-II [26]) as well as a single technical indicator commonly used in market timing applications.

7.1 Expansion of Metric Set

The primary characteristic that defines a given problem as a multiobjective optimization one is that we are required to optimize a number of measures of performance that are often competing. In the context of market timing, this means that we are required to consider more than one metric of financial performance in order for our approach to be a multiobjective one. From the metrics introduced in Chapter 2, we have selected the following five metrics for our multiobjective approach:

- Annualized Rate of Return (AROR): The AROR is a measure of expected return that is annualized for standardization. The aim is to maximize AROR, which in turn will maximize profit.

- Annualized Portfolio Risk: This metric provides a measure of volatility in the transactions generated by a candidate market timing strategy. As lower volatility results in a market timing strategy that is more stable (the loss or gain caused by missing or additional transactions can be more readily estimated), this metric is minimized.

- Value at Risk (VaR): The VaR represents the potential value of capital that could be lost during trading, and is annualized for standardization. As this is a measure of potential loss, the aim is to minimize VaR.

- Transactions Count: To reduce the effects of sample error and increase confidence in a candidate market timing strategy, we aim to maximize this metric after considering the various transaction costs involved.

- Solution Length: Shorter solutions are faster to execute and more easily comprehended by the end users. We aim to minimize this metric.

For deeper insight, interpretation and calculation of these metrics, the reader is referred to Chapter 2, where they are discussed in more detail.

7.2 Algorithms

The algorithms adapted to operate as multiobjective are PSO, PSO^{SP} and our GA benchmark algorithm from the previous chapter.[1] We decided to adopt a Pareto dominance-based approach for multiobjective optimization, which results in returning a Pareto set of solutions across the five objectives being optimized. In the following subsections, we discuss the extensions proposed in order to tackle market timing using a dominance-based approach to multiobjective optimization. These include modifications to fitness evaluation, global archive management, GA selection operators, tracking personal bests for PSO and neighborhood selection in PSO. In order to avoid confusion, the multiobjective variants of the algorithms would be prefixed with the Greek letter Lambda, resulting in the multiobjective algorithms being labeled as λ-PSO, λ-PSO^{SP} and λ-GA.[2]

7.2.1 General Modifications

The first modification we performed was to redefine the fitness function. As the approach presented in Chapter 6 used AROR as its sole optimized objective, evaluating fitness was straight forward: higher values are better. In this chapter, we pursue the optimization of five financial metrics, and since we are using a dominance-based approach, a solution can only be considered to be fitter than another if, and only if, the solution at hand is not worse than the one it is being compared to in all five metrics and better than it in at least one metric. All non-dominated solutions, those not outclassed in quality by others on any of the

[1] PSO^{SP} also represents the PSO^{S} algorithm. Since pruning is a user defined parameter in PSO^{SP}, the algorithm can devolve into PSO^{S} if the pruning is disabled. The algorithm used depends on the results of hyperparameter optimization and the value discovered for whether to activate pruning or not.

[2] The reason Lambda was chosen as a prefix is due to its relation to the hypervolume calculation. The hypervolume calculation is one of the way we assess and compare the performance of the multiobjective optimization algorithms presented in this chapter and is based on the Lebesgue measure. The symbol used to represent this measure is the Greek letter Lambda, and we borrow this symbol to demarcate our multiobjective optimization algorithms

metrics that are part of the fitness function, that are discovered throughout the run of the algorithm are then collectively known as the Pareto set, and that is the final result returned by all algorithms.

Since we need to keep track of all the non-dominated solutions as they are discovered, a global archive is maintained by all algorithms for the storage of all non-dominated solutions found. Upon the discovery of a new solution, the solution is compared against the current occupants of the global archive. If the solution is dominated by any of the current occupants, it is rejected. If the new solution remains non-dominated after comparison with the current archive, then it is admitted. Upon admittance, if the new solution dominates one or more occupants, these are removed from the archive. The archive used is unbounded, meaning it can store an arbitrary number of solutions with no limitations in terms of size. Pseudocode for the operations maintained by the archive can be seen in Algorithm 11. The global archive is updated at the end of every iteration of each algorithm. For GA this occurs after the generation of a new population, and for the PSO algorithms, this occurs after the particle state updates.

7.2.2 GA Modifications

Besides adopting a multiobjective fitness function and a global archive, the only other modification to the GA first described in Chapter 6 is modifying the tournament selection used in selecting parents for crossover and mutation. In order to perform tournament selection, we first select random individuals from the current population and attempt to admit them to a temporary archive. Candidates that are dominated by other candidates will either not be allowed admittance into the archive or be removed. A random selection is then made form the surviving, non-dominated occupants to represent the selected individual. This is performed twice with every crossover event and once with every mutation event to produce the required parent(s) for the operation. The pseudocode for λ-GA can be seen in Algorithm 12.

Algorithm 11 Archive used to maintain non-dominated solutions.

1. $d(x, y)$: dominance score – the number of objectives where solution x is better than solution y
2. N: number of objectives being optimized.
3. $archive \leftarrow \{\}$
4. **procedure** ADD_TO_ARCHIVE(x)
5. $remove \leftarrow \{\}$
6. $add\_solution \leftarrow False$
7. **for** every s_i in $archive$ **do**
8. **if** $d(x, s_i) > 0$ **then**
9. $add\_solution \leftarrow add\_solution$ OR $True$
10. **end if**
11. **if** $d(x, s_i) = N$ **then**
12. $remove \leftarrow remove \cup s_i$
13. **end if**
14. **end for**
15. **if** $add\_solution = True$ **then**
16. $archive \leftarrow archive \cup x$
17. **end if**
18. **for** every solution r_i in $remove$ **do**
19. delete r_i from $archive$
20. **end for**
21. **end procedure**
22.
23. **function** SELECT( )
24. **return** random member from $archive$
25. **end function**

Algorithm 12 λ-GA Algorithm

1. N: population size
2. M: mutation probability
3. C: crossover probability
4. *population*: a population of candidate solutions
5. $fitnesses$: fitnesses associated with population at time t
6. *archive*: a non-dominated archive ▷ Refer to Algorithm 11
7. $population_t \leftarrow$ populate with candidate solution of size N
8. $fitnesses_t \leftarrow calculate\_fitness(population_t)$
9. **for** i: 1 to N **do**
10. $archive.add(popultaion_t(i))$
11. **end for**
12. **repeat** ▷ main loop
13. $population_{t+1} \leftarrow \{\}$
14. $i \leftarrow 0$
15. **repeat**
16. **if** $random() < M$ **then**
17. $selected \leftarrow tournament\_selection(population_t)$ ▷ Refer to Algorithm 13
18. $mutated \leftarrow mutate(selected)$
19. $population_{t+1} \leftarrow population_{t+1} \cup mutated$
20. $i \leftarrow i + 1$
21. **end if**
22. **if** $random() < C$ **then**
23. $p_1 \leftarrow tournament\_selection(population_t)$ ▷ Refer to Algorithm 13
24. $p_2 \leftarrow tournament\_selection(population_t)$ ▷ Refer to Algorithm 13
25. $population_{t+1} \leftarrow population_{t+1} \cup crossover(p_1, p_2)$
26. $i \leftarrow i + 2$
27. **end if**
28. **until** i = N
29. $population_{t+1} \leftarrow population_{t+1} \cup archive.select()$
30. $population_t \leftarrow population_{t+1}$
31. $fitnesses_t \leftarrow calculate\_fitness(population_t)$
32. **for** i: 1 to N **do**
33. $archive.add\_to\_archive(popultaion_t(i))$
34. **end for**
35. **until** stopping criteria met
36. **return** *archive*

Algorithm 13 Tournament Selection Procedure for λ-GA

```
function TOURNAMENT_SELECTION( )
    local_archive ← new non-dominated archive          ▷ Refer to Algorithm 11
    for i: 1 to tournament size do
        local_archive.add_to_archive(population_t[random()])
    end for
    return local_archive.select()
end function
```

7.2.3 PSO Modifications

As with the GA, our first modification was to adopt a multiobjective fitness function and a global archive to maintain a Pareto front. For PSO algorithms, our second modification is related to how we maintained personal bests for the particles to participate in the velocity update function. All the particles in our multiobjective PSO maintain a personal archive were non-dominated personal solutions are stored. After a state update and the subsequent fitness evaluation, the particle state is considered for admittance into its own personal archive. As with the global archive, these personal archives are also unbounded. When it comes time to select a personal best to participate in the velocity update function, we select one at random from the the particle's personal archive. Our final modification to PSO algorithms was on how a neighborhood best is selected. For neighborhood selection, we use a temporary archive that stores the non-dominated neighbors of the particle at hand. Each neighboring particle will also select a solution at random from its own personal archive to be admitted into the temporary archive on the condition that it is non-dominated by the current solutions. A solution is then selected at random from this temporary archive to represent the neighborhood best. The pseudocode for λ-PSO can be seen in Algorithm 14.

Another issue we faced, was how to adapt the stochastic state update procedure of PSO^{S} and PSO^{SP} to work with a multiobjective fitness. The solution we proposed for calculating the probabilities of updating the cognitive and social components is to use the dominance score. Given two solutions x and y, we define the dominance score $d(x, y)$ as the number of objectives where solution x is better than solution y. We can calculate the update probabilities in the following

manner:

$$cognitive = \begin{cases} y_i(t) - x_i(t) & \text{if rand()} < |\frac{d(y_i(t), x_i(t))}{d(x_i(t), y_i(t)) + d(y_i(t), x_i(t))}| \\ 0 & \text{otherwise} \end{cases} \tag{14}$$

$$social = \begin{cases} \hat{y}_i(t) - x_i(t) & \text{if rand()} < |\frac{d(\hat{y}_i(t), x_i(t))}{d(x_i(t), \hat{y}_i(t)) + d(\hat{y}_i(t), x_i(t))}| \\ 0 & \text{otherwise} \end{cases} \tag{15}$$

- x: particle
- i: current particle index
- y: personal best
- $\hat{y}$: neighborhood best
- $d(x, y)$: the dominance score of x over y
- $rand()$: random number between 0 and 1

The pseudocode for both λ-PSO$^{\text{S}}$ and λ-PSO$^{\text{SP}}$ can be seen in Algorithm 16

7.3 Experimental Setup

In order to evaluate the performance of the algorithms, we first performed hyperparameter optimization on them using IRace [58]. The IRace procedure was provided with a budget of 300 evaluations and set to return a single configuration for the parameters of the algorithm being tuned. All algorithms have a tuned population size and number of iterations. The additional parameters tuned for GA are mutation probability, crossover probability and tournament size. For all PSO variants, the parameters also included neighborhood size, cognitive coefficient, social coefficient and velocity clamping technique. For λ-PSO$^{\text{SP}}$, pruning frequency and pruning threshold are also considered for tuning. The parameter space searched by IRace for each algorithm can be seen in Table 23 and the final parameter values discovered can be seen in Table 24.

Algorithm 14 λ-PSO Algorithm.

1. *archive*: a non-dominated archive ▷ Refer to Algorithm 11
2. S: swarm
3. N: swarm size
4. x: particle state
5. y: particle personal best
6. initialize swarm S
7. **for** i: 1 to N **do**
8. $\quad archive.add\_to\_archive(x_i)$
9. $\quad y_i.add\_to\_archive(x_i)$
10. **end for**
11. **repeat**
12. $\quad$ **for** every particle x_i in S **do**
13. $\quad\quad$ **for** every component j in particle i **do**
14. $\quad\quad\quad bias \leftarrow \alpha v_{ij}(t)$
15. $\quad\quad\quad cognitive \leftarrow c_1 r_1(y_i.select()_j - x_{ij}(t))$
16. $\quad\quad\quad \hat{y} \leftarrow get\_neighbor(\text{i})$ ▷ Refer to Algorithm 15
17. $\quad\quad\quad social \leftarrow c_2 r_2(\hat{y}_j - x_{ij}(t))$
18. $\quad\quad\quad v_{ij}(t+1) \leftarrow cognitive + social$
19. $\quad\quad\quad$ **if** $clamp = Clerc$ **then**
20. $\quad\quad\quad\quad v_{ij}(t+1) = v_{ij}(t+1) \times clerc\_coefficient$
21. $\quad\quad\quad$ **else if** $clamp = Factor$ **then**
22. $\quad\quad\quad\quad v_{ij}(t+1) \leftarrow v_{ij}(t+1) + bias$
23. $\quad\quad\quad\quad v_{ij}(t+1) = v_{ij}(t+1) \times velocity\_clamp\_factor$
24. $\quad\quad\quad$ **end if**
25. $\quad\quad\quad v_{ij}(t) = v_{ij}(t+1)$
26. $\quad\quad\quad x_{ij} = x_{ij} + v_{ij}(t)$
27. $\quad\quad$ **end for**
28. $\quad$ **end for**
29. $\quad f(S) \leftarrow calculate\_fitness(S)$
30. $\quad$ **for** i: 1 to N **do**
31. $\quad\quad archive.add\_to\_archive(x_i)$
32. $\quad\quad y_i.add\_to\_archive(x_i)$
33. $\quad$ **end for**
34. **until** stopping criteria met
35. **return** *archive*

Algorithm 15 Get_Neighbor function used to select a neighborhood guide for multiobjective PSO algorithms.

```
function GET_NEIGHBOR(i)
    local_archive ← new non-dominated archive
    neighbors ← current neighbors of particle i
    for neighbor n in neighbors do
        local_archive.add_to_archive(n)
        local_archive.add_to_archive(y_n.select())
    end for
    return local_archive.select()
end function
```

All algorithms are then trained and tested using Trend Representative Testing on the same training and testing datasets, as described in Chapter 6. For each algorithm, we hold one partition for testing, and use the other nine partitions for training, and go through the available strands to produce ten distinct training and testing datasets. Each algorithm is run against each dataset ten times to factor in the effects of stochasticity. This would result in 100 experiments being run per algorithm. Also, as in Chapter 6, all algorithms had access to a set of 63 technical indicators to compose individuals from. All the indicator parameters are initialized to random values, with those representing lengths of history constrained to be within 1 and 45, guaranteeing at least five buy, sell or hold signals to be generated with a typical US trading year of 252 days.

7.4 Computational Results

Table 25 shows the best performing solutions discovered per objective being optimized. For every objective, we identify the test strand where the best value was achieved, followed by the best solutions discovered by each algorithm in that strand. In the case of more than one solution being non-dominated, we follow a lexicographical approach in order to determine the winning solution based on the following order: AROR, Portfolio Risk, VaR, Transactions Count and Solution Length. The reasoning behind this ordering is that we seek solutions that

Algorithm 16 λ-PSO$^{\mathrm{S}}$ and λ-PSO$^{\mathrm{SP}}$ algorithms. If Pruning is enabled, then the algorithm becomes λ-PSO$^{\mathrm{SP}}$. Otherwise, it is considered a λ-PSO$^{\mathrm{S}}$ algorithm.

1. *archive*: a non-dominated archive ▷ refer to Algorithm 11
2. S: swarm
3. N: swarm size
4. x: particle state
5. y: particle personal best
6. $d(x, y)$: dominance score – the number of objectives where solution x is better than solution y
7. initialize swarm S
8. **for** i: 1 to N **do**
9. $\quad archive.add\_to\_archive(x_i)$
10. $\quad y_i.add\_to\_archive(x_i)$
11. **end for**
12. **repeat**
13. $\quad$ **for** every particle x_i in S **do**
14. $\quad\quad$ **for** every component j in particle i **do**
15. $\quad\quad\quad bias \leftarrow \alpha v_{ij}(t)$
16. $\quad\quad\quad$ **if** $random() < \frac{d(y_i.select(),x_i(t))}{d(x_i(t),y_i.select())+d(y_i.select(),x_i(t))}$ **then**
17. $\quad\quad\quad\quad cognitive \leftarrow c_1 r_1(y_i.select()_j - x_{ij}(t))$
18. $\quad\quad\quad$ **else**
19. $\quad\quad\quad\quad cognitive \leftarrow 0$
20. $\quad\quad\quad$ **end if**
21. $\quad\quad\quad \hat{y} \leftarrow get\_neighbor(\mathrm{i})$ ▷ Refer to Algorithm 15
22. $\quad\quad\quad$ **if** $random() < \frac{d(\hat{y},x_i(t))}{d(x_i(t),\hat{y})+d(\hat{y},x_i(t))}$ **then**
23. $\quad\quad\quad\quad social \leftarrow c_2 r_2(\hat{y}_j - x_{ij}(t))$
24. $\quad\quad\quad$ **else**
25. $\quad\quad\quad\quad social \leftarrow 0$
26. $\quad\quad\quad$ **end if**
27. $\quad\quad\quad v_{ij}(t+1) \leftarrow cognitive + social$
28. $\quad\quad\quad$ **if** $clamp = Clerc$ **then**
29. $\quad\quad\quad\quad v_{ij}(t+1) = v_{ij}(t+1) \times clerc\_coefficient$
30. $\quad\quad\quad$ **else if** $clamp = Factor$ **then**
31. $\quad\quad\quad\quad v_{ij}(t+1) \leftarrow v_{ij}(t+1) + bias$
32. $\quad\quad\quad\quad v_{ij}(t+1) = v_{ij}(t+1) \times velocity\_clamp\_factor$
33. $\quad\quad\quad$ **end if**
34. $\quad\quad\quad v_{ij}(t) = v_{ij}(t+1)$
35. $\quad\quad\quad x_{ij} = x_{ij} + v_{ij}(t)$
36. $\quad\quad$ **end for**
37. $\quad$ **end for**

Algorithm 17 λ-PSO$^{\text{S}}$ and λ-PSO$^{\text{SP}}$ algorithms continued.

38. $f(S) \leftarrow calculate\_fitness(S)$
39. **for** i: 1 to N **do**
40. $archive.add\_to\_archive(x_i)$
41. $y_i.add\_to\_archive(x_i)$
42. **end for**
43. **if** Pruning is Enabled **then** ▷ Implies λ-PSO$^{\text{SP}}$ if True
44. **if** current iteration meets pruning deadline **then**
45. updated_pruning() ▷ Refer to Algorithm 10
46. **end if**
47. **end if**
48. **until** stopping criteria met
49. **return** *archive*

Table 23: IRace Search Space for Multiobjective Algorithms

Algorithm	**Parameter**	**Search Space**
All	Population Size	integer: 20 – 50
	Iterations/Generations	integer: 20 – 100
GA	Mutation Probability	real: 0 – 1
	Crossover Probability	real: real: 0 – 1
	Tournament Size	integer: 2 – 50
PSO and PSO$^{\text{SP}}$	Neighborhood Size	integer: 1 – 100
	c1	real: 2.00 – 5.00
	c2	real: 2.00 – 5.00
	Clamp Type	factor, clerc
	Velocity Clamp Factor	real: 0 – 1 (only if Clamp Type is factor)
PSO$^{\text{SP}}$	Prune?	Enabled, Disabled
	Pruning Threshold	real: 0 – 1 (only if Prune is Enabled)
	Pruning Deadline	integer: 1 – 100 (only if Prune is Enabled)

maximize profit, first and foremost. In case of a draw we prefer solutions that minimize risk, represented in this case by Portfolio Risk and VaR. When encountering a draw after considering profits and risk, we seek solutions that maximize the number of transactions, as this increases our confidence in them as discussed

Table 24: IRace discovered configurations for each of the algorithms tested.

λ-PSO		**λ-PSO^{SP}**		**λ-GA**	
Parameter	*Value*	*Parameter*	*Value*	*Parameter*	*Value*
Population	50	Population	43	Population	49
Iterations	67	Iterations	83	Generations	24
Neighbors	50	Neighbors	43	Mutation Probability	0.8087
c1	3.7543	c1	4.3644	Crossover Probability	0.1409
c2	2.502	c2	2.5136	Tournament Size	32
Clamp	Scaling	Clamp	Clerc		
Scaling Factor	0.5314	Scaling Factor	–		
		Pruning?	Enabled		
		Pruning Threshold	0.0959		
		Pruning Deadline	2		

in Chapter 2. Finally, when encountering a draw after considering the preceding four metrics, we seek solutions of minimal length as they are less computationally expensive and are more comprehensible by the end user. We can see from Table 25 that λ-PSO^{SP} was the algorithm that returned the best value for all objectives. We cannot, however, claim that the solutions discovered by λ-PSO^{SP} are Pareto-dominant when compared to the other algorithms with the exception of the case of Portfolio Risk with the MGA1 test strand. This is also an interesting solution as it shows the possibility of achieving returns (a positive AROR), using a single technical indicator (Solution length of one), with no losses at all (zero Portfolio Risk and VaR) albeit with a low confidence (a low Transactions Count). With multiobjective optimization, the designer of a market timing strategy now has a choice to prioritize their objectives as they see fit. Apart from the best performing solutions reported per objective in Table 25 there is a multitude of solutions representing various compromises between the optimized objectives from which the designer of the market timing strategy can choose a compromise that best fits their needs. This is only possible by using a dominance-based multiobjective optimization approach to market timing.

When comparing AROR to the values obtained in Chapter 6, we can see that the AROR value of the best performing solution discovered is higher than the maximum value discovered in Chapter 6: 25.16 compared to 16.08 respectively. A

comparison of the AROR values can be seen in Table 27. The maximum values are used in the comparison since AROR is an objective that is maximized. It is interesting to note that all maximum values obtained via the multiobjective versions of the algorithms are higher than their single objective counterparts with the exception of two cases (PSO^{SP} vs PSO on the EXC3 and PUK1 test strands).

In order to evaluate the dominance aspect of the algorithms, we compare the hypervolume covered by the Pareto fronts produced by them. Hypervolume is a measure of the combined dominated space across all objectives being optimized covered by the solutions in the Pareto set returned by the algorithm. Hypervolume measured on two objectives would simply become the area under the curve formed by the Pareto set. When optimizing more than two objectives, the notion of hypervolume is extended to measure the aggregate "volume" across all the objectives that falls under the Pareto set. We use the implementation in [32, 9] to measure hypervolume and the measurements can be seen in Table 26. The hypervolume measurements used a reference point of (0, 1000000, 1000000, 0) for AROR, Portfolio Risk, VaR and Transactions Count respectively. A value of zero was used for AROR as this represents a break-even situation where the strategy neither lost capital nor made any profits. The value of 1000000 is used for Portfolio Risk and VaR as this is the same value used for initial capital when performing backtesting and would represent risking the full volume of the capital allocated for investment. A value of zero is used for the number of transactions as this is technically the least amount of transactions that the market timing strategy can generate: 0 = no transactions. As for solution length, we find the largest value reported in the Pareto set being tested and use that as the reference point. When considering hypervolume, larger values are better and these are highlighted in bold based on the mean values obtained per test strand and algorithm. We can see that λ-PSO^{SP} has outperformed the other two algorithms when it came to downtrends. As for uptrends, we have five wins for λ-PSO^{SP}, two for λ-GA and two for λ-PSO. In sideways movements, we have five wins for λ-PSO^{SP}, three for λ-GA and a single win for λ-PSO. In order to see if any of the algorithms has a statistically significant advantage in terms of dominance, we used the Friedman non-parametric test with the Holm correction on the mean hypervolumes attained by all three algorithms divided by trend type [33]. The results of the Friedman test can be seen in Table

Table 25: Best performance per objective. For every optimized objective, we find the best performing instance. The test strand where this is observed is in brackets next to the primary objective name. The best discovered solution per algorithm observed within the strand and objective at hand is then listed. The top performing solution is highlighted in bold. In case of a tie, we consider the objectives in a lexicographical approach using the following order: AROR, Portfolio Risk, VaR, Transactions Count and Solution Length.

Primary Objective (Strand)			**λ-PSO^{SP}**	**λ-GA**	**λ-PSO**
AROR (ATRO1)	▶	**AROR**	**2.5162E+01**	1.6296E+01	1.8093E+01
		Portfolio Risk	**3.3176E+06**	2.0895E+06	2.5298E+06
		VaR	**3.9490E+05**	2.6933E+05	1.8831E+05
		Transactions Count	**7.8000E+01**	7.0000E+01	6.8000E+01
		Solution Length	**2.0000E+00**	5.7000E+01	3.6000E+01
Portfolio Risk (MGA1)		**AROR**	**2.9173E+00**	2.8743E+00	2.8743E+00
	▶	**Portfolio Risk**	**0.0000E+00**	0.0000E+00	0.0000E+00
		VaR	**0.0000E+00**	0.0000E+00	0.0000E+00
		Transactions Count	**2.0000E+00**	2.0000E+00	2.0000E+00
		Solution Length	**1.0000E+00**	3.8000E+01	3.8000E+01
VaR (JBLU1)		**AROR**	**1.4984E+01**	1.2095E+01	1.3313E+01
		Portfolio Risk	**1.0970E+06**	1.0717e+06	1.0115E+06
	▶	**VaR**	**0.0000E+00**	0.0000E+00	0.0000E+00
		Transactions Count	**1.0200E+02**	6.8000E+01	7.6000E+01
		Solution Length	**3.8000E+01**	3.5000E+01	5.8000E+01
Transactions Count (LUV1)		**AROR**	**-5.1663E-02**	-9.9755E-02	-8.5752E-02
		Portfolio Risk	**4.6062E+05**	4.7781E+05	4.3228E+05
		VaR	**5.8006E+04**	6.5248E+04	6.0277E+04
	▶	**Transactions Count**	**6.3400E+02**	6.1600E+02	6.1800E+02
		Solution Length	**5.1000E+01**	1.0000E+01	4.4000E+01
Solution Length (ATRO1)		**AROR**	**1.8919E+01**	3.6600E+00	1.2000E+01
		Portfolio Risk	**2.5523E+06**	1.0528E+06	1.5216E+06
		VaR	**2.9994E+05**	9.1406E+04	1.4375E+05
		Transactions Count	**7.8000E+01**	6.6000E+01	6.0000E+01
	▶	**Solution Length**	**1.0000E+00**	2.0000E+00	2.6000E+01

Table 26: Hypervolume results for each algorithm over the ten datasets. The minimum, mean and max values are obtained by running each algorithm ten times on each dataset. Best mean results are highlighted in bold.

			λ-PSO^{SP}			λ-GA			λ-PSO		
#	**Trend**	**Test Strand**	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**	**Min**	**Mean**	**Max**
0	↑	IAG1	0.00E+00	2.07E+12	2.07E+13	0.00E+00	3.14E+13	1.57E+14	7.50E+14	**1.08E+15**	1.28E+15
	↔	MGA4	7.59E+15	**8.68E+15**	9.47E+15	5.27E+15	7.46E+15	8.30E+15	4.15E+15	5.15E+15	5.65E+15
	↓	IAG2	3.79E+15	**4.28E+15**	4.80E+15	1.03E+15	1.93E+15	2.28E+15	7.45E+14	1.16E+15	1.30E+15
1	↑	BSX1	3.00E+13	9.91E+14	1.47E+15	1.01E+13	1.81E+15	3.03E+15	1.96E+15	**2.26E+15**	3.14E+15
	↔	LUV1	0.00E+00	2.49E+13	2.49E+14	0.00E+00	1.86E+14	4.65E+14	0.00E+00	**1.95E+14**	4.46E+14
	↓	KFY1	8.82E+15	**9.43E+15**	9.59E+15	2.43E+15	5.13E+15	5.71E+15	2.18E+15	2.78E+15	2.99E+15
2	↑	EXC1	8.61E+15	**9.40E+15**	1.02E+16	3.00E+15	4.47E+15	5.54E+15	1.69E+15	2.37E+15	3.00E+15
	↔	LUV2	9.90E+15	**1.03E+16**	1.05E+16	5.44E+15	6.74E+15	7.75E+15	3.16E+15	3.71E+15	4.46E+15
	↓	KFY2	1.09E+16	**1.20E+16**	1.24E+16	4.88E+15	6.84E+15	8.08E+15	4.41E+15	4.94E+15	5.55E+15
3	↑	AVNW1	2.89E+13	4.05E+14	7.16E+14	2.29E+15	**2.51E+15**	2.63E+15	1.92E+15	2.02E+15	2.13E+15
	↔	PUK1	0.00E+00	2.82E+14	3.93E+14	7.83E+14	**6.63E+15**	1.11E+16	2.74E+15	4.13E+15	5.26E+15
	↓	LUV3	7.14E+15	**7.86E+15**	8.22E+15	1.58E+15	2.86E+15	3.70E+15	1.82E+15	2.49E+15	2.66E+15
4	↑	KFY3	5.96E+15	**7.30E+15**	8.07E+15	1.93E+15	3.57E+15	4.70E+15	2.08E+15	2.66E+15	2.90E+15
	↔	EXC2	2.48E+16	**2.69E+16**	2.89E+16	1.20E+16	1.52E+16	1.66E+16	1.04E+16	1.17E+16	1.25E+16
	↓	LUV4	1.06E+16	**1.16E+16**	1.24E+16	5.54E+14	3.70E+15	6.77E+15	1.67E+15	2.38E+15	2.78E+15
5	↑	EXC3	8.83E+15	**9.78E+15**	1.07E+16	4.63E+15	6.14E+15	6.95E+15	3.55E+15	4.12E+15	4.30E+15
	↔	PUK2	3.26E+15	4.66E+15	6.31E+15	7.38E+15	**1.33E+16**	1.71E+16	9.46E+15	1.06E+16	1.13E+16
	↓	MGA1	2.13E+16	**2.57E+16**	2.69E+16	3.58E+15	4.67E+15	5.22E+15	1.89E+15	3.23E+15	3.93E+15
6	↑	ED1	6.40E+15	**6.86E+15**	7.19E+15	2.31E+15	3.36E+15	3.92E+15	1.46E+15	2.04E+15	2.39E+15
	↔	EXC4	3.04E+16	**3.27E+16**	3.38E+16	9.92E+15	1.37E+16	1.59E+16	9.40E+15	1.06E+16	1.14E+16
	↓	PUK3	4.86E+15	**5.73E+15**	6.10E+15	5.92E+14	1.61E+15	2.20E+15	5.31E+14	1.16E+15	1.48E+15
7	↑	BSX2	1.65E+16	**1.72E+16**	1.75E+16	6.87E+15	9.81E+15	1.17E+16	4.63E+15	5.74E+15	6.20E+15
	↔	ED2	1.25E+16	**1.28E+16**	1.36E+16	7.41E+15	8.86E+15	9.19E+15	4.23E+15	4.82E+15	5.24E+15
	↓	JBLU1	7.44E+15	**8.45E+15**	8.89E+15	5.27E+15	7.34E+15	8.37E+15	2.71E+15	3.50E+15	4.19E+15
8	↑	MGA2	0.00E+00	1.86E+14	2.49E+14	4.68E+14	**2.37E+15**	3.55E+15	9.83E+14	1.55E+15	2.52E+15
	↔	MGA3	2.00E+15	5.01E+15	7.30E+15	4.66E+15	**7.07E+15**	1.15E+16	4.64E+15	5.61E+15	6.39E+15
	↓	ATRO1	1.15E+16	**1.37E+16**	1.53E+16	7.63E+14	6.86E+15	8.59E+15	5.85E+15	6.78E+15	7.20E+15
9	↑	AVNW2	8.00E+15	**8.80E+15**	9.57E+15	3.61E+15	5.24E+15	6.11E+15	4.13E+15	4.58E+15	4.81E+15
	↔	EXC5	1.00E+16	2.19E+16	2.63E+16	7.02E+15	**2.81E+16**	3.67E+16	2.02E+16	2.29E+16	2.50E+16
	↓	AVNW3	9.10E+15	**1.03E+16**	1.12E+16	3.52E+15	6.13E+15	7.61E+15	3.36E+15	3.89E+15	4.20E+15

Table 27: A comparison of the AROR values between the single objective and multiobjective optimization algorithms. The maximum values obtained in the experiments are used since AROR is an objective that is maximized. The higher values per algorithm pair (single objective versus multiobjective) is highlighted in bold.

			Algorithm					
#		Test Strand	PSO^S	λ-PSO^{SP}	GA	λ-GA	PSO	λ-PSO
0	↑	IAG1	-3.42	**0.04**	-1.39	**0.06**	-3.04	**0.97**
	↔	MGA4	1.70	**2.08**	1.60	**2.14**	2.09	**2.30**
	↓	IAG2	2.17	**2.90**	2.16	**2.92**	2.17	**2.95**
1	↑	BSX1	-0.11	**0.63**	-0.13	**0.73**	-0.02	**1.27**
	↔	LUV1	-0.08	**0.07**	-0.01	**0.03**	-0.04	**0.07**
	↓	KFY1	2.17	**3.39**	2.67	**2.81**	2.66	**3.02**
2	↑	EXC1	2.92	**3.44**	2.90	**3.08**	2.80	**3.05**
	↔	LUV2	2.46	**2.79**	2.64	**2.92**	2.62	**2.88**
	↓	KFY2	2.85	**6.52**	2.05	**3.80**	3.61	**3.89**
3	↑	AVNW1	1.22	**0.77**	1.29	**1.83**	1.26	**2.08**
	↔	PUK1	**0.38**	0.06	0.00	**0.76**	0.00	**0.64**
	↓	LUV3	6.12	**6.16**	4.63	**6.00**	4.70	**7.13**
4	↑	KFY3	2.94	**3.08**	2.67	**2.91**	2.80	**3.30**
	↔	EXC2	1.62	**5.14**	1.55	**2.08**	1.60	**2.49**
	↓	LUV4	2.95	**4.89**	2.96	**3.75**	2.80	**3.74**
5	↑	EXC3	**2.39**	2.35	2.14	**2.31**	2.38	**2.55**
	↔	PUK2	0.76	**2.04**	0.51	**2.40**	1.07	**3.16**
	↓	MGA1	3.80	**7.85**	3.15	**3.37**	2.80	**4.35**
6	↑	ED1	2.32	**2.58**	1.98	**2.75**	2.44	**2.84**
	↔	EXC4	3.96	**14.27**	3.72	**5.88**	3.47	**6.91**
	↓	PUK3	3.66	**4.53**	3.66	**3.95**	2.80	**3.64**
7	↑	BSX2	3.76	**4.33**	4.03	**4.15**	3.32	**4.33**
	↔	ED2	2.67	**2.82**	2.70	**2.71**	2.63	**2.75**
	↓	JBLU1	13.09	**14.98**	12.06	**12.63**	11.95	**13.43**
8	↑	MGA2	0.00	**0.09**	-3.39	**0.44**	-0.04	**0.68**
	↔	MGA3	1.34	**1.79**	0.51	**0.79**	0.48	**1.60**
	↓	ATRO1	11.51	**25.16**	16.08	**16.30**	11.31	**18.09**
9	↑	AVNW2	5.67	**12.02**	5.65	**8.60**	3.99	**10.88**
	↔	EXC5	0.56	**3.54**	0.96	**3.85**	0.87	**3.42**
	↓	AVNW3	2.20	**4.79**	2.14	**4.74**	2.10	**3.33**

Table 28: Average rankings of each algorithm according to the Friedman non-parametric test with the Holm post-hoc test over the mean hypervolume. Statistical significance at 0.05 percentage level is observed in downtrends where λ-PSO^{SP} outperforms both λ-PSO and λ-GA.

Trend	Algorithm	Ranking	p-value	Holm
Uptrend	λ-GA (control)	1.8	–	–
	λ-PSO^{SP}	1.8	0.9999	0.05
	λ-PSO	2.4	0.1797	0.025
Sideways	λ-GA (control)	1.5	–	–
	λ-PSO^{SP}	2.0	0.2636	0.05
	λ-PSO	2.5	0.0253	0.025
Downtrend	λ-PSO^{SP} (control)	1.0	–	–
	λ-GA	**2.0**	**0.0253**	**0.05**
	λ-PSO	**3.0**	**7.7442E-6**	**0.025**

28. We can see that λ-PSO^{SP} had a statistically significant advantage over λ-GA and λ-PSO in downtrends; no statistically significant differences when it came to uptrends and sideways movements.

Although hypervolume begins to provide some idea of the performance of the algorithms on the five objectives, it does not provide information regarding the spread of the solutions on the Pareto front. Having solutions that are evenly spread across the Pareto front will provide a larger diversity of solutions for the user to select from. The results also do not provide enough relative context on their performance. We addressed these limitations in the second set of experiments described in the next sections.

7.5 Comparison Against NSGA-II and MACD

The results presented in the previous section were limited in two primary ways. The first limitation is that the results of PSO algorithms are only compared to a GA benchmark that was introduced in this thesis. In this section, we compare the performance of these algorithms against a more established, well-researched multiobjective optimization algorithm and a widely used technique for market timing. The second limitation is that we have no insight on the quality of the

Pareto sets returned by the algorithms in terms of diversity. Ideally, we would prefer sets that cover a larger area of the Pareto front on the objective space and not sets that are clustered around a few point points on the Pareto front.

In order to compare the performance of our algorithms against a well established multiobjective optimization algorithm and a technical analysis indicator that is widely used for market timing applications, we have selected the Nondominated Sorting Genetic Algorithm II (NSGA-II) [26] as a benchmark. The NSGA-II algorithm was first introduced in 2002, and is now amongst the most widely used and cited algorithms within the domain of multiobjective optimization. Early multiobjective optimization algorithms suffered from a number of limitations, including adopting a non-elitist approach, the need for specifying one or more parameters for the algorithm to run and suffering high complexity on the order of $O(MN^3)$, where M is the number of objectives and N is the population size. NSGA-II introduces a number of measures to directly address these limitations. Since its introduction, NSGA-II has been widely adopted in a number of domains, including being the benchmark in a number of market timing applications such as [16]. As for a technical analysis indicator that is widely used market timing applications, we opted to used the Moving Average Converge Diverge (MACD) indicator [76]. The MACD indicator was explored as an example of technical analysis indicators in Chapter 2.

In order to perform the comparison, NSGA-II and the MACD indicator utilized the same experimental setup and dataset as the λ-PSO$^{\text{SP}}$, λ-PSO and λ-GA algorithms in the previous set of experiments. NSGA-II underwent hyperparameter optimization using IRace with the same budget as the other algorithms. The final configuration selected by IRace for NSGA-II can be seen in Table 29. From Table 29, we can see that although the population size was comparable to our GA benchmark, the number of generations needed was three times that of our GA benchmark at 72 compared to the λ-GA's 24. The mutation and crossover probabilities discovered are relatively low compared to typical values used for genetic algorithms, and even our own GA benchmark. There are no values for tournament selection as it is fixed at two as part of the NSGA-II specification. Since MACD also has a set of parameters, we experimented with three versions: one using industry standard default values, one optimized using IRace with the same

budget as the other algorithms and one optimized using IRace with three times the budget available to the other algorithms. During experimentation though, the MACD variants using parameter values from both IRace runs did not produce any transactions in any of our training and testing sets. The only configuration that did result in some transactions was the one with industry default values for the parameters, and hence all further references to MACD will be to the one using those values. The values used for the MACD parameters can also be seen in Table 29.

Table 30 shows the hypervolume results for all algorithms including NSGA-II and MACD. As can be seen from Table 30, NSGA-II and MACD did not achieve any wins in terms of mean hypervolume when compared to λ-PSO$^{\text{SP}}$, λ-PSO and λ-GA. The λ-PSO$^{\text{SP}}$ algorithm maintains a considerable lead with 21 wins, followed by GA with 6 wins and finally PSO with 3 wins. In some instances, λ-PSO$^{\text{SP}}$ showed a mean hypervolume an order of magnitude higher than NSGA-II as can be seen when triplets 6 and 7 were used in testing. On the other hand, MACD on its own failed to produce transactions under certain testing strands, indicated by achieving zero hypervolume. In cases where it did produce transactions, the mean hypervolume is still significantly lower than the best performing algorithm's mean hypervolume for that strand.

Table 31 shows the best performance achieved per objective optimized, including solutions from both NSGA-II and MACD. As with Table 25, we identify the algorithm and strand where the best performing solution was observed in terms of the objective at hand, and compare it with the best performing solutions obtained by the remaining algorithms. Again, in case of more than one solution being non-dominated, we follow a lexicographical approach in our comparison based on the following ordering: AROR, Portfolio Risk, VaR, Transactions Count and Solution Length. As can be seen in Table 31, λ-PSO$^{\text{SP}}$ retained its edge over all other algorithms including the two new benchmarks: NSGA-II and MACD. The fact that MACD was able to achieve a value of 2.016 for AROR without any transactions when looking for the best performing solution for VaR is not a mistake but instead a limitation of how the AROR value is calculated. Since the JBLU1 strand is 125 days, plugging in that number in the formula from Chapter 2 with the same initial and final capital would result in the value observed. It is in such situations that

Table 29: IRace discovered configurations for NSGA-II and MACD.

NSGA-II		**MACD**	
Parameter	*Value*	*Parameter*	*Value*
Population Size	47	Short EMA Period	12
Generations	72	Long EMA Period	26
Mutation Probability	0.0462	Signal Period	9
Crossover Probability	0.5559		

it is important to consider the number of transactions and, hence, our confidence in the strategy. A transaction count of zero would result in the lowest confidence possible in a candidate market timing strategy, and therefore the candidate presented by MACD for the JBLU1 strand is an unworthy one. We can also see that while optimizing for Solution Length, λ-PSO^{SP} was able to achieve a higher AROR and Transactions Count than MACD, albeit with a higher risk profile, while also utilizing a single technical indicator. The indicator that λ-PSO^{SP} ended up utilizing was the Hammer Candlestick pattern with a trending period of 24 and a smoothing period of 10. Another interesting phenomenon is that λ-PSO^{SP} was able to achieve the highest AROR while optimizing for that particular metric by using only two technical indicators. With the exception of MACD, this is an order of magnitude lower in length when compared with λ-PSO, λ-GA and NSGA-II. The strategy proposed by λ-PSO^{SP} in that scenario depends on the Harami Cross and Rickshaw Man Candlestick patterns with the following parameters:

- Harami Cross: Trend Period: 21, Smoothing Period: 5, Weight: 0.97
- Rickshaw Man: Trend Period: 29, Smoothing Period: 12, Weight: 0.03

Table 32 shows the comparison of all the multiobjective optimization algorithms mentioned thus far with their single objective counterparts from the previous chapter, including the additional benchmarks of MACD and NSGA-II. Instead of highlighting the higher achievers in a pairwise comparison as in Table 27, here we highlight the highest achiever across all the algorithms per testing strand. As with Table 27, the reason behind including this comparison is to see whether following a Pareto dominance-based approach to market timing improved the quality of the

Table 30: Hypervolume results for each algorithm over the ten datasets including NSGA-II and MACD as benchmarks. The minimum, mean and max values are obtained by running each algorithm ten times on each dataset. Best mean results are highlighted in bold.

			λ-PSO^{SP}			λ-GA			λ-PSO			NSGA-II			MACD
#	Trend	Strand	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	Min	Mean	Max	
0	↑	IAG1	0.00E+00	2.07E+12	2.07E+13	0.00E+00	3.14E+13	1.57E+14	7.50E+14	**1.08E+15**	1.28E+15	0.00E+00	0.00E+00	0.00E+00	0.00E+00
	↔	MGA4	7.59E+15	**8.68E+15**	9.47E+15	5.27E+15	7.46E+15	8.30E+15	4.15E+15	5.15E+15	5.65E+15	2.18E+14	8.83E+14	2.23E+15	3.33E+14
	↓	IAG2	3.79E+15	**4.28E+15**	4.80E+15	1.03E+15	1.93E+15	2.28E+15	7.45E+14	1.16E+15	1.30E+15	3.10E+13	2.06E+14	5.70E+14	1.44E+15
1	↑	BSX1	3.00E+13	9.91E+14	1.47E+15	1.01E+13	1.81E+15	3.03E+15	1.96E+15	**2.26E+15**	3.14E+15	0.00E+00	2.90E+14	1.29E+15	0.00E+00
	↔	LUV1	0.00E+00	2.49E+13	2.49E+14	0.00E+00	1.86E+14	4.65E+14	0.00E+00	**1.95E+14**	4.46E+14	0.00E+00	0.00E+00	0.00E+00	0.00E+00
	↓	KFY1	8.82E+15	**9.43E+15**	9.59E+15	2.43E+15	5.13E+15	5.71E+15	2.18E+15	2.78E+15	2.99E+15	5.40E+14	1.36E+15	2.47E+15	0.00E+00
2	↑	EXC1	8.61E+15	**9.40E+15**	1.03E+16	3.00E+15	4.47E+15	5.54E+15	1.69E+15	2.37E+15	3.00E+15	3.12E+13	6.36E+14	1.73E+15	5.30E+14
	↔	LUV2	9.90E+15	**1.03E+16**	1.05E+16	5.44E+15	6.74E+15	7.75E+15	3.16E+15	3.71E+15	4.46E+15	1.86E+14	1.05E+15	2.30E+15	1.27E+15
	↓	KFY2	1.09E+16	**1.20E+16**	1.24E+16	4.88E+15	6.84E+15	8.08E+15	4.41E+15	4.94E+15	5.55E+15	2.52E+14	1.01E+15	1.86E+15	0.00E+00
3	↑	AVNW1	2.89E+13	4.05E+14	7.16E+14	2.29E+15	**2.51E+15**	2.63E+15	1.92E+15	2.02E+15	2.13E+15	0.00E+00	3.30E+13	1.46E+14	0.00E+00
	↔	PUK1	0.00E+00	2.82E+14	3.93E+14	7.83E+14	**6.63E+15**	1.11E+16	2.74E+15	4.13E+15	5.26E+15	0.00E+00	2.00E+14	8.40E+14	4.67E+14
	↓	LUV3	7.14E+15	**7.86E+15**	8.22E+15	1.58E+15	2.86E+15	3.70E+15	1.82E+15	2.49E+15	2.66E+15	1.24E+14	6.30E+14	1.19E+15	5.08E+12
4	↑	KFY3	5.96E+15	**7.30E+15**	8.07E+15	1.93E+15	3.57E+15	4.70E+15	2.08E+15	2.66E+15	2.90E+15	0.00E+00	8.35E+14	1.72E+15	0.00E+00
	↔	EXC2	2.48E+16	**2.69E+16**	2.89E+16	1.20E+16	1.52E+16	1.66E+16	1.04E+16	1.17E+16	1.25E+16	0.00E+00	2.05E+15	6.19E+15	0.00E+00
	↓	LUV4	1.06E+16	**1.16E+16**	1.24E+16	5.54E+14	3.70E+15	6.77E+15	1.67E+15	2.38E+15	2.78E+15	0.00E+00	9.08E+14	1.75E+15	2.98E+15
5	↑	EXC3	8.83E+15	**9.78E+15**	1.07E+16	4.63E+15	6.14E+15	6.95E+15	3.55E+15	4.12E+15	4.30E+15	3.70E+14	9.18E+14	1.76E+15	9.38E+13
	↔	PUK2	3.26E+15	4.66E+15	6.31E+15	7.38E+15	**1.33E+16**	1.71E+16	9.46E+15	1.06E+16	1.13E+16	4.66E+14	1.66E+15	3.90E+15	8.41E+14
	↓	MGA1	2.13E+16	**2.57E+16**	2.69E+16	3.58E+15	4.68E+15	5.23E+15	1.89E+15	3.23E+15	3.93E+15	2.62E+14	7.06E+14	1.52E+15	3.86E+14
6	↑	ED1	6.40E+15	**6.86E+15**	7.19E+15	2.31E+15	3.36E+15	3.92E+15	1.46E+15	2.04E+15	2.39E+15	7.19E+13	3.83E+14	1.35E+15	0.00E+00
	↔	EXC4	3.04E+16	**3.27E+16**	3.38E+16	9.92E+15	1.37E+16	1.59E+16	9.40E+15	1.06E+16	1.14E+16	1.16E+14	1.08E+15	3.65E+15	2.57E+14
	↓	PUK3	4.86E+15	**5.73E+15**	6.10E+15	5.92E+14	1.61E+15	2.20E+15	5.31E+14	1.16E+15	1.48E+15	3.29E+13	1.22E+14	3.59E+14	1.05E+15
7	↑	BSX2	1.65E+16	**1.72E+16**	1.75E+16	6.87E+15	9.81E+15	1.17E+16	4.63E+15	5.74E+15	6.20E+15	4.44E+14	1.27E+15	2.16E+15	3.15E+14
	↔	ED2	1.25E+16	**1.28E+16**	1.36E+16	7.41E+15	8.86E+15	9.19E+15	4.23E+15	4.82E+15	5.24E+15	3.78E+14	1.37E+15	2.43E+15	3.31E+15
	↓	JBLU1	7.44E+15	**8.45E+15**	8.89E+15	5.27E+15	7.34E+15	8.37E+15	2.71E+15	3.50E+15	4.19E+15	7.95E+13	7.08E+14	2.02E+15	0.00E+00
8	↑	MGA2	0.00E+00	1.86E+14	2.49E+14	4.68E+14	**2.37E+15**	3.55E+15	9.83E+14	1.55E+15	2.52E+15	0.00E+00	4.20E+13	2.11E+14	0.00E+00
	↔	MGA3	2.00E+15	5.01E+15	7.30E+15	4.66E+15	**7.07E+15**	1.15E+16	4.64E+15	5.61E+15	6.39E+15	0.00E+00	4.02E+14	1.32E+15	0.00E+00
	↓	ATRO1	1.15E+16	**1.37E+16**	1.53E+16	7.63E+14	6.86E+15	8.59E+15	5.85E+15	6.78E+15	7.17E+15	0.00E+00	2.51E+14	6.95E+14	5.56E+14
9	↑	AVNW2	8.00E+15	**8.80E+15**	9.57E+15	3.61E+15	5.24E+15	6.11E+15	4.13E+15	4.58E+15	4.82E+15	1.77E+14	9.44E+14	1.92E+15	0.00E+00
	↔	EXC5	1.00E+16	2.19E+16	2.63E+16	7.02E+15	**2.81E+16**	3.67E+16	2.02E+16	2.29E+16	2.50E+16	0.00E+00	2.87E+15	1.34E+16	7.96+14
	↓	AVNW3	9.10+15	**1.03E+16**	1.16E+16	3.52E+15	6.13E+15	7.61E+15	3.36E+15	3.86E+15	4.20E+15	1.66E+14	1.27E+15	2.35E+15	0.00E+00

Table 31: Best Performance Per Objective including both NSGA-II and MACD. For every optimized objective, we find the best performing instance. The test strand where this is observed is in brackets next to the primary objective name. The best discovered solution per algorithm observed within the strand and objective at hand is then listed. The top performing solution is highlighted in bold. In case of a tie, we consider the objectives in a lexicographical approach using the following order: AROR, Portfolio Risk, VaR, Transactions Count and Solution Length.

Primary Objective			**λ-PSO^{SP}**	**λ-GA**	**λ-PSO**	**NSGA-II**	**MACD**
AROR (ATRO1)	▶	**AROR**	**2.5162E+01**	1.6296E+01	1.8093E+01	1.3302E+01	1.5705E+00
		Portfolio Risk	**3.3176E+06**	2.0895E+06	2.5298E+06	2.0316E+06	4.1877E+05
		VaR	**3.9490E+05**	2.6933E+05	1.8831E+05	2.0658E+05	1.7218E+04
		Transactions Count	**7.8000E+01**	7.0000E+01	6.8000E+01	6.4000E+01	1.0000E+01
		Solution Length	**2.0000E+00**	5.7000E+01	3.6000E+01	3.2000E+01	1.0000E+00
Portfolio Risk (MGA1)		**AROR**	**2.9173E+00**	2.8743E+00	2.8743E+00	2.8000E+00	2.6536E+00
	▶	**Portfolio Risk**	**0.0000E+00**	0.0000E+00	0.0000E+00	0.0000E+00	4.1339E+05
		VaR	**0.0000E+00**	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
		Transactions Count	**2.0000E+00**	2.0000E+00	2.0000E+00	0.0000E+00	4.0000E+00
		Solution Length	**1.0000E+00**	3.8000E+01	3.8000E+01	1.2000E+01	1.0000E+00
VaR (JBLU1)		**AROR**	**1.4984E+01**	1.2095E+01	1.3313E+01	1.1953E+01	2.0160E+00
		Portfolio Risk	**1.0970E+06**	1.0717E+06	1.0115E+06	1.1022E+06	0.0000E+00
	▶	**VaR**	**0.0000E+00**	0.0000E+00	0.0000E+00	0.0000E+00	0.0000E+00
		Transactions Count	**1.0200E+02**	6.8000E+01	7.6000E+01	6.6000E+01	0.0000E+00
		Solution Length	**3.8000E+01**	3.5000E+01	5.8000E+01	3.3000E+01	1.0000E+00
Transactions Count (LUV1)		**AROR**	**-5.1663E-02**	-9.9755E-02	-8.5752E-02	-9.9213E-02	6.2546E-01
		Portfolio Risk	**4.6062E+05**	4.7781E+05	4.3228E+05	6.1172E+05	1.1358E+06
		VaR	**5.8006E+04**	6.5248E+04	6.0277E+04	8.3613E+04	3.1387E+04
		Transactions Count	**6.3400E+02**	6.1600E+02	6.1800E+02	6.2800E+02	9.2000E+01
	▶	**Solution Length**	**5.1000E+01**	1.0000E+01	4.4000E+01	5.0000E+00	1.0000E+00
Solution Length (ATRO1)		**AROR**	**1.8919E+01**	3.6600E+00	1.2000E+01	9.7551E-01	1.5705E+00
		Portfolio Risk	**2.5523E+06**	1.0528E+06	1.5216E+06	9.9237E+05	4.1877E+05
		VaR	**2.9994E+05**	9.1406E+04	1.4375E+05	1.2718E+05	1.7218E+04
		Transactions Count	**7.8000E+01**	6.6000E+01	6.0000E+01	8.0000E+01	1.0000E+01
	▶	**Solution Length**	**1.0000E+00**	2.0000E+00	2.6000E+01	7.0000E+00	1.0000E+00

Table 32: A comparison of the AROR values between the single objective and multiobjective optimization algorithms including NSGA-II and MACD benchmarks. The maximum values obtained in the experiments are used since AROR is an objective that is maximized. The highest value per Test Strand is highlighted in bold.

			Algorithm							
#		Test Strand	PSO^{S}	λ-PSO^{SP}	GA	λ-GA	PSO	λ-PSO	NSGA-II	MACD
0	↑	IAG1	-3.42	0.04	-1.39	0.06	-3.04	0.97	-0.04	**3.04**
	↔	MGA4	1.70	2.08	1.60	2.14	2.09	**2.30**	1.87	2.07
	↓	IAG2	2.17	2.90	2.16	2.92	2.17	**2.95**	2.42	2.43
1	↑	BSX1	-0.11	0.63	-0.13	0.73	-0.02	1.27	1.13	**1.57**
	↔	LUV1	-0.08	0.07	-0.01	0.03	-0.04	0.07	-0.02	**0.63**
	↓	KFY1	2.17	**3.39**	2.67	2.81	2.66	3.02	3.16	2.55
2	↑	EXC1	2.92	**3.44**	2.90	3.08	2.80	3.05	2.98	2.89
	↔	LUV2	2.46	2.79	2.64	**2.92**	2.62	2.88	2.86	2.53
	↓	KFY2	2.85	6.52	2.05	3.80	3.61	**3.89**	3.42	1.87
3	↑	AVNW1	1.22	0.77	1.29	1.83	1.26	2.08	2.07	**2.77**
	↔	PUK1	0.38	0.06	0.00	**0.76**	0.00	0.64	0.40	0.33
	↓	LUV3	6.12	6.16	4.63	6.00	4.70	**7.13**	4.67	1.83
4	↑	KFY3	2.94	3.08	2.67	2.91	2.80	**3.30**	2.91	2.93
	↔	EXC2	1.62	**5.14**	1.55	2.08	1.60	2.49	2.48	1.39
	↓	LUV4	2.95	**4.89**	2.96	3.75	2.80	3.74	3.68	2.53
5	↑	EXC3	2.39	2.35	2.14	2.31	2.38	**2.55**	2.31	2.11
	↔	PUK2	0.76	2.04	0.51	2.40	1.07	**3.16**	3.09	0.99
	↓	MGA1	3.80	**7.85**	3.15	3.37	2.80	4.35	3.70	2.65
6	↑	ED1	2.32	2.58	1.98	2.75	2.44	2.84	2.59	**2.95**
	↔	EXC4	3.96	**14.27**	3.72	5.88	3.47	6.91	4.87	2.08
	↓	PUK3	3.66	**4.53**	3.66	3.95	2.80	3.64	3.24	3.22
7	↑	BSX2	3.76	**4.33**	4.03	4.15	3.32	**4.33**	3.92	2.54
	↔	ED2	2.67	**2.82**	2.70	2.71	2.63	2.75	2.75	2.24
	↓	JBLU1	13.09	**14.98**	12.06	12.63	11.95	13.43	11.95	2.01
8	↑	MGA2	0.00	0.09	-3.39	0.44	-0.04	0.68	0.20	**1.93**
	↔	MGA3	1.34	**1.79**	0.51	0.79	0.48	1.60	0.81	0.91
	↓	ATRO1	11.51	**25.16**	16.08	16.30	11.31	18.09	13.30	1.57
9	↑	AVNW2	5.67	12.02	5.65	8.60	3.99	10.88	**11.48**	1.60
	↔	EXC5	0.56	3.54	0.96	**3.85**	0.87	3.42	3.17	0.69
	↓	AVNW3	2.20	**4.79**	2.14	4.74	2.10	3.33	3.54	1.75

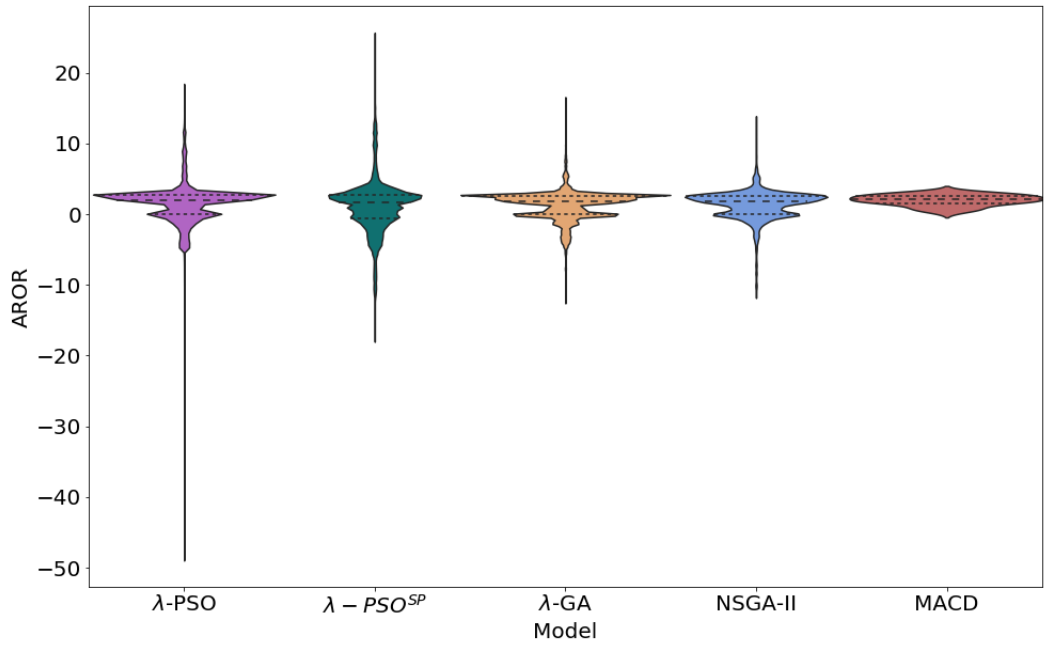

Figure 16: Violin plot comparing AROR performance for all multiobjective algorithms, along with NSGA-II and MACD, across all testing strands. We can see that the medians and the bulk of their solution distributions for all algorithms are close to each other. Only λ-PSO$^{\text{SP}}$ has shown a tail achieving higher AROR values than the other algorithms in the comparison.

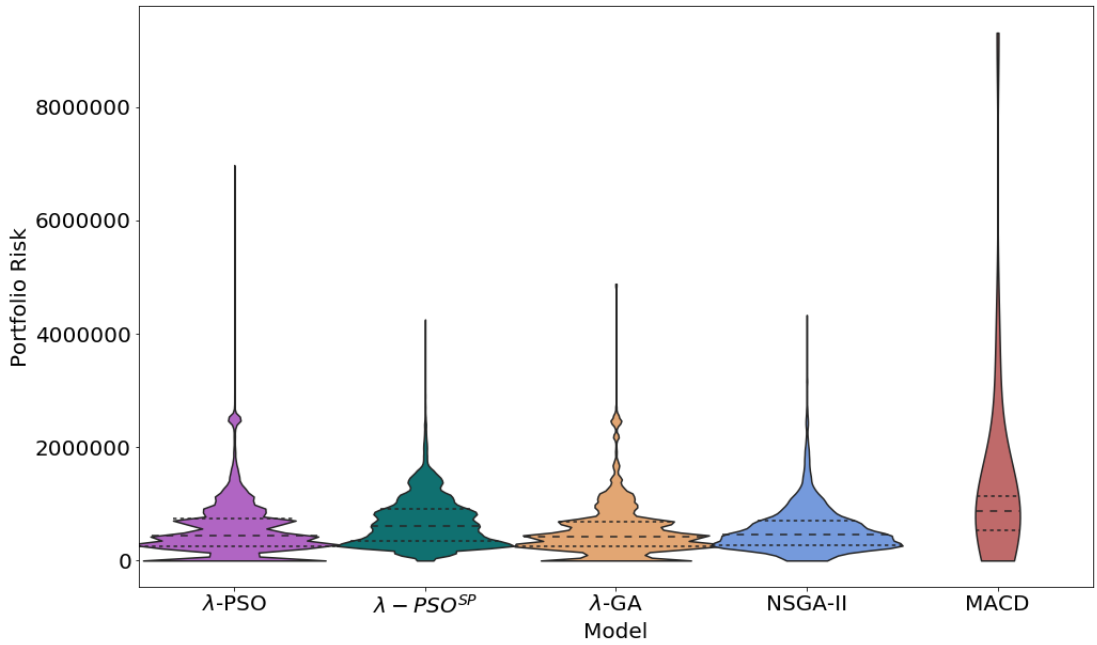

Figure 17: Violin plot comparing Annualized Portfolio Risk performance for all multiobjective algorithms, along with NSGA-II and MACD, across all testing strands. With the exception of MACD, all multiobjective algorithms have their medians and the bulk of their solution distributions are close to each other. MACD, on the other hand displays a higher median and a long tail stretching into higher values of risk, indicating a worse performance when compared to the other algorithms.

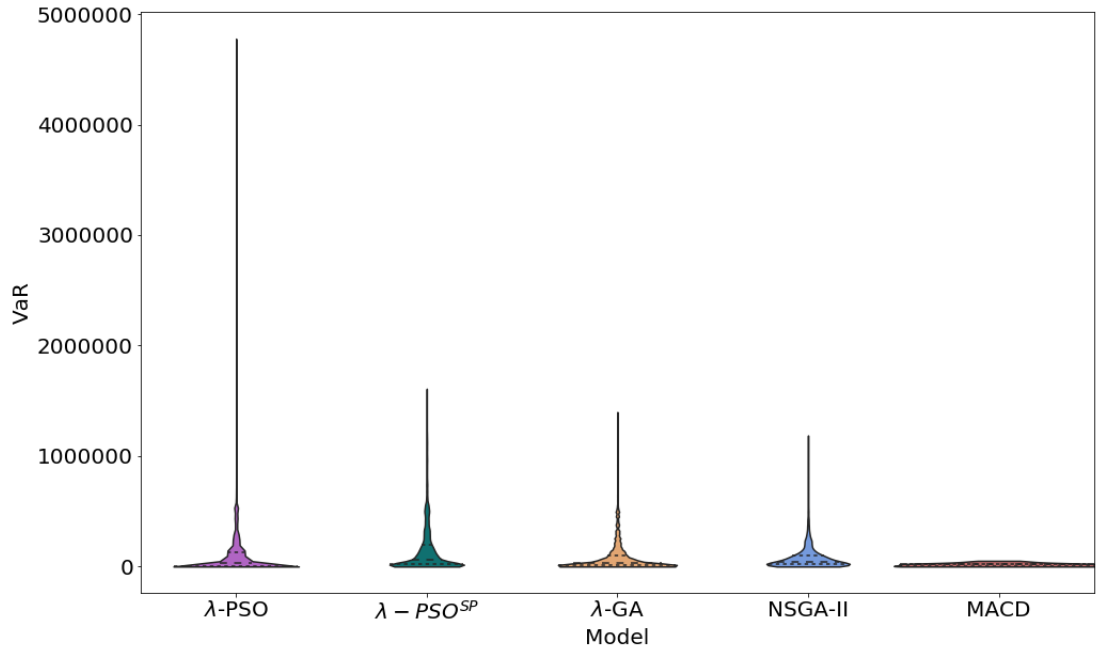

Figure 18: Violin plot comparing VaR performance for all multiobjective algorithms, along with NSGA-II and MACD, across all testing strands. We can see that the medians and the bulk of their solution distributions for all algorithms are close to each other. λ-PSO displays an exceptionally long tail stretching into higher values of VaR indicating a higher potential for losses when compared to the other algorithms albeit at a lower probability.

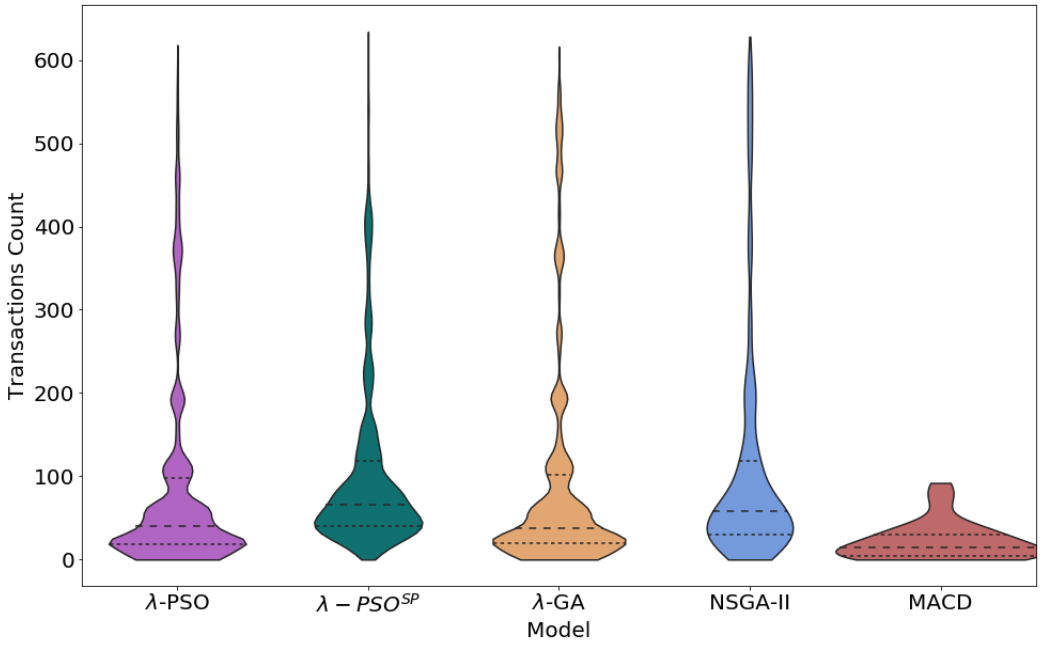

Figure 19: Violin plot comparing transactions count for all multiobjective algorithms, along with NSGA-II and MACD, across all testing strands. The highest median observed was that of λ-PSO$^{\text{SP}}$, followed by NSGA-II, λ-PSO, λ-GA and finally MACD. With the exception of MACD, all algorithms have tails extending into higher transaction counts. Based on these observations, we can conclude that solutions returned by λ-PSO$^{\text{SP}}$ are relatively the most stable and have the lowest sample error. MACD solutions, on the other hand, are relatively the least stable and have the highest sample error compared to the solutions returned by the other algorithms.

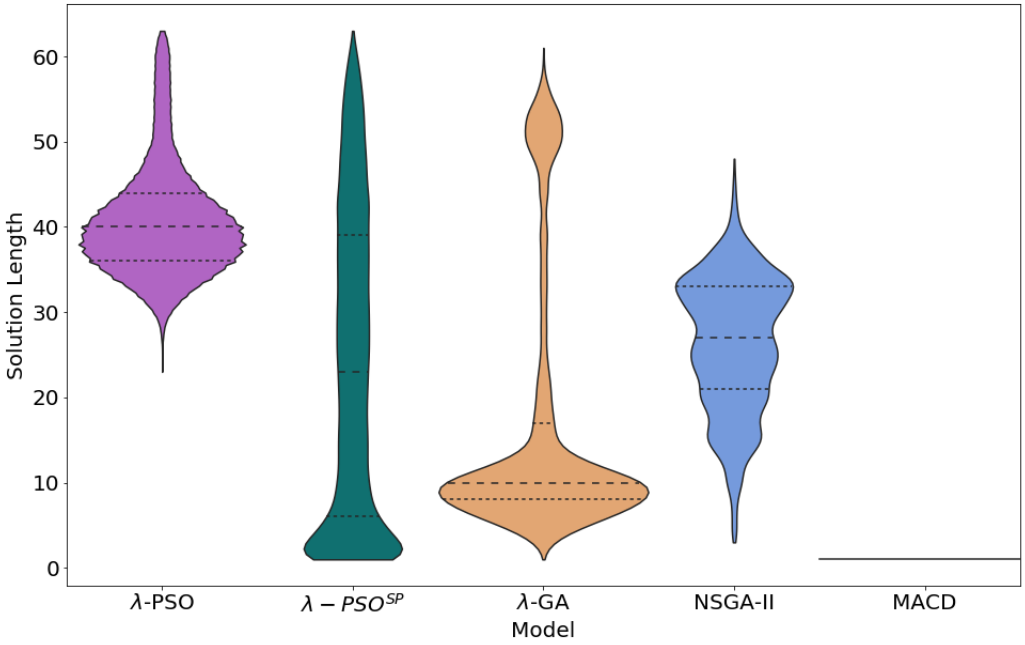

Figure 20: Violin plot comparing solution length for all multiobjective algorithms, along with NSGA-II and MACD, across all testing strands. When comparing λ-PSO to λ-PSO$^{\text{SP}}$, we can see that the pruning procedure has resulted in comparatively shorter solutions based on the lower median and the main body of the solution distribution manifesting significantly lower than λ-PSO. When considering the other algorithms, we can see that λ-PSO$^{\text{SP}}$ has achieved shorter solutions and are directly comparable to the MACD solution which had a fixed length of one component.

Table 33: Average rankings of each algorithm according to the Friedman non-parametric test with the Holm post-hoc test over the mean hypervolume. Although λ-PSO^{SP} lost its edge over λ-GA when compared in a wider context containing both NSGA-II and MACD, it retains a statistically significant edge over PSO and the two additional benchmarks. NSGA-II and MACD have performed worse than all other algorithms under various trend conditions in a statistically significant manner.

Trend	Algorithm	Ranking	p-value	Holm
Uptrend	λ-GA (control)	1.8	–	–
	λ-PSO^{SP}	1.8	1.0	0.05
	λ-PSO	2.4	0.39614	0.025
	NSGA-II	**4.05**	**0.00156**	**0.0167**
	MACD	**4.95**	**8.3982E-6**	**0.0125**
Sideways	λ-GA (control)	1.5	–	–
	λ-PSO^{SP}	2.1	0.3961	0.05
	λ-PSO	2.5	0.1573	0.025
	NSGA-II	**4.35**	**5.5656E-5**	**0.0167**
	MACD	**4.55**	**1.608E-5**	**0.0125**
Downtrend	λ-PSO^{SP} (control)	1.0	–	–
	λ-GA	2.0	0.1573	0.05
	λ-PSO	**3.2**	**0.0019**	**0.025**
	NSGA-II	**4.4**	**1.522E-6**	**0.0167**
	MACD	**4.4**	**1.522E-6**	**0.0125**
All	λ-PSO^{SP} (control)	1.3	–	–
	λ-GA	1.8	0.4795	0.05
	λ-PSO	**2.9**	**0.0237**	**0.025**
	NSGA-II	**4.2**	**4.1098E-5**	**0.0167**
	MACD	**4.8**	**7.431E-7**	**0.0125**

solutions generated even if just considering the single objective AROR. The rankings of the algorithms based on the number of wins via attaining maximum AROR per testing strand are as follows: λ-PSO^{SP} (13), λ-PSO (8), MACD (6), λ-GA (3), NSGA-II (1), PSO^{S} (0), GA (0) and PSO (0). We can see that the multiobjective optimization variants of PSO dominate the rest of the algorithms and hold the top two positions. This is followed by MACD, then the multiobjective variants of GA, although NSGA-II was only able to achieve a singular win. The single objective variants of our algorithms fared the worst, scoring no wins when compared with their multiobjective counterparts. This indicates that pursuing a multiobjective optimization approach improves the overall quality of the market timing strategies generated when compared to a single objective optimization approach.

In order to provide an overall picture comparing the performance of the multiobjective algorithms, relative to each other and to the benchmarks, we plot the solutions returned per financial metric and aggregated across all testing strands using violin plots [35]. A violin plot extends Tukey's Box and Whisker plots by displaying a kernel density estimation of the data points along with the summary statistics using a Gaussian kernel. The kernel density estimation is presented visually as the contours of the shape rendered for every category in the plot, while the summary statistics are represented by three lines rendered within the body of the shape. The dotted middle line represents the median, while the bottom and top dotted lines represent the first and third quartiles of the interquartile range. Figures 16 to 20 show the violin plots of all algorithms for AROR, Portfolio Risk, VaR, Transactions Count and Solution Length. From the AROR violin plot, Figure 16, we can observe that the medians and main bulk of the solution distributions for all algorithms are close to each other. The PSO algorithms display interesting behavior in having long tails extending well beyond those of their counterparts: λ-PSO^{SP} having a tail extending higher than rest showing the potential of achieving higher returns and λ-PSO extending significantly lower than its counterparts showing a higher potential of negative returns albeit at a lower probability. The Portfolio Risk violin plot, Figure 17, shows that all the algorithms in the comparison have the bulk of their distribution and medians around the same level. This is with the exception of MACD that has a higher interquartile range and a long tail stretching into higher values of risk, indicating a worse performance than the

other algorithms in the comparison. As for VaR, we can see from the associated plot, Figure 18, that all algorithms have their medians and the main bulk of their distributions close to each other. Only λ-PSO displays a long tail stretching into higher values of VaR indicating a higher potential for losses albeit at a low probability. Figure 19 shows the violin plot for transactions count and from that we can observe that the highest median was achieved by λ-PSO$^{\text{SP}}$, followed by NSGA-II, λ-PSO, λ-GA and MACD. All algorithms, bar MACD, have tails that extend higher into transaction counts. Based on the observations, we can surmise that λ-PSO$^{\text{SP}}$ returns the most stable solutions and MACD returns the least stable. Finally, Figure 19 shows the violin plot for solution length. By comparing λ-PSO to λ-PSO$^{\text{SP}}$, we can see that the pruning procedure has resulted in comparatively shorter solutions as evident by the lower interquartile range of λ-PSO$^{\text{SP}}$ and the manifestation of its main body of its solution distribution at a significantly lower level than λ-PSO. We can also observe that λ-PSO$^{\text{SP}}$ has achieved shorter solutions to its counterparts and is comparable to MACD which had a fixed solution length of one component.

As with the previous set of experiments, we reconducted the Friedman non-parametric test with the Holm correction on the mean hypervolumes to check if there are statistical significance differences. The results can be seen in Table 33. From Table 33, we can see that NSGA-II and MACD have performed worse than all other multiobjective optimization algorithms across all trend types. In particular, both NSGA-II and MACD performed statistically significantly worse than the control algorithm across all trend types. This suggests that our algorithms are better suited than NSGA-II and MACD when tackling market timing as a multiobjective optimization problem. We can see that λ-PSO$^{\text{SP}}$ and our multiobjective GA have consistently ranked in the top two across each trend type, and shows a slight edge in ranking when considering all trends, albeit in a statistically non-significant manner.

7.6 Diversity of the Pareto Front

In the previous section, we addressed the first limitation in the results obtained by our multiobjective optimization algorithms by comparing them with NSGA-II and

MACD as benchmarks. In this section, we address the second limitation which was lack of insight into the diversity of the solutions in the returned Pareto sets and how they are spread across their respective Pareto fronts. The more diversity in the solutions contained within a given Pareto set, the more spread they are across their respective Pareto front giving the end user more choice to select solutions that better suit their needs. In order to get insight into the diversity of the Pareto sets returned by the multiobjective optimization algorithms, we plot their results against each test strand using RadViz [36]. RadViz is a method of adapting a scatter plot to display multivariate data containing more than two dimensions. The various dimensions being plotted in a RadViz diagram are rendered as anchors distributed equally across the circumference of a circle containing points representing the dataset being visualized. Each point in the dataset is rendered as a point in this circle tethered to each of the dimensional anchors on the circumference on the plot area. The amount of tension in each tether is commensurate to the normalized value each point has to a given dimensional anchor and the points are rendered within the plot area, where they reach an equilibrium across all the tensions contained within the tethers. In our case, we have five dimensional anchors: AROR, Portfolio Risk, VaR, Transactions Count and Solution Length. As Portfolio Risk, VaR and Solutions Length are metrics that are minimized, we are interested in points that are situated far from the anchors associated with those metrics. The opposite is true for AROR and Transactions Count, where we are interested in points that are rendered close to those dimensional anchors. An example of a RadViz diagram and how we can interpret it can be seen in Figure 21. By observing the RadViz plots, we can visually identify which algorithms attained better diversity by looking at the spread of their corresponding Pareto sets. We use RadViz to plot the Pareto sets returned by λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for each testing strand, and the results can be seen in Appendix II.

From the RadViz plots, we can see that the λ-PSO$^{\text{SP}}$ algorithm consistently achieves the largest spread across the plot area, covering all areas covered by the other multiobjective optimization algorithms. This is despite the fact that no explicit measures where taken to promote diversity, such as the crowding distance measure used in NSGA-II. When present, MACD has consistently existed outside the main body occupied by the Pareto sets of the other algorithms, in an area

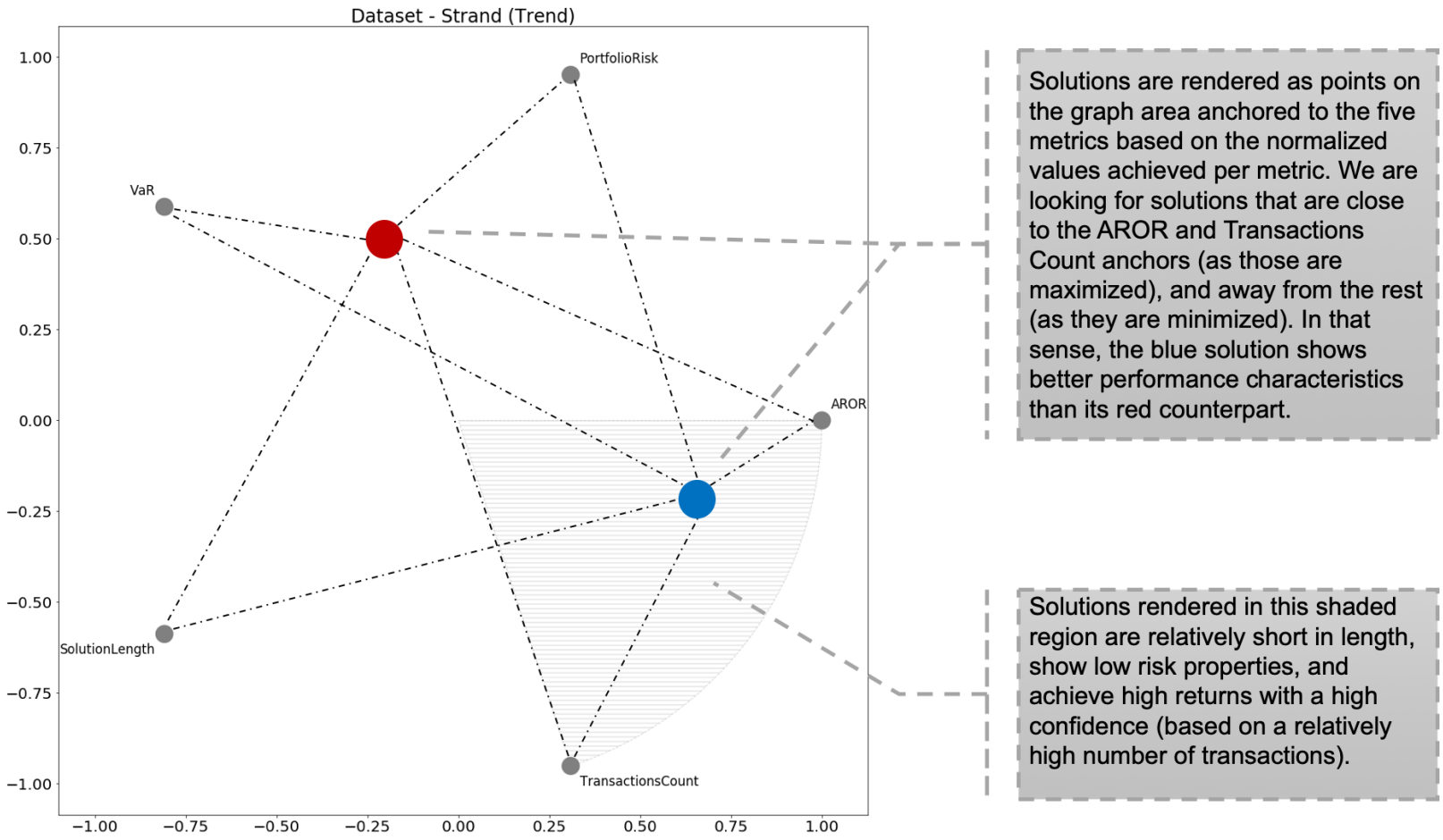

Figure 21: RadViz Interpretation

to the upper right of the plotting area. This implies that the solution presented by MACD can achieve good values of AROR but at the expense of high risk (attraction to the Portfolio Risk and VaR anchors) and low confidence (repulsion from the Transactions Count anchor). The NSGA-II Pareto sets are smaller in size when compared to the ones generated by the other algorithms, with the exception of BSX1 (↑), where the NSGA-II Pareto set always lies to the right of the PSO Pareto sets and to the left of both λ-GA and λ-PSO^{SP}. This implies that the λ-GA and λ-PSO^{SP} have a higher probability of landing in the desired quadrant of solutions with high profitability, low risk and high confidence – the quadrant indicated by gray shading in the plots. The algorithm that shows the least diversity is the λ-PSO algorithm, resulting in Pareto sets that are relatively tightly clustered near the center of the plotting area. Based on the RadViz visualizations of the algorithms and the observations seen in the results earlier in the chapter, we can observe that the multiobjective λ-PSO^{SP} algorithm returns Pareto sets with a high level of diversity and those Pareto sets contain competent solutions across all the metrics being considered. This is followed closely by λ-GA, then by λ-PSO, NSGA-II and MACD, respectively.

7.7 Summary

In this chapter, we have evolved from considering market timing as a single objective optimization problem to a multiobjective one. We started by increasing the number of financial metrics optimized from one to five to include: AROR, Portfolio Risk, Value at Risk (VaR), Transactions Count and Solution Length. We followed a Pareto dominance-based approach, and modified GA, PSO and PSO^{SP} to use the expanded set of metrics and archives to maintain sets of non-dominated solutions. This resulted in the multiobjective algorithms: λ-GA, λ-PSO and λ-PSO^{SP} respectively. Computational results showed that these algorithms achieved higher AROR values when compared to their single objective counter parts from the previous chapter and that λ-PSO^{SP} displayed a statistically significant improvement over λ-PSO and λ-GA in downtrends based on mean hypervolume. In order to provide a better comparative context for the algorithms, we compared

their performance against NSGA-II (a well established multiobjective optimization algorithm) and MACD (a widely used technical indicator in market timing applications). Results from the comparison showed that NSGA-II and MACD performed statistically significantly worse than λ-PSO, λ-GA and λ-PSO$^{\text{SP}}$ across all of the three trend types based on mean hypervolume. The Pareto sets returned by the algorithms for each testing strand was visualized using RadViz in order to gain insight about the diversity of those Pareto sets. The RadViz plots show that λ-PSO$^{\text{SP}}$ displayed the most diversity and its solutions were present in all areas of the plot that the Pareto sets of the other algorithms occupied. The least diverse algorithm was λ-PSO, consistently producing Pareto sets that occupied the center of the plotting area in one contiguous body.

In the next chapter, we conclude our work on PSO and market timing by summarizing all the work presented so far and suggest avenues for future research.

Chapter 8

Conclusions and Future Research

"There is no real ending. It's just the place where you stop the story."

–Frank Herbert

The research in this thesis presented novel Particle Swarm Optimization (PSO) and Genetic Algorithms (GA) for market timing. Market timing is the issue of deciding when to buy or sell a given asset on a financial market. A market timing strategy can be composed of a set of components that digest current market context and return a recommendation on the action to take. Each of the components returns a recommendation on an action to take multiplied by a weight, and the final action taken is based on an aggregate of the individual recommendations multiplied by their respective weights. Previous approaches on using computational intelligence to aid the formation of market timing strategies was vastly dominated by GA based on the volume of publications [40][84]. These approaches either attempted to optimize the parameters for a preset selection of components or select a subset from a set of components with predefined parameters. None of the surveyed approaches attempted to perform both functions simultaneously. This limits the designer of a market timing strategy to committing computational resources to one of those functions at a time, curtailing the flexibility in choice of components and parameters for consideration. When it came to training and testing, all of the surveyed approaches followed a protocol known as Step Forward testing. In Step Forward testing, a stream of financial data is split arbitrarily into two sections: the earlier section of the stream to be used for training while the latter was

used for testing. One of the main criticisms for the use of Step Forward testing is that algorithms have the liability of overfitting to only the trends observed in the data and thus suffering a significant degradation in performance when encountering unseen trends during live training. The majority of literature observed on the use of computational intelligence for market timing tackled it as a single objective optimization problem. When dealing with market timing strategies deployed in live trading scenarios, users of these strategies would gauge performance using a number of financial metrics that represent various aspects of profits, losses and exposure to risk. Modeling a market timing strategy as a single objective optimization problem limits its utility for live trading, as it constrains optimization to only one of the aforementioned aspects. Although a number of approaches modeled market timing as multiobjective optimization problems, such as [85, 16], they are limited in scope and pale in comparison to those that modeled it as a single objective optimization problem in terms of volume.

Our work in this thesis uses PSO and GA to tackle market timing in a novel fashion and addresses some of the limitations observed in literature. In particular, our contributions address the following challenges:

1. Optimizing both the selection of components and the tuning of their parameters by introducing a formulation that considers both in a simultaneous fashion.

2. Explicitly exposing the algorithms to a variety of trends both during training and testing using a novel protocol called Trend Representative Testing.

3. Tackling market timing as a multiobjective optimization problem to better accommodate the needs of the designers of algorithmic trading systems who would optimize various aspects of profits, losses and risks.

Our work also introduces a number of PSO variants that tackle market timing as a single objective optimization problem (PSO^{S}, PSO^{SR} and PSO^{SP}) and as a multiobjective optimization problem (λ-PSO^{S} and λ-PSO^{SP}). In the next two sections, we summarize our contributions using both PSO and GA, followed by suggestions for future research.

8.1 Contributions

8.1.1 Market Timing Strategies with PSO

Our first approach addressed the question: Can PSO be adopted to tackle market timing in such manner that considers both the selection of components and the tuning of their parameters simultaneously? In this approach, market timing is tackled as a single objective optimization problem, with the aim of maximizing a single metric of financial fitness: the Sharpe Ratio.

We started by devising a suitable encoding for a market timing strategy to be used by the particles in the swarm. As discussed earlier, a market timing strategy is composed by a set of components, each with an associated weight and a set of parameters. We decided to encode our market timing strategies as associative arrays, with the first level identifying the component type and the second associating a weight and parameter values with the given instance of that indicator. An example of this encoding can be seen in Chapter 5 (Figure 5). This encoding scheme is used to represent candidate solutions by all the algorithms presented in this thesis. We then adapted the basic PSO algorithm to use this encoding and tackle market timing, and that involved a number of modifications. The first modification was that the addition, subtraction and multiplication operators were overridden and implemented at the component level to allow for the velocity update function to be processed while being agnostic to the type of components involved. The second modification was adopting either a decreasing inertia schedule with velocity clamping or Clerc's constriction, the choice of which was left to the user. This is used to contain the particles within the search space and prevent their explosion beyond its bounds. With these modifications in place, the basic PSO algorithm can tackle market timing using our proposed formulation.

We also introduced two new PSO models: PSO^{S} and PSO^{SR}. PSO^{S} modifies the velocity update procedure, making the particles more reluctant to give up their current positions on the search landscape unless it is probabilistically beneficial to so. The reason behind adopting this procedure, known as Stochastic State Update, is that we want to decrease the tendency of particles to prematurely converging on a local optima. PSO^{SR} extends PSO^{S} by addressing the problem of finding the least

sufficing subset of components that maximize the financial metric being optimized. It does so by frequently polling the particles in the swarm for components whose weights have fallen below a specific value. Offending components are removed from all particles in the swarm. The frequency of pruning and the threshold under which pruning is executed are user defined parameters. The pruning algorithm for PSO^{SR} was presented in Chapter 5 (Algorithm 5).

In order to test our approach, we used used five variants of PSO including variations of basic PSO, PSO^{S} and PSO^{SR}. The swarm size for all variants was set at a 100 particles. Experiments were allowed to run for a 100 iterations, repeated 20 times to account for the effects of stochasticity. For training and testing, we used the Step Forward testing procedure on the data of four stocks with two years of daily prices for training and another year for testing. All PSO variants had access to six technical indicators to use for composing market timing strategies. Although the results showed that none of the variants were able to achieve good values for the Sharpe Ratio, the new introduced models did perform competitively with basic PSO. Nevertheless, the experiments did prove that selecting components and tuning their parameters in a simultaneous fashion is feasible, using PSO. Previous approaches in literature would only do one of the tasks at a time: either select components for the strategy with preset values for parameters or tune the parameters of a preset combination of components.

8.1.2 Trend Representative Testing

As mentioned earlier, one of the main issues identified in current literature is the tendency to overfit to training patterns while using Step Forward testing. We addressed this shortcoming by explicitly exposing candidate solutions to various trend types during both training and testing in a novel procedure we called Trend Representative Testing. The main impetus behind Trend Representative Testing is that through explicit exposure to upwards, downwards and sideways trends the possibility of niching towards one particular trend over the others is reduced and users will have better estimations of the performance of candidate market timing solutions under various market conditions. This is based on the recommendations of domain experts [44]. Our objectives with Trend Representative Testing were to

build a library of datasets that embody particular trends in price data and devise a methodology that utilizes this library in a manner to address the shortcomings of Step Forward Testing.

To build the library of trends, we started by acquiring the raw daily prices of all stocks traded on the Nasdaq and NYSE markets. The raw streams are then cleaned of price shocks, and the clean data is subsampled using sliding windows of various sizes to produce strands. The strands are then analyzed to identify the underlying trend and its intensity. These annotated strands are then stored in our library for use. In order to use these strands in training and testing, we formed two sets of triplets, where a triplet is composed of one uptrend, one sideways trend and one downtrend. During training, one triplet is selected at random from the set of training triplets, backtesting occurs against every constituent strand and the average performance reported. A similar process happens during the testing phase, but utilizing the triplets from the testing set.

Our first set of experiments with Trend Representative Testing compared it directly with Step Forward Testing. The algorithms used in this set of experiments included PSO, PSO^S and a Genetic Algorithm (GA) benchmark. The modifications required to the basic GA algorithm to allow it to use our encoding and Trend Representative Testing included redefining the crossover and mutation operators, the details of which can be seen in Chapter 6. As all algorithms have parameters, they went through hyperparameter optimization using the IRace algorithm [58]. For Trend Representative Testing, 30 strands were formed into 10 triplets, where one triplet is held out for testing and the remaining nine triplets are used in training. This results into 10 distinct training and testing datasets. All algorithms had access to 15 technical indicators to use as components in the market timing strategies. For Step Forward testing, we considered each of the strands as the 30% forward testing data and obtain the preceding 70% of data required per strand from the associated raw streams to form the training data. The Annualized Rate of Returns (AROR) was selected as the metric to be optimized, in this case maximized as it is a measure of profits. Each of the experiments was repeated 10 times to cater for the effects of stochasticity. Based on the Friedman non-parametric test of the results, both Step Forward Testing and Trend Representative Testing showed to be statistically equivalent based on mean fitness, which makes Trend

Representative Testing a viable alternative to Step Forward Testing. The advantage Trend Representative Testing has is that it is explicit in exposing candidate solutions to a multitude of trends, evaluating solutions' performance under various market conditions.

Our second set of experiments expanded the number of indicators available to the algorithms from 15 to 63. As the algorithms have parameters, they all went through hyperparameter optimization using IRace. During hyperparameter optimization, the pruning procedure of PSO^{SR} was found to be ineffective, and the algorithm was excluded from experimentation. The final set of algorithms tested was GA, PSO and PSO^{S}. A new set of 10 triplets were used to form the training and testing datasets. Looking at the results, no statistically significant differences were observed among the algorithms according to the Friedman non-parametric test of the mean fitness. This makes the PSO algorithms competitive to GA, the current market incumbent algorithm. Since PSO^{SR} was excluded from experimentation, we devised a new pruning procedure that is not as aggressive. The new pruning procedure only sets the weight of components market for pruning to zero instead of removing it from all particles in the swarm. This allows their reintroduction into the candidate solutions through a subsequent velocity update where it is found in an effective configuration. The PSO variant using this new pruning procedure with Stochastic State Update is labeled as PSO^{SP}. Comparing the performance of PSO^{SP} with the other algorithms, we observed that it is statistically equivalent, with the advantage that it was able to prune components arriving at shorter market timing strategies that are quicker to execute and easier to comprehend.

8.1.3 Multiobjective Optimization

Considering market timing as a single objective optimization is not sufficient in producing market timing strategies that are suitable for live trading. Algorithmic trading systems need to consider multiple aspects of potential profits, losses and exposure to risk. Our next step, therefore, was to consider market timing as a multiobjective optimization problem. In order to do so, we expanded the set of metrics optimized to include: Annualized Rate of Return (AROR), Annualized Portfolio

Risk, Value at Risk (VaR), Transactions Count and Solution length. We used a Pareto dominance based approach for multiobjective optimization and modified the PSO, PSO^{SP} and GA algorithms to adopt this approach using the five metrics to be optimized. This resulted in the multiobjective algorithms: λ-GA, λ-PSO and λ-PSO^{SP} respectively[1]. General modifications to all algorithms included the use of an archive to keep track of non-dominated solutions as they are discovered by the algorithms. At the end of an algorithm's run, the contents of this archive is returned as the discovered Pareto set. For λ-GA, the tournament selection operator was modified to return a random non-dominated member of a subset of the population, also selected at random. For the PSO family of algorithms, each particle maintains a personal non-dominated archive to keep track of personal bests. A personal best is selected at random from a particle personal archive during velocity update. As for selecting the neighborhood best, a non-dominated candidate is selected at random from the a particle's current set of non-dominated neighbors as well as candidates from their personal archives. The Stochastic State Update mechanism is updated to use the dominance score (a measure of the number of objectives where one solution dominates another) in its formulation.

The algorithms are trained and tested using Trend Representative Testing, using the same dataset used Chapter 6 in order to allow for comparison. As all the algorithms have parameters, they first go through hyperparameter optimization using IRace. Comparing the AROR results of the multiobjective variants of the algorithms with their single objective counterparts, we can see that the multiobjective variants achieved better results in the majority of cases. Looking at the mean hypervolumes achieved by the multiobjective optimization algorithms, we can see that λ-PSO^{SP} attained a statistically significant edge over the other algorithms in downtrends. We can also see that λ-PSO^{SP} achieved the best results per objective when compared to the other algorithms.

Two primary limitations are present in the results of the multiobjective optimization variants of our algorithms. Firstly, the results are all from algorithms

[1]Both λ-PSO^{S} and λ-PSO^{SP} originate from the same algorithm, with the emergent variant based on setting of the Pruning parameter. If Pruning is enabled, then the emergent variant is λ-PSO^{SP}, otherwise the emergent variant is λ-PSO^{S}.

introduced in this thesis and we have no comparison with an established multiobjective optimization algorithm or a widely used market timing technique. Secondly, we have no insight into the diversity of the Pareto sets returned by the algorithms. In order to address the first limitation, we compare the performance of our algorithms against NSGA-II and the MACD technical indicator. The results from the comparison show that NSGA-II and MACD performed statistically significantly worse than all our multiobjective optimization algorithms. As for the second limitation, we use RadViz [36] to visualize the Pareto sets returned by all the algorithms, including NSGA-II and MACD. The RadViz plots show that λ-PSO$^{\text{SP}}$ had the best spread across all strands, despite not having any explicit diversity promoting measures (such as the crowding distance mechanism employed by NSGA-II).

8.2 Suggestions for Future Research

The research presented on the application of PSO to tackling market timing can be extended in two ways: approaches where we extend the capabilities of the algorithms presented thus far, or approaches where we expand the scope of how market timing is tackled. Although the suggestions presented here were made with PSO in mind, the vast majority of them can also be applied to GA algorithms. Suggestions for future research under both avenues are presented in the following subsections.

8.2.1 Extending the Capabilities of the Current Algorithms

One of the most direct extensions to the work presented in this thesis is to expand on the set of financial metrics being optimized. This can grow to include more sophisticated measures of profit and risk, and more accurate simulation of slippage and transaction costs. As the scope of optimization increases, the contention between the different objectives will increase making it more difficult to locate usable areas on the Pareto fronts.

Another extension in regards to the scope of optimization is to consider diversity as a first class citizen when it comes to optimization. Although λ-PSO$^{\text{SP}}$

showed a fairly good level of diversity in its Pareto sets without implementing any explicit measures that improve diversity, such as the crowding distance measure used by the NSGA family of algorithms [26, 25], it will be worthwhile in evaluating the effect of applying measures that promote diversity to the PSO algorithms presented in this thesis. The value of such an extension could be that the Pareto sets generated by PSO algorithms will have a more controlled form of spread across the objective landscape, giving the users greater flexibility in choosing solutions that best suit their needs.

A third possible extension to the scope of optimization is to optimize the set of financial metrics for every trend type. Although this would be considered niching, it would provide the user with an increased probability of achieving their financial goals. This however comes at the cost of increasing the complexity of the overall algorithmic trading system as it entails the use of three separate market timing strategies that are activated based on the current trend detected in the market.

A final suggestion to extend the capabilities of the current algorithms would be to include fundamental analysis components into the set of signal generating components available to the algorithms [74]. This can include considering information regarding profits and earnings reports, information regarding mergers and acquisitions, sales performance history and even current and previous sentiment regarding the asset in question. The challenge here would be to align the data required by the new components with the price data used by the current technical analysis indicators.

8.2.2 Expanding the Scope of How Market Timing is Tackled

A natural extension that would expand the scope of our work is to adapt other algorithms to use the formulation presented in Chapter 2 and Trend Representative Testing, presented in Chapter 6. The application of other algorithms would provide a more comprehensive comparative context for the performance of PSO. It would also present the opportunity to use Meta-learning to see if particular algorithms excel at producing competent market timing strategies under certain conditions [13]. In this scenario, a user will adopt an ensemble approach [75], where a number

of algorithms would be used in accordance to the current circumstances detected to produce candidate market timing strategies.

Another approach for expanding the scope of our work would be to adapt the algorithms presented in this thesis to include other issues faced by designers of algorithmic trading systems into consideration. As aforementioned in Chapter 1, a designer of algorithmic trading systems is faced by a number of issues: objective definition, portfolio optimization, market timing and execution optimization [44]. Though objectives are explicitly set by a user, the other aspects of the system can be considered as optimization problems and we have only attended to one of them in this thesis: market timing. By including portfolio optimization and execution optimization alongside market timing, we will be closer to building algorithms that in essence generate complete algorithmic trading systems, and not just strategies to tackle singular aspects of trading.

A final suggestion for an expansion of scope is dynamic optimization [30]. Over time, candidate solutions selected for live trading will drift from areas of optimal performance and suffer from degradation. This can be in reaction to changes in the domain of optimization either by internal or external factors. To stay capable of achieving a user's goal, a trading system would have to be able to detect these changes and adapt. Dynamic optimization is concerned with a system's capacity of detecting drift from optimality at time t, find new regions of optimality and track its trajectory over time. By considering the issue of dynamic optimization, either on the micro scope of market timing or macro scope of algorithmic trading, our systems would be more suited for live trading by seamlessly adapting to ever changing market conditions and bringing us closer to autonomous trading systems that require no user intervention.

Bibliography

[1] Abdelbar, A. M. and Abdelshahid, S. (2003). Swarm optimization with instinct-driven particles. In *Congress on Evolutionary Computation, CEC 2003 - Proceedings*, vol. 2, pp. 777–782.

[2] Abido, M. (2009). Multiobjective particle swarm optimization for environmental/economic dispatch problem. *Electric Power Systems Research*, 79(7), pp. 1105 – 1113.

[3] Al-kazemi, B. and Mohan, C. (2002). Multi-phase generalization of the particle swarm optimization algorithm. In *Proceedings of the 2002 Congress on Evolutionary Computation. CEC'02 (Cat. No.02TH8600)*, vol. 1, pp. 489–494.

[4] Allen, F. and Karjalainen, R. (1999). Using genetic algorithms to find technical trading rules. *Journal of Financial Economics*, 51(2), pp. 245–271.

[5] Angeline, P. (1998). Using selection to improve particle swarm optimization. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pp. 84–89.

[6] Armano, G. and Farmani, M. R. (2016). Multiobjective clustering analysis using particle swarm optimization. *Expert Systems with Applications*, 55, pp. 184 – 193.

[7] Bengoetxea, E. and Larranaga, P. (2010). EDA-PSO: A hybrid paradigm combining estimation of distribution algorithms and particle swarm optimization.

Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), 6234 LNCS, pp. 416–423.

[8] Bera, A., Sychel, D. and Sacharski, B. (2014). Improved Particle Swarm Optimization method for investment strategies parameters computing. *Journal of Theoretical and Applied Computer Science*, 8(4), pp. 45–55.

[9] Beume, N. et al. (2009). On the complexity of computing the hypervolume indicator. *IEEE Transactions on Evolutionary Computation*, 13(5), pp. 1075–1082.

[10] Bianchi, L. et al. (2009). A survey on metaheuristics for stochastic combinatorial optimization. *Natural Computing*, 8(2), pp. 239–287.

[11] Blackwell, T. M. and Bentley, P. (2002). Don't push me! Collision-avoiding swarms. In *Proceedings of the 2002 Congress on Evolutionary Computation, CEC 2002*, vol. 2, pp. 1691–1696.

[12] Brabazon, A. and O'Neill, M. (2004). Evolving technical trading rules for spot foreign-exchange markets using grammatical evolution. *Computational Management Science*, 1(3), pp. 311–327.

[13] Brazdil, P. et al. (2008). *Metalearning: Applications to Data Mining*. Springer Publishing Company, Incorporated, 1st edn.

[14] Brits, R. (2002). *Niching Strategies for Particle Swarm Optimization*. Ph.D. thesis, University of Pretoria.

[15] Brits, R., Engelbrecht, A. and Bergh, F. V. D. (2002). A niching particle swarm optimizer. In *Proceedings of the 4th Asia- ...*, vol. 2, pp. 1–5.

[16] Briza, A. C. and Naval Jr., P. C. (2011). 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), pp. 1191–1201.

[17] C.A. Coello Coello, E. L. and Aguirre, A. (2002). Mopso: A proposal for multiple objective particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 2, pp. 1051–1056.

[18] Cai, Q. et al. (2013). A novel stock forecasting model based on fuzzy time series and genetic algorithm. *Procedia Computer Science*, 18, pp. 1155 – 1162, 2013 International Conference on Computational Science.

[19] Carlisle, A. and Dozier, G. (2000). Adapting particle swarm optimization to dynamic environments. In *Proc. of the International Conference on Artificial Intelligence*.

[20] Chakravarty, S. and Dash, P. K. (2012). A PSO based integrated functional link net and interval type-2 fuzzy logic system for predicting stock market indices. *Applied Soft Computing*, 12(2), pp. 931–941.

[21] Chen, S.-M. and Kao, P.-Y. (2013). TAIEX forecasting based on fuzzy time series, particle swarm optimization techniques and support vector machines. *Information Sciences*, 247, pp. 62–71.

[22] Clerc, M. (1999). The swarm and the queen: towards a deterministic and adaptive particle swarm optimization. In *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99 (Cat. No. 99TH8406)*, vol. 3, pp. 1951–1957.

[23] Clerc, M. (2002). Think locally act locally-a framework for adaptive particle swarm optimizers. *IEEE Journal of Evolutionary Computation*, 29, pp. 1951–1957.

[24] de la Fuente, D. et al. (2006). Genetic algorithms to optimise the time to make stock market investment. In *Genetic and Evolutionary Computation Conference*, pp. 1857–1858.

[25] Deb, K. and Jain, H. (2014). An Evolutionary Many-Objective Optimization Algorithm Using Reference-Point-Based Nondominated Sorting Approach, Part I: Solving Problems With Box Constraints. *IEEE Transactions on Evolutionary Computation*, 18(4), pp. 577–601.

[26] Deb, K. et al. (2002). A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Transactions on Evolutionary Computation*, 6(2), pp. 182–197.

[27] Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *J Mach Learn Res*, 7, pp. 1–30.

[28] Eberhart, R. and Kennedy, J. (1995). A new optimizer using particle swarm theory. *MHS'95 Proceedings of the Sixth International Symposium on Micro Machine and Human Science*, pp. 39–43.

[29] Elbeltagi, E., Hegazy, T. and Grierson, D. (2005). Comparison among five evolutionary-based optimization algorithms. *Advanced Engineering Informatics*, 19(1), pp. 43 – 53.

[30] Engelbrecht, A. P. (2005). *Fundamentals of Computational Swarm Intelligence*. John Wiley & Sons Ltd.

[31] Fieldsend, J. E. and Singh, S. (2002). A multiobjective algorithm based upon particle swarm optimization, an efficient data structure and turbulence. In *Proceedings of the 2002 U.K. Workshop on Computational Intelligence*, pp. 37–44.

[32] Fonseca, C. M., Paquete, L. and López-Ibáñez, M. (2006). An improved dimension-sweep algorithm for the hypervolume indicator. In *Proceedings of the 2006 Congress on Evolutionary Computation (CEC 2006)*, Piscataway, NJ: IEEE Press, pp. 1157–1163.

[33] García, S. et al. (2010). 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), pp. 2044–2064.

[34] Glover, F. and Sörensen, K. (2015). Metaheuristics. *Scholarpedia*, 10, p. 6532.

[35] Hintze, J. L. and Nelson, R. D. (1998). Violin plots: A box plot-density trace synergism. *The American Statistician*, 52(2), pp. 181–184.

[36] Hoffman, P., Grinstein, G. and Pinkney, D. (1999). Dimensional anchors: A graphic primitive for multidimensional multivariate information visualizations. In *Proceedings of the 1999 Workshop on New Paradigms in Information Visualization and Manipulation in Conjunction with the Eighth ACM Internation Conference on Information and Knowledge Management*, New York, NY, USA: Association for Computing Machinery, NPIVM '99, p. 9–16.

[37] Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. Ann Arbor, MI: University of Michigan Press, second edition, 1992.

[38] Hoos, H. H. and Stützle, T. (2005). *Stochastic Local Search Foundations And Applications*. Morgan Kaufmann.

[39] Hu, X. and Eberhart, R. (2002). Multiobjective Optimization using Dynamic Neighborhood Particle Swarm Optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 2, pp. 1677–1681.

[40] Hu, Y. et al. (2015). Application of evolutionary computation for rule discovery in stock algorithmic trading: A literature review. *Applied Soft Computing*, 36, pp. 534–551.

[41] Kampouridis, M. and Otero, F. E. (2017). Evolving trading strategies using directional changes. *Expert Systems with Applications*, 73, pp. 145–160.

[42] Karathanasopoulos, A., Dunis, C. and Khalil, S. (2016). Modelling, forecasting and trading with a new sliding window approach: the crack spread example. *Quantitative Finance*, 7688(September), pp. 1–12.

[43] Kassabalidis, I. N. et al. (2002). Dynamic security border identification using enhanced particle swarm optimization. In *IEEE Transactions on Power Systems*, vol. 17, pp. 723–729.

[44] Kaufman, P. J. (2013). *Trading Systems and Methods*. John Wiley & Sons, Inc, 5th edn.

[45] Kennedy, J. (1997). The particle swarm: social adaptation of knowledge. In *Proceedings of 1997 IEEE International Conference on Evolutionary Computation (ICEC '97)*, pp. 303–308.

[46] Kennedy, J. (2003). Neighborhood Topologies in Fully-Informed and Best-Of-Neighborhood Particle Swarms. In *Proceedings of IEEE International Workshop on Soft Computing in Industrial Applications*, pp. 45–50.

[47] Kennedy, J. and Eberhart, R. (1995). Particle swarm optimization. In *Neural Networks, 1995. Proceedings.*, pp. 1942–1948, `arXiv:1011.1669v3`.

[48] Kennedy, J. and Eberhart, R. C. (2001). *Swarm Intelligence*. Morgan Kaufmann.

[49] Kennedy, J. and Spears, W. M. (1998). Matching algorithms to problems: an experimental test of the particle swarm and some genetic algorithms on the multimodal problem generator. In *1998 IEEE International Conference on Evolutionary Computation Proceedings. IEEE World Congress on Computational Intelligence (Cat. No.98TH8360)*, pp. 78–83.

[50] Kim, Y. et al. (2017). An intelligent hybrid trading system for discovering trading rules for the futures market using rough sets and genetic algorithms. *Applied Soft Computing*, 55, pp. 127–140.

[51] Koay, C. A. and Srinivasan, D. (2003). Particle swarm optimization-based approach for generator maintenance scheduling. In *2003 IEEE Swarm Intelligence Symposium, SIS 2003 - Proceedings*, pp. 167–173.

[52] Konstantinos E. Parsopoulos, M. N. V. and Tasoulis, D. (2004). Multiobjective Optimization using Parallel Vector Evaluated Particle Swarm Optimization. In *Proceedings of the IASTED International Conference on Artificial Intelligence and Applications*, vol. 2, ACTA Press, pp. 823–828.

[53] Koza, J. R. (1995). Survey of genetic algorithms and genetic programming. In *In In Proceedings of the Wescon 95 - Conference Record: Microelectronics, Communications Technology, Producing Quality Products, Mobile and Portable Power, Emerging Technologies*, IEEE Press, pp. 589–594.

[54] L. Zhang, Z. M. M. M., C. Liu and Liang, Y. (2003). Solving multi objective optimization problems using particle swarm optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, vol. 4, pp. 2400–2405.

[55] Ladyzynski, P. and Grzegorzewski, P. (2013). Particle swarm intelligence tunning of fuzzy geometric protoforms for price patterns recognition and stock trading. *Expert Systems with Applications*, 40(7), pp. 2391–2397.

[56] Liu, C. F., Yeh, C. Y. and Lee, S. J. (2012). Application of type-2 neuro-fuzzy modeling in stock price prediction. *Applied Soft Computing*, 12(4), pp. 1348–1358.

[57] Liu, X. et al. (2019). Coevolutionary particle swarm optimization with bottleneck objective learning strategy for many-objective optimization. *IEEE Transactions on Evolutionary Computation*, 23(4), pp. 587–602.

[58] López-Ibáñez, M. et al. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3, pp. 43–58.

[59] M., B. T. and Bentley, P. J. (2002). Dynamic search with charged swarms. In *Proceedings of the genetic and evolutionary*, p. 8.

[60] Mendes, R., Kennedy, J. and Neves, J. (2003). Watch thy neighbor or how the swarm can learn from its environment. In *Proceedings of IEEE Swarm Intelligence Symposium*, pp. 88–94.

[61] Meza, J. et al. (2017). Movpso: Vortex multi-objective particle swarm optimization. *Applied Soft Computing*, 52, pp. 1042 – 1057.

[62] Mohamed, I. and Otero, F. E. B. (2018). 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*, Springer International Publishing, pp. 435–436.

[63] Moore, J. and Chapman, R. (1999). Application of particle swarm to multiobjective optimization. Tech. rep., Department of Computer Science and Software Engineering, Auburn University.

[64] Moraglio, A. et al. (2008). Geometric Particle Swarm Optimization. *Journal of Artificial Evolution and Applications*, 2008, pp. 1–14.

[65] Mostaghim, S. and Tiech, J. (2003). Strategies for finding local guides in multi-objective particle swarm optimization (mopso). In *Proceedings of the IEEE Swarm Intelligence Symposium*, pp. 26–33.

[66] Mousa, A., El-Shorbagy, M. and Abd-El-Wahed, W. (2012). Local search based hybrid particle swarm optimization algorithm for multiobjective optimization. *Swarm and Evolutionary Computation*, 3, pp. 1 – 14.

[67] Murphy, J. (1999). *Technical Analysis of the Financial Markets: A Comprehensive Guide to Trading Methods and Applications*. Penguin.

[68] Nguyen, T. H., Shirai, K. and Velcin, J. (2015). Sentiment analysis on social media for stock movement prediction. *Expert Systems with Applications*, 42(24), pp. 9603 – 9611.

[69] Pan, A. et al. (2018). A diversity enhanced multiobjective particle swarm optimization. *Information Sciences*, 436-437, pp. 441 – 465.

[70] Parsopoulos, K. E. (2001). Stretching technique for obtaining global minimizers through Particle Swarm Optimization. In *In Proceedings of The IEEE Workshop on Particle Swarm Optimization*, 1, pp. 22–29.

[71] Parsopoulos, K. E. and Varhatis, M. N. (2002). Particle Swarm Optimization Method in Multiobjective Problems. In *Proceedings of the ACM Symposium on Applied Computing*, pp. 603–607.

[72] Parsopoulos, K. E. and Vrahatis, M. N. (2001). Modification of the Particle Swarm Optimizer for locating all the global minima. In *In Proceedings of The International Conference on Artificial Neural Networks and Genetic Algorithm*, pp. 324–327.

[73] Patterson, S. (2013). *Dark Pools: The Rise of A.I. Trading Machines and the Looming Threat to Wall Street*. Random House Business Books.

[74] Penman, S. H. (2013). *Financial Statement Analysis and Security Valuation*. McGraw-Hill.

[75] Polikar, R. (2012). *Ensemble Learning*, Boston, MA: Springer US. pp. 1–34.

[76] Pring, M. (2002). *Technical Analysis Explained*. McGraw-Hill.

[77] Rahimi, S., Abdollahpouri, A. and Moradi, P. (2018). A multi-objective particle swarm optimization algorithm for community detection in complex networks. *Swarm and Evolutionary Computation*, 39, pp. 297 – 309.

[78] Raquel, C. R. and Naval, P. C. (2005). An effective use of crowding distance in multiobjective particle swarm optimization. In *Proceedings of the 7th Annual Conference on Genetic and Evolutionary Computation*, New York, NY, USA: Association for Computing Machinery, GECCO '05, p. 257–264.

[79] Reyes-Sierra, M. and Coello, C. C. (2006). Multi-objective particle swarm optimizers: A survey of the state-of-the-art. *International Journal of Computaional Intelligence Research*, 2(3), pp. 1–18.

[80] Scheepers, C., Engelbrecht, A. P. and Cleghorn, C. W. (2019). Multi-guide particle swarm optimization for multi-objective optimization: empirical and stability analysis. *Swarm Intelligence*, 13(3), pp. 245–276.

[81] Sharpe, W. F. (1975). Adjusting for risk in portfolio performance measurement. *The Journal of Portfolio Management*, 1(2), pp. 29–34.

[82] Shi, Y. and Eberhart, R. (1998). A Modified Particle Swarm Optimizer. In *The 1998 IEEE International Conferences on Evolutionary Computation*, pp. 69–73.

[83] Shi, Y. and Eberhart, R. (1998). Parameter selection in particle swarm optimization. In *Evolutionary Programming VII, 7th International Conference*, pp. 591–600.

[84] Soler-Dominguez, A., Juan, A. A. and Kizys, R. (2017). A Survey on Financial Applications of Metaheuristics. *ACM Computing Surveys*, 50(1), pp. 1–23.

[85] Subramanian, H. et al. (2006). Designing Safe, Profitable Automated Stock Trading Agents Using Evolutionary Algorithms. In *Genetic and Evolutionary Computation Conference*, vol. 2, p. 1777.

[86] Sun, Y. and Gao, Y. (2015). An Improved Hybrid Algorithm Based on PSO and BP for Stock Price Forecasting. *The Open Cybernetics & Systemics Journal*.

[87] Togelius, J., De Nardi, R. and Moraglio, A. (2008). Geometric PSO + GP = particle swarm programming. *2008 IEEE Congress on Evolutionary Computation, CEC 2008*, pp. 3594–3600.

[88] U. Baumgartner, C. Magele and W. Renhart (2004). Pareto optimality and particle swarm optimization. *IEEE Transactions on Magnetics*, pp. 1172–1175.

[89] van den Bergh, F. (2001). *An analysis of particle swarm optimizers*. Ph.D. thesis, University of Petoria.

[90] Wang, F., Yu, P. L. and Cheung, D. W. (2014). Combining technical trading rules using particle swarm optimization. *Expert Systems with Applications*, 41(6), pp. 3016–3026.

[91] X. Hu, R. E. and Shi, Y. (2003). Particle Swarm with Extended Memory for Multiobjective Optimization. In *Proceedings of the IEEE swarm Intelligence Symposium*, pp. 193–197.

[92] Yen, G. and Lu, H. (2003). Dynamic population strategy assisted particle swarm optimization. In *Proceedings of the IEEE International Symposium on Intelligent Control*, pp. 697 – 702.

[93] Yu, Y., Duan, W. and Cao, Q. (2013). The impact of social and conventional media on firm equity value: A sentiment analysis approach. *Decision Support Systems*, 55(4), pp. 919 – 926.

[94] Zhang, Y. and Huang, S. (2003). Multiobjective optimization using distance-based particle swarm optimization. In *Proceedings of the International Conference on Computational Intelligence, Robotics and Autonomous Systems*.

[95] Zhou, Y., Wang, J. and Road, X. W. (2007). A Discrete Estimation of Distribution Particle Swarm Optimization for Combinatorial Optimization Problems. In *Third International Conference on Natural Computation*, IEEE Computer Society.

Appendix I: Box Plots for Step Forward vs Trend Representative Testing Experiments

In this appendix, we present the boxplot diagrams of the results obtained while comparing Step Forward Testing to Trend Representative Testing. Each plot represents the results obtained per testing strand, which is visible at the top of each plot. The algorithms and testing schemes used are listed across the x-axis, while fitness is displayed across the y-axis.

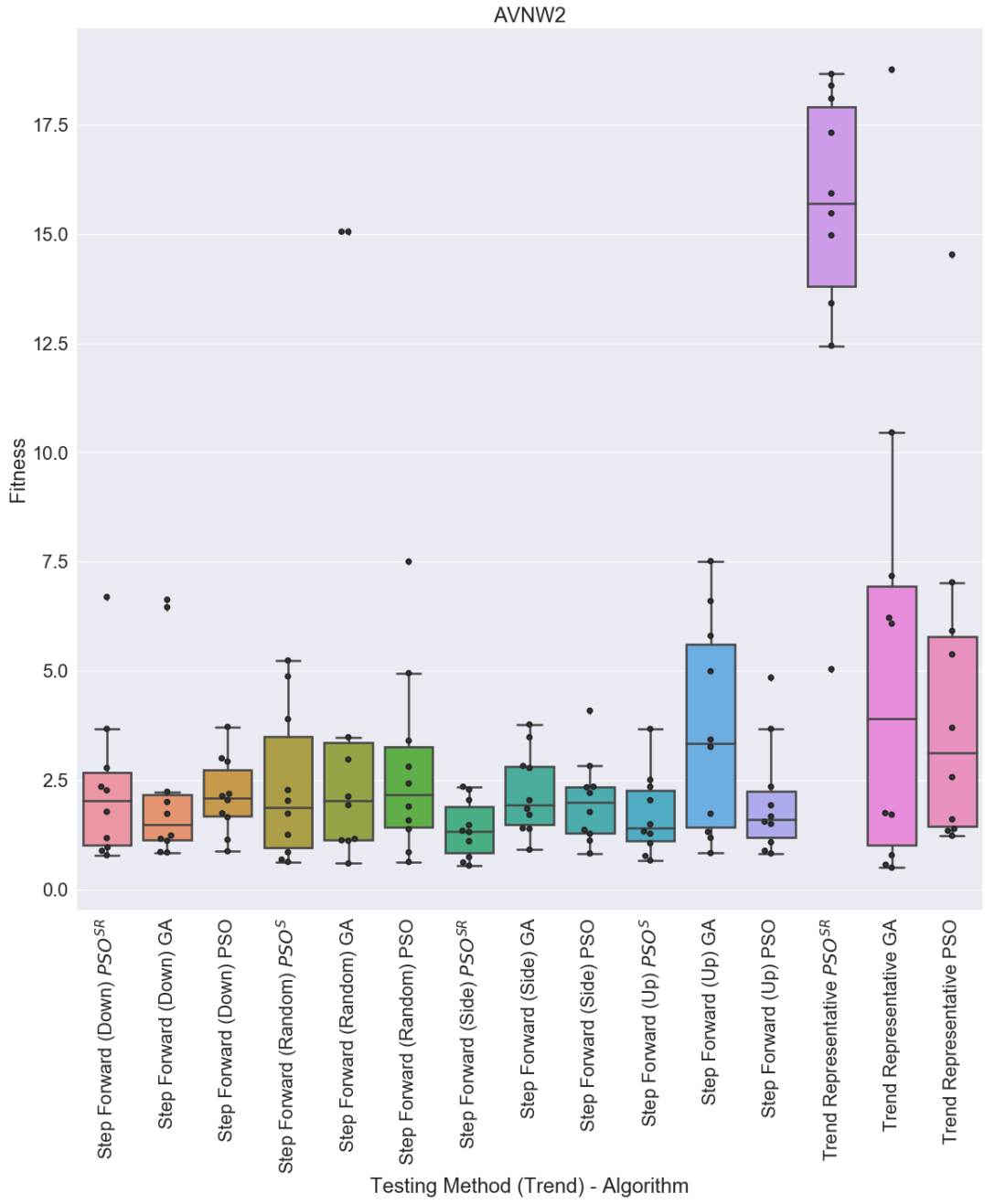

Figure 22: Algorithm box plots for AVNW2

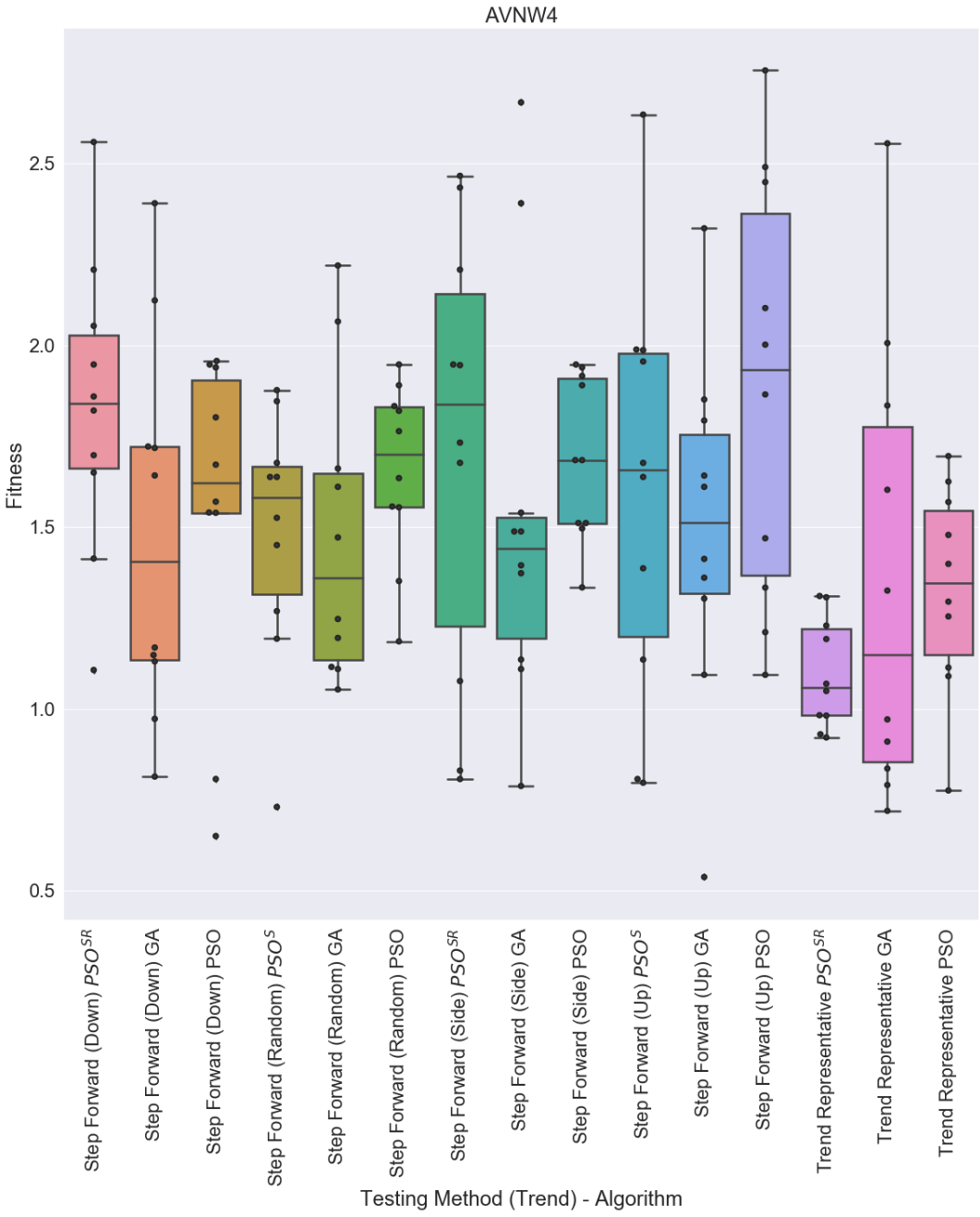

Figure 23: Algorithm box plots for AVNW4

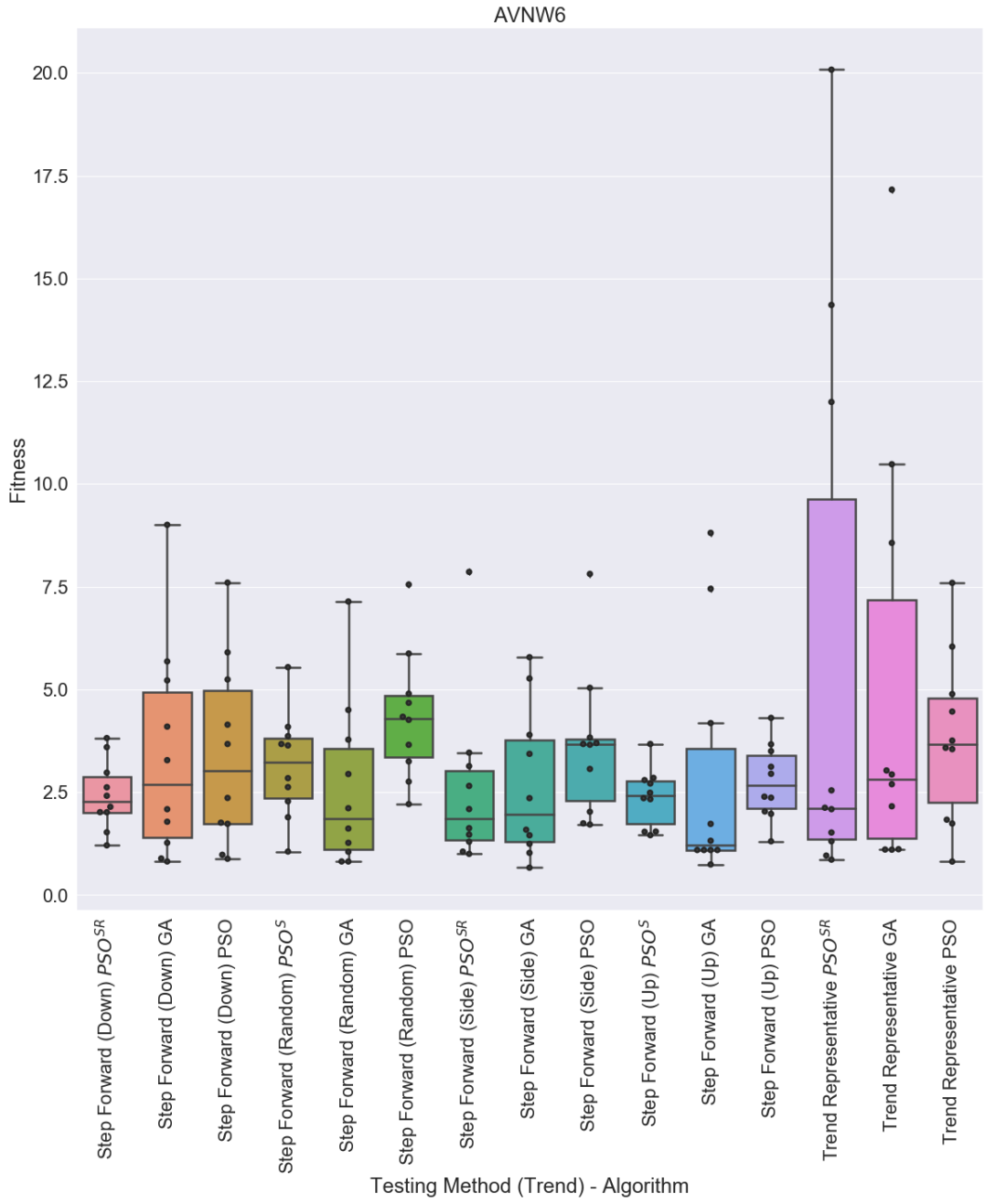

Figure 24: Algorithm box plots for AVNW6

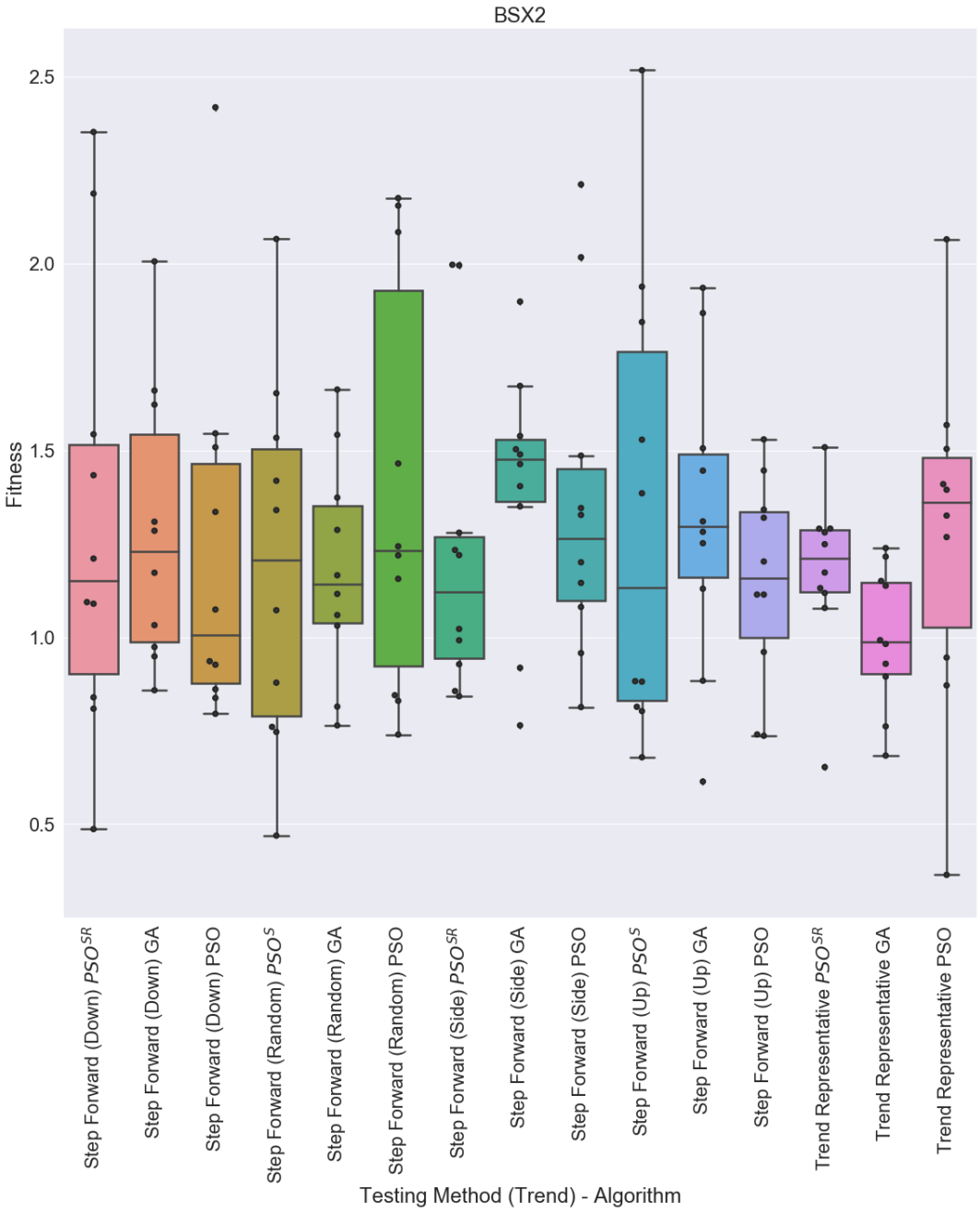

Figure 25: Algorithm box plots for BSX2

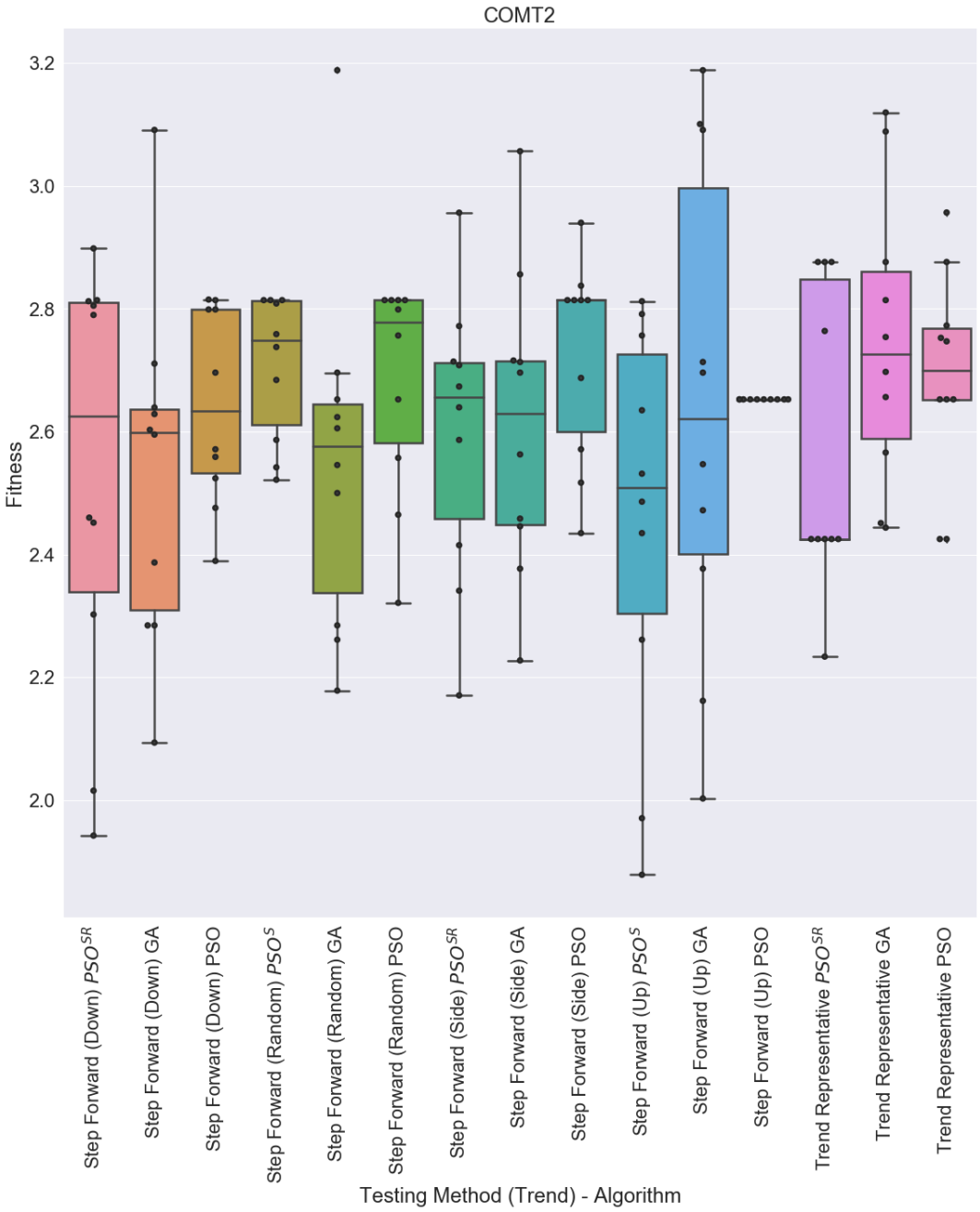

Figure 26: Algorithm box plots for COMT2

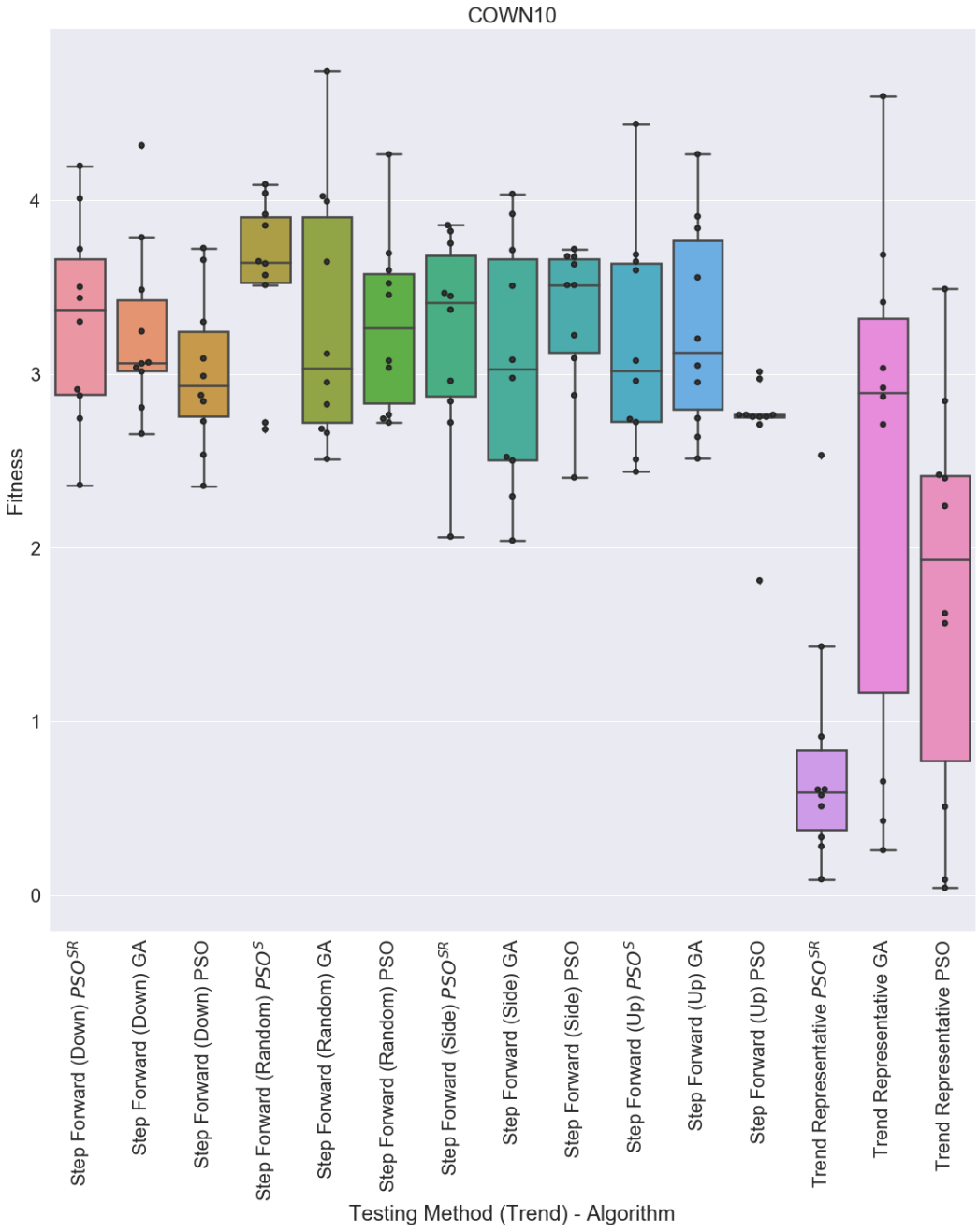

Figure 27: Algorithm box plots for COWN10

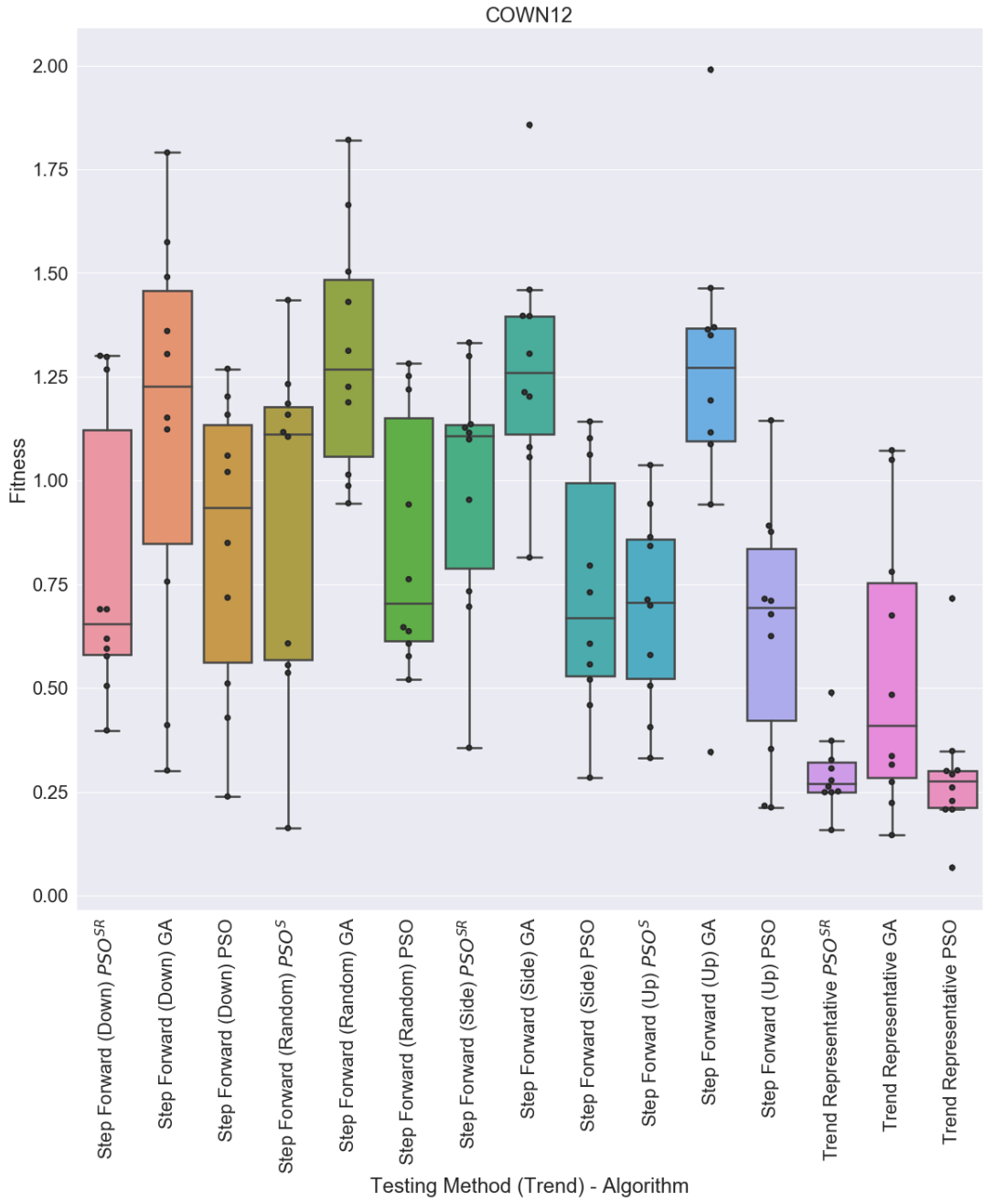

Figure 28: Algorithm box plots for COWN12

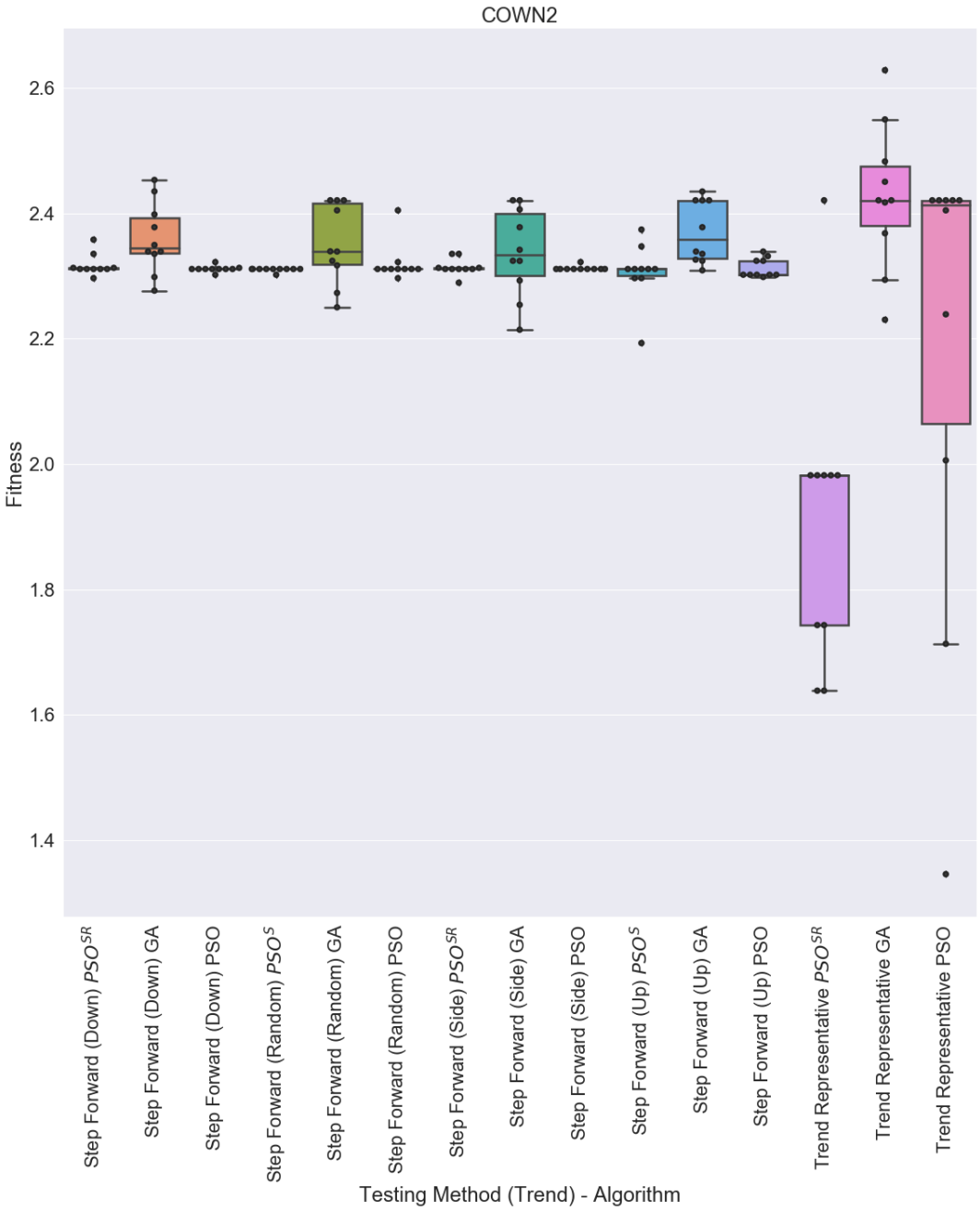

Figure 29: Algorithm box plots for COWN2

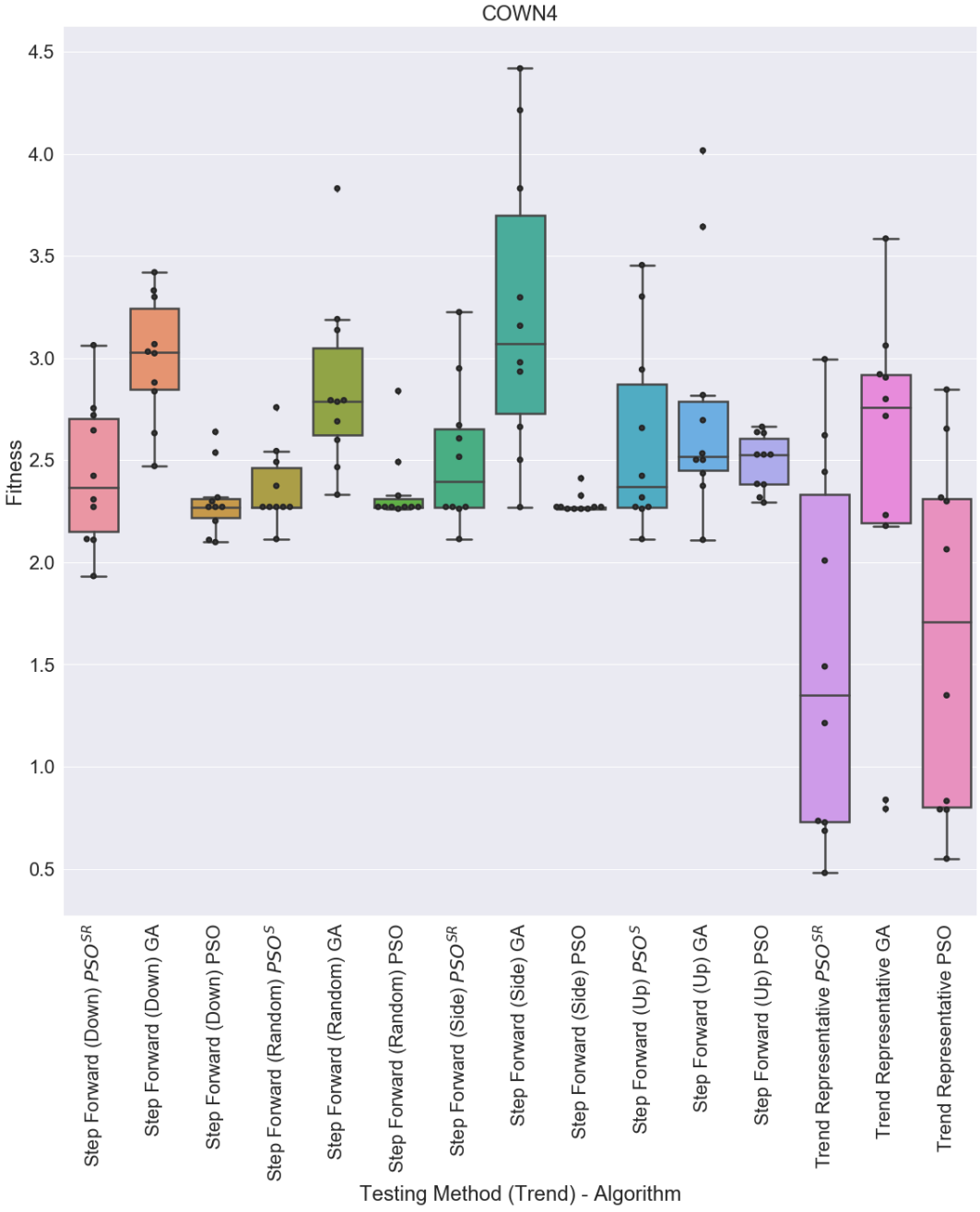

Figure 30: Algorithm box plots for COWN4

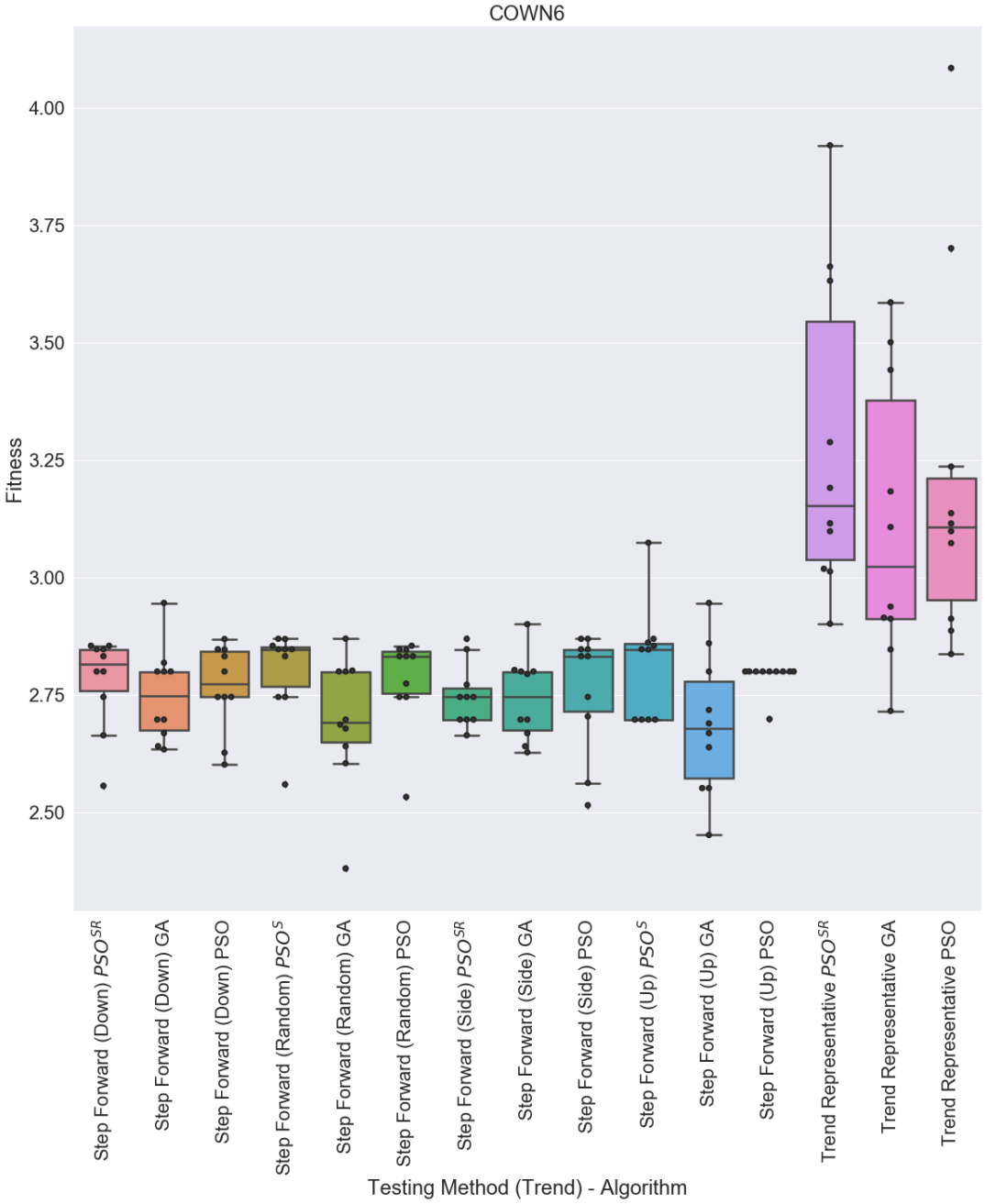

Figure 31: Algorithm box plots for COWN6

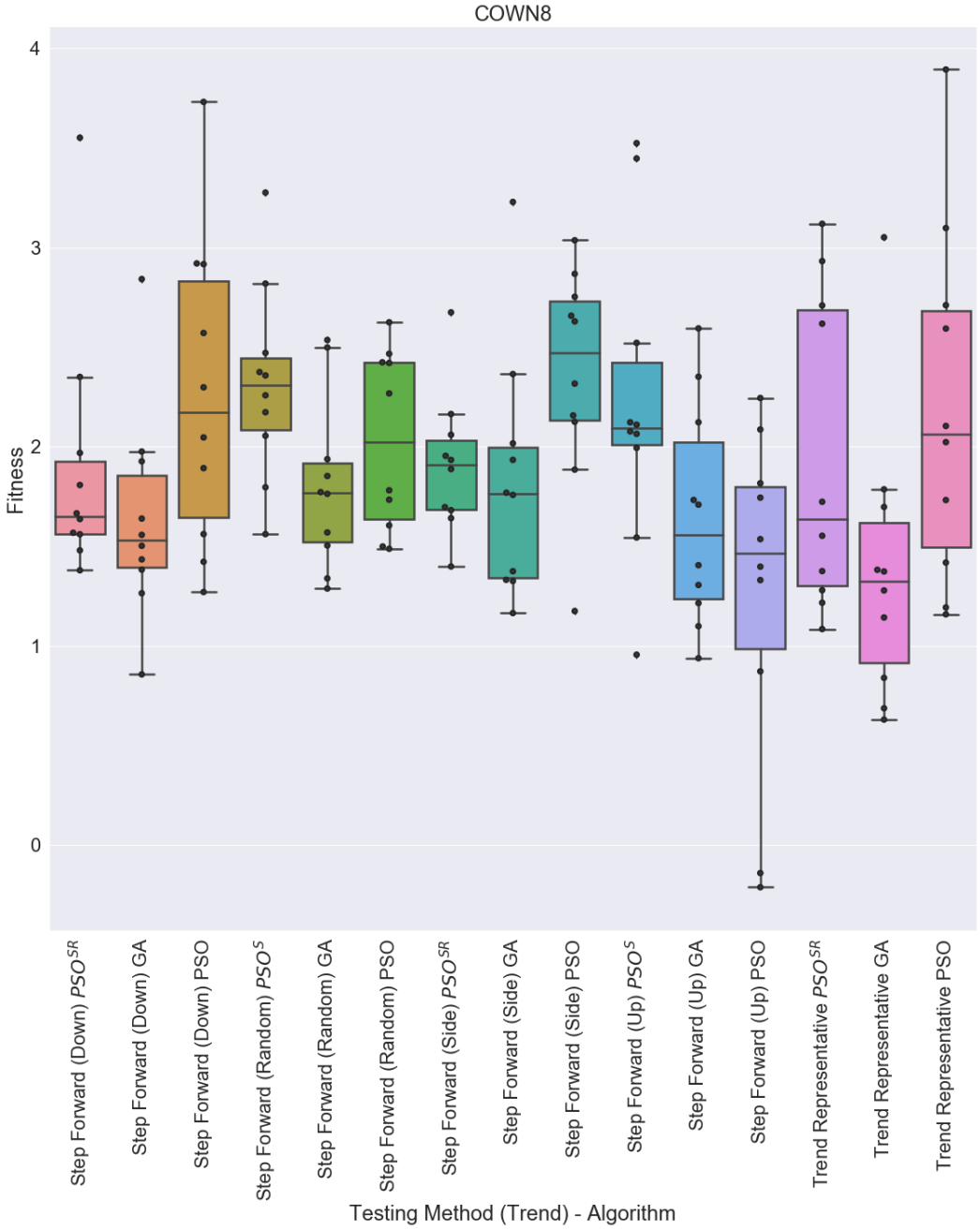

Figure 32: Algorithm box plots for COWN8

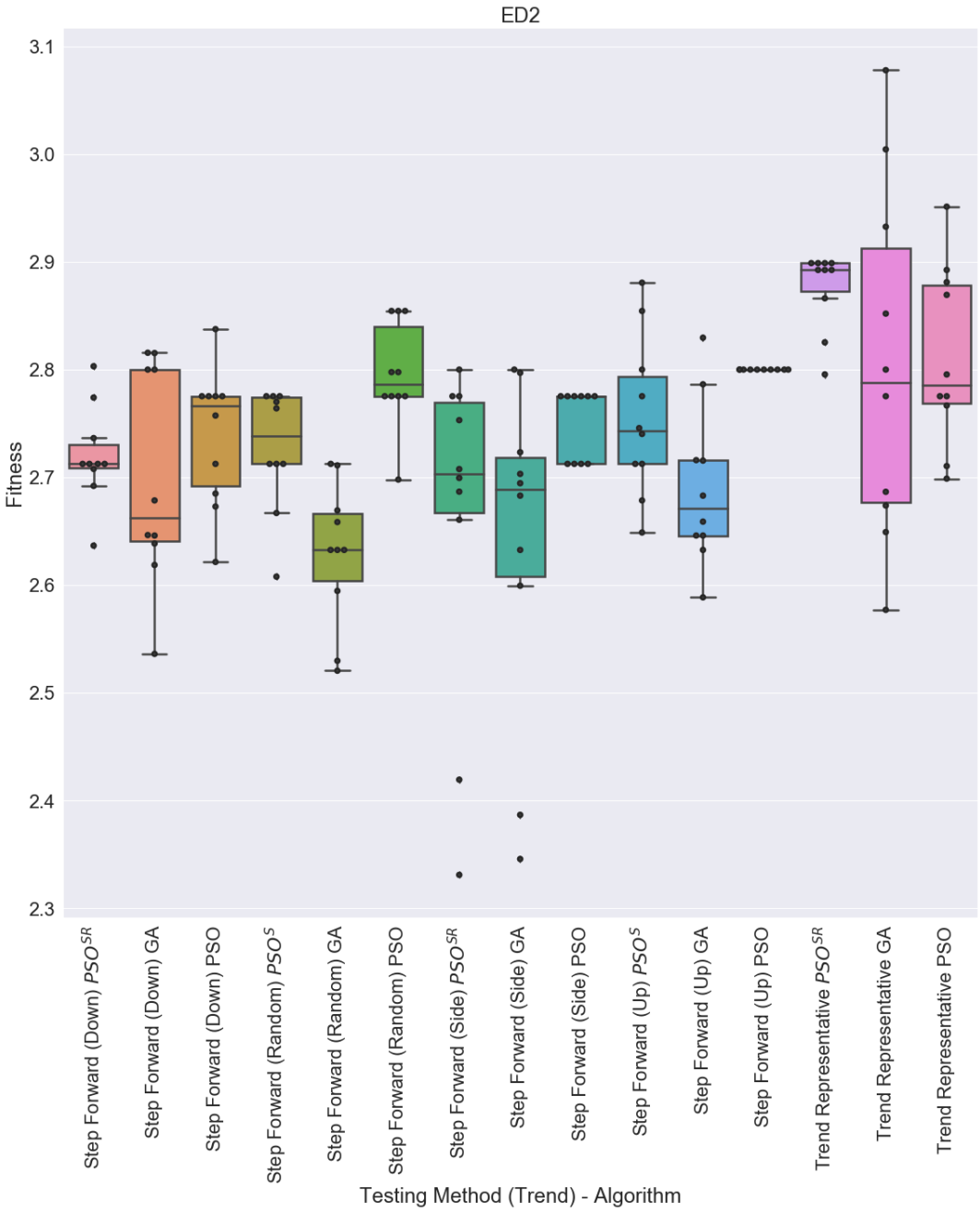

Figure 33: Algorithm box plots for ED2

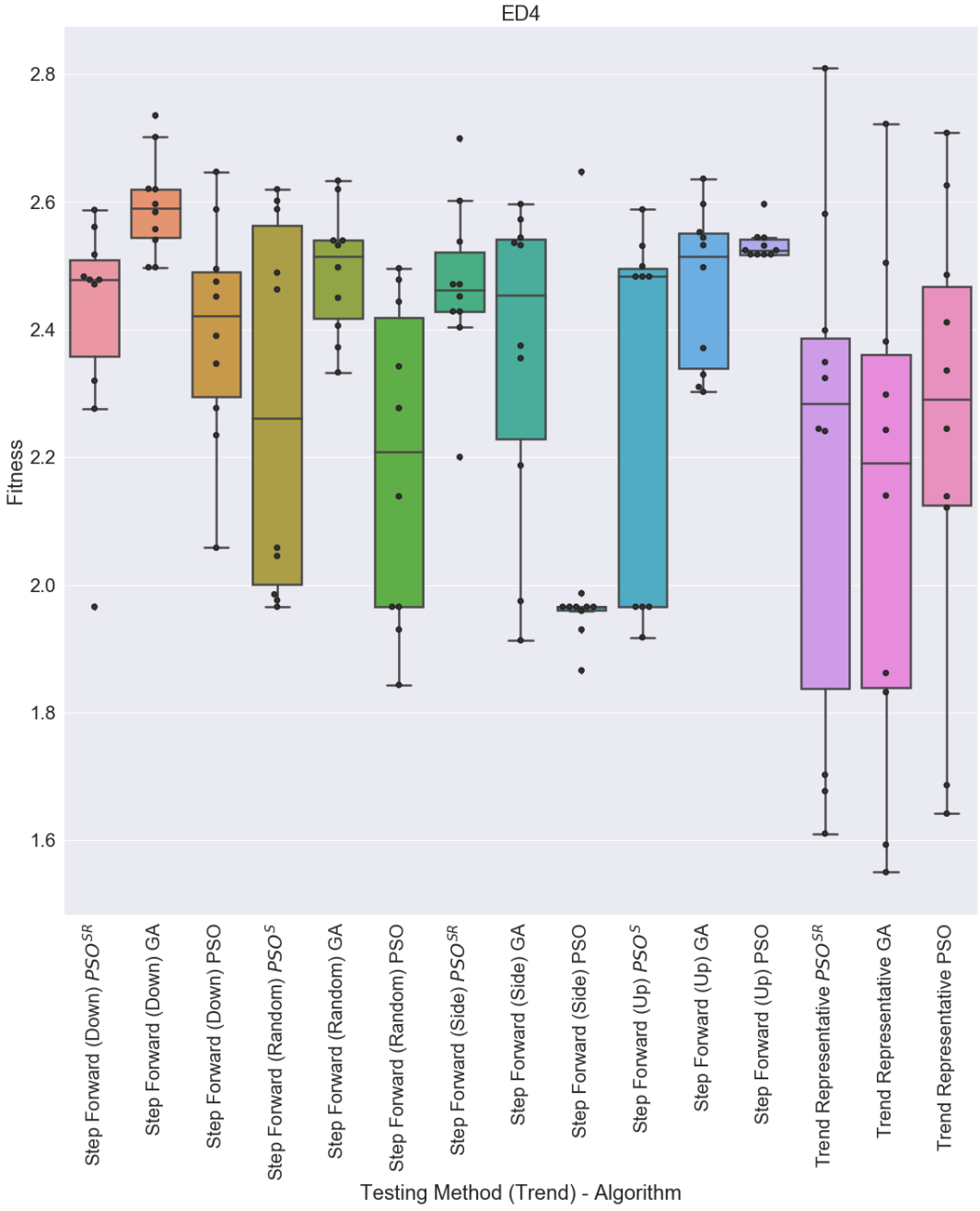

Figure 34: Algorithm box plots for ED4

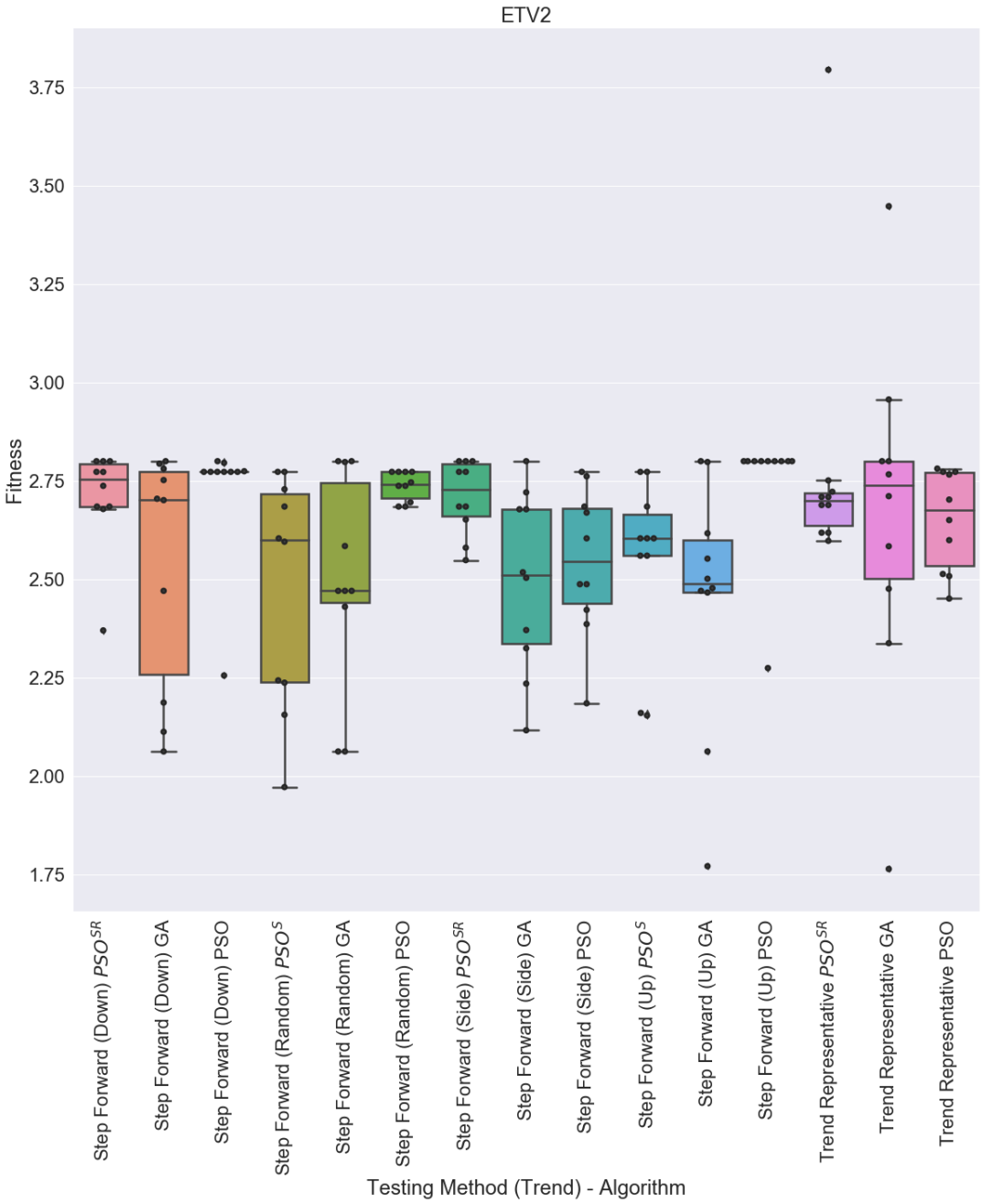

Figure 35: Algorithm box plots for ETV2

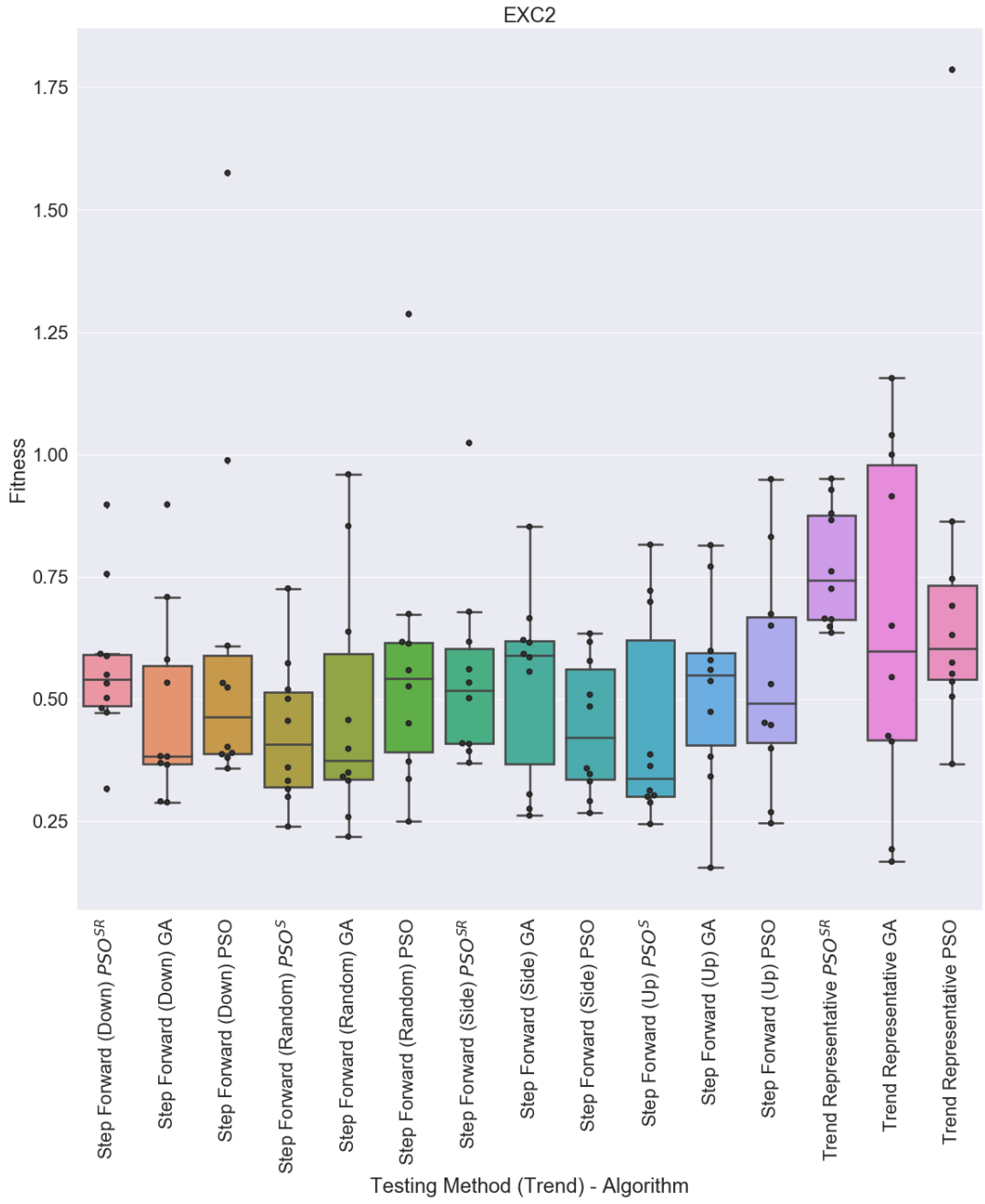

Figure 36: Algorithm box plots for EXC2

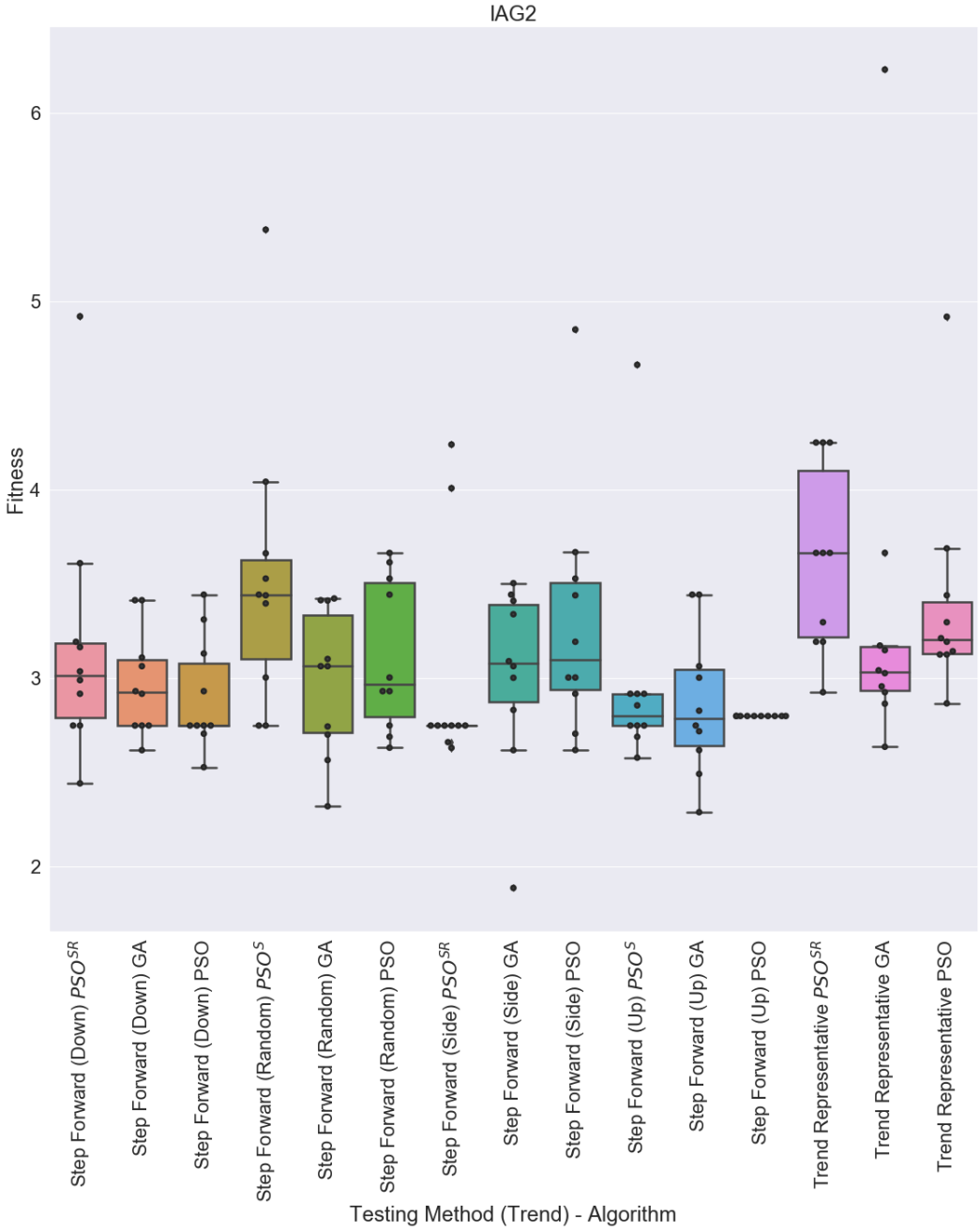

Figure 37: Algorithm box plots for IAG2

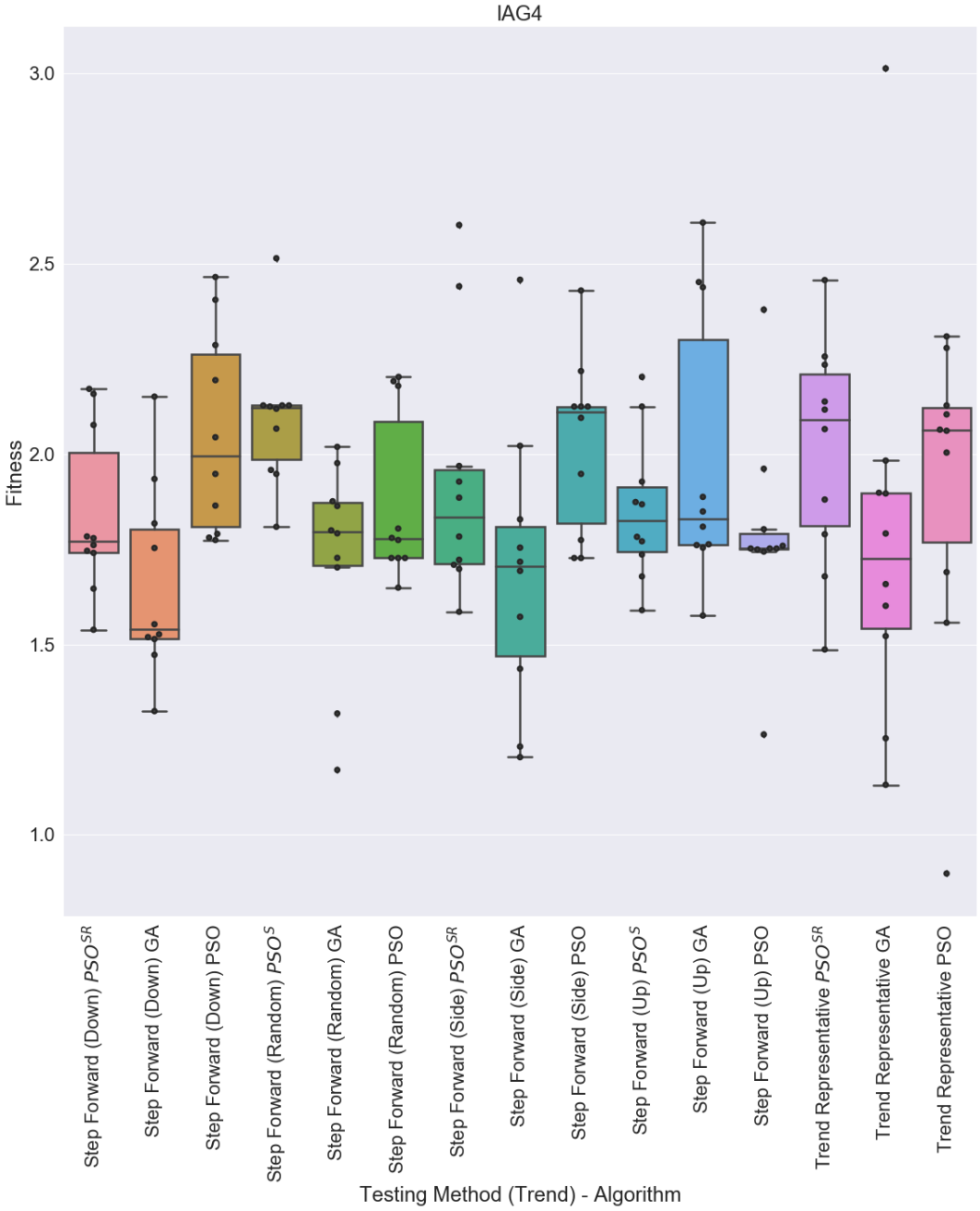

Figure 38: Algorithm box plots for IAG4

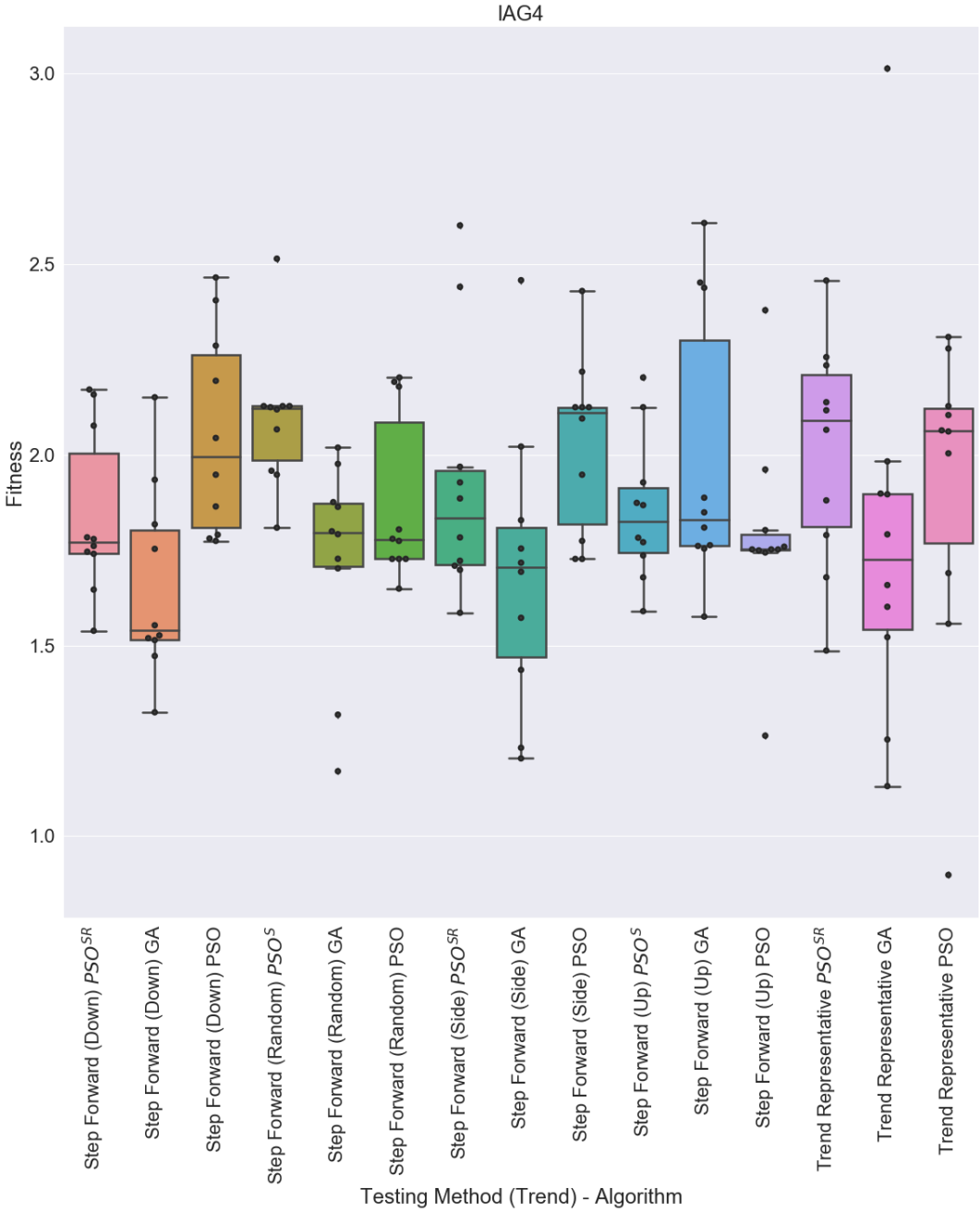

Figure 39: Algorithm box plots for IAG4

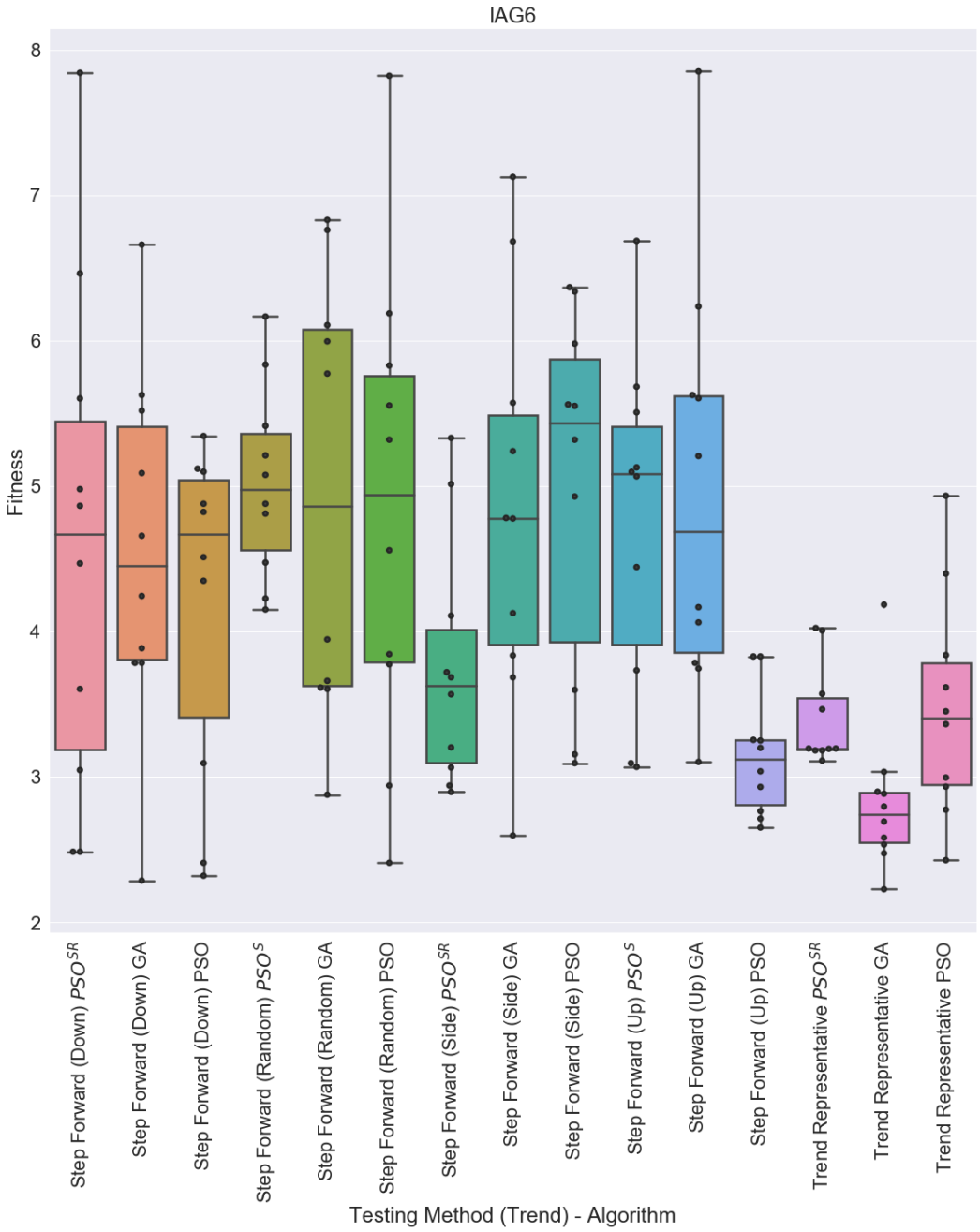

Figure 40: Algorithm box plots for IAG6

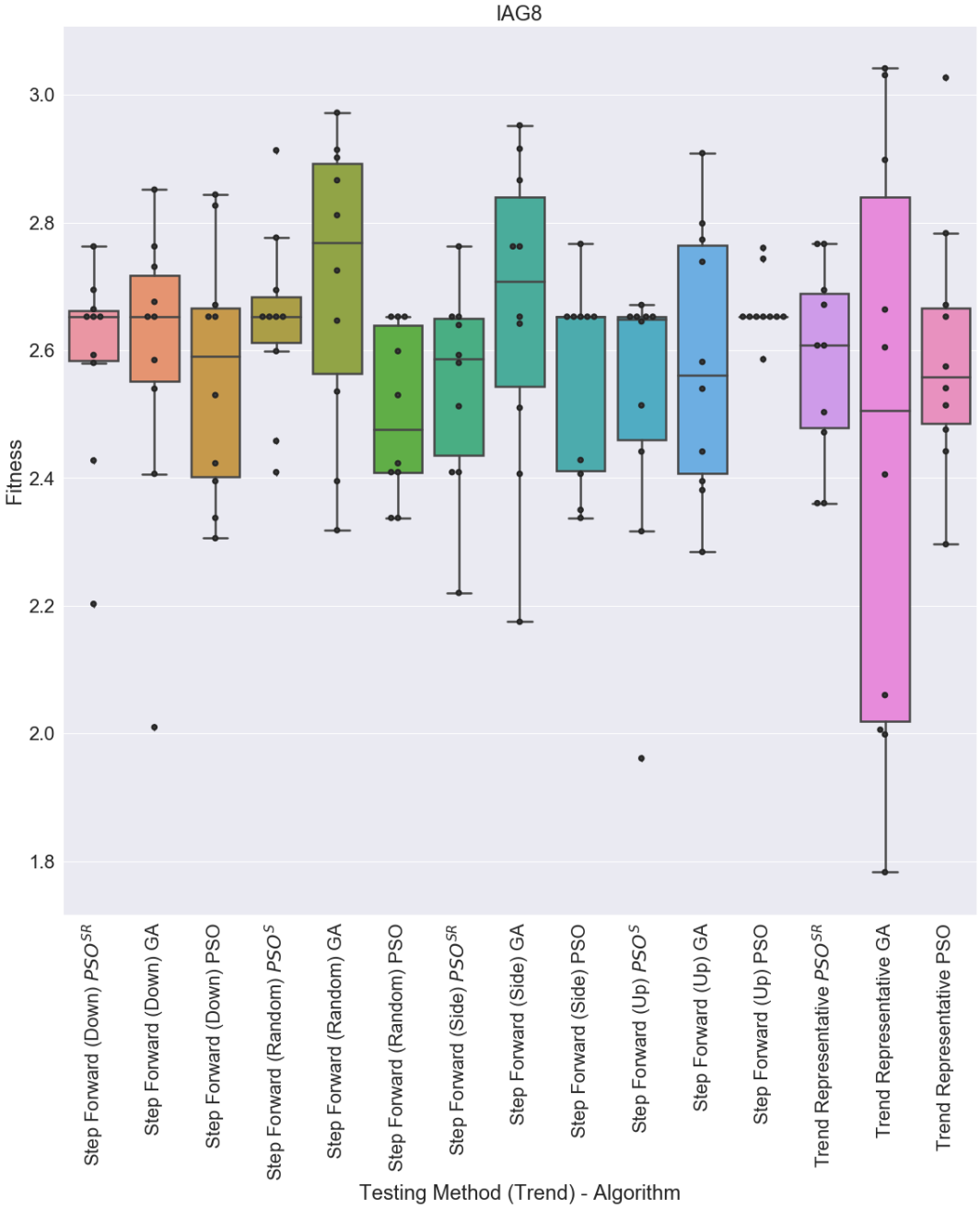

Figure 41: Algorithm box plots for IAG8

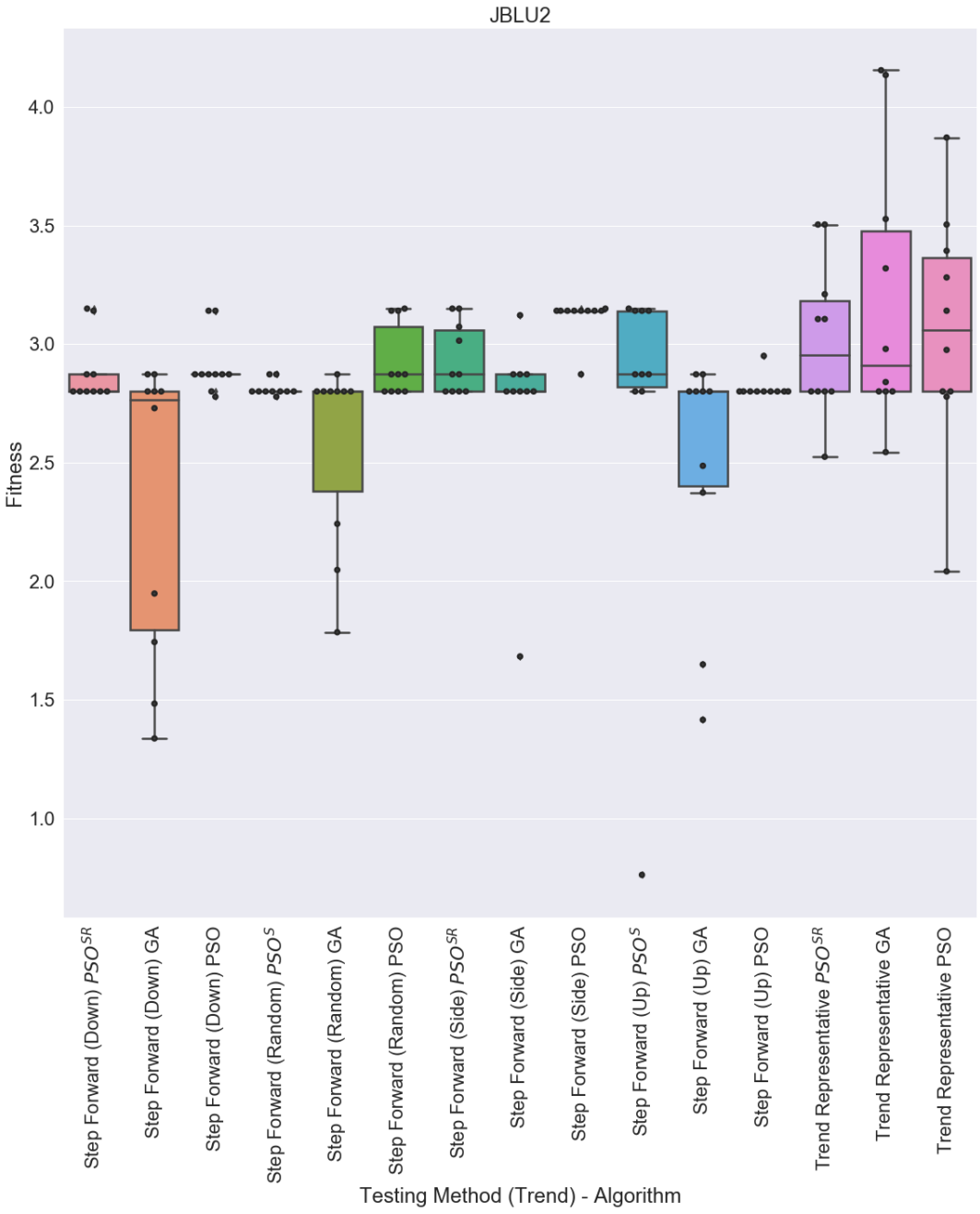

Figure 42: Algorithm box plots for JBLU2

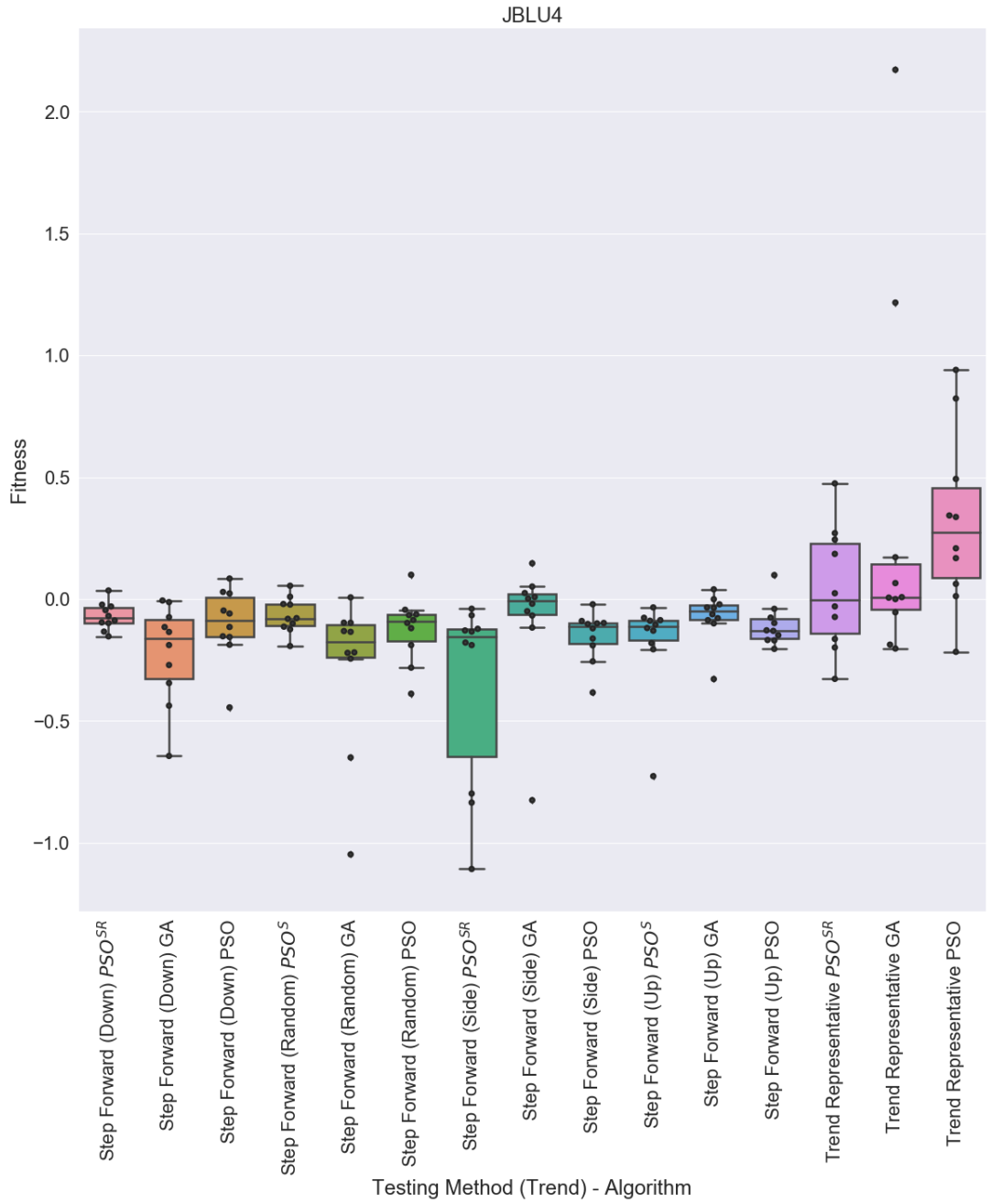

Figure 43: Algorithm box plots for JBLU4

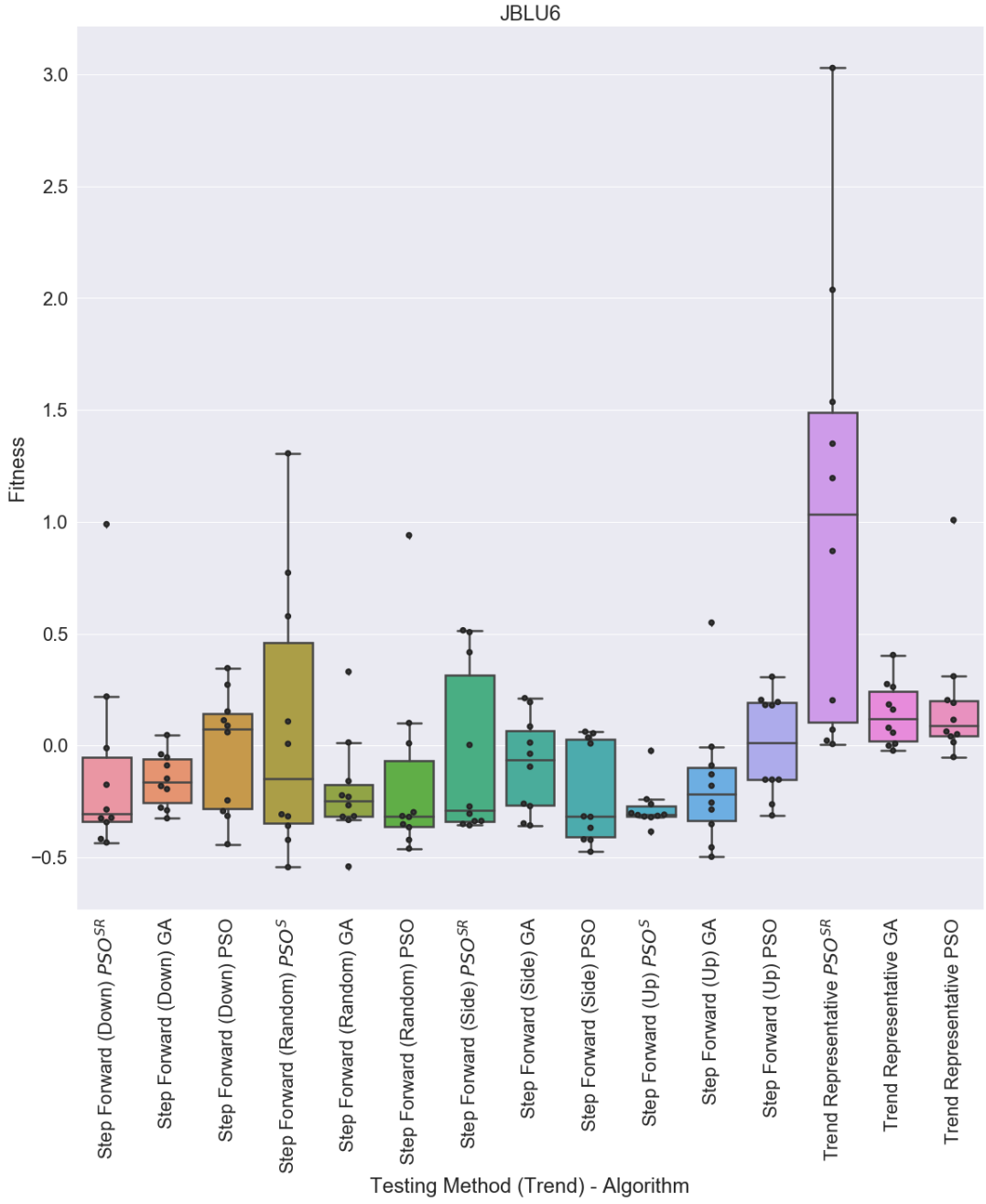

Figure 44: Algorithm box plots for JBLU6

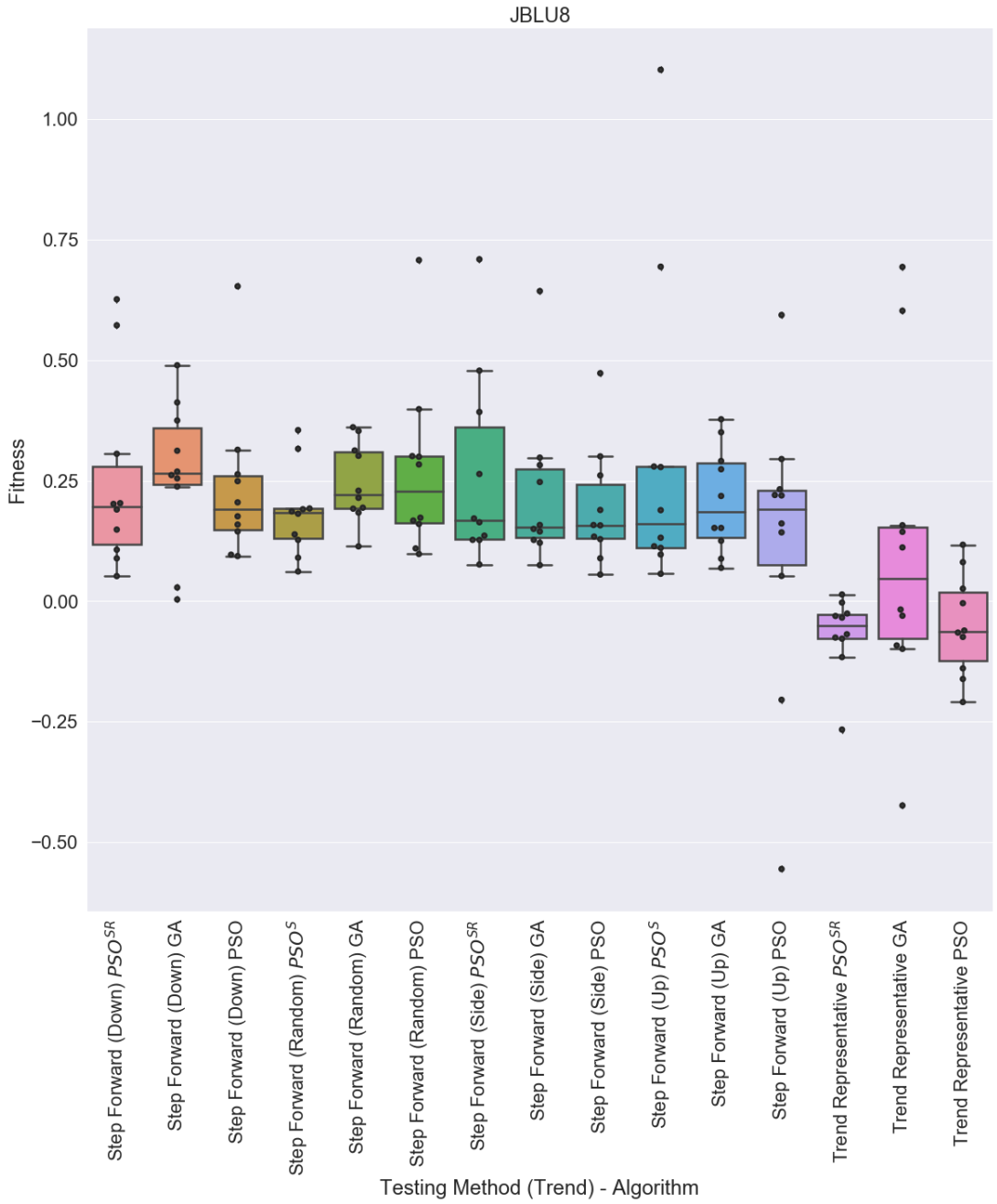

Figure 45: Algorithm box plots for JBLU8

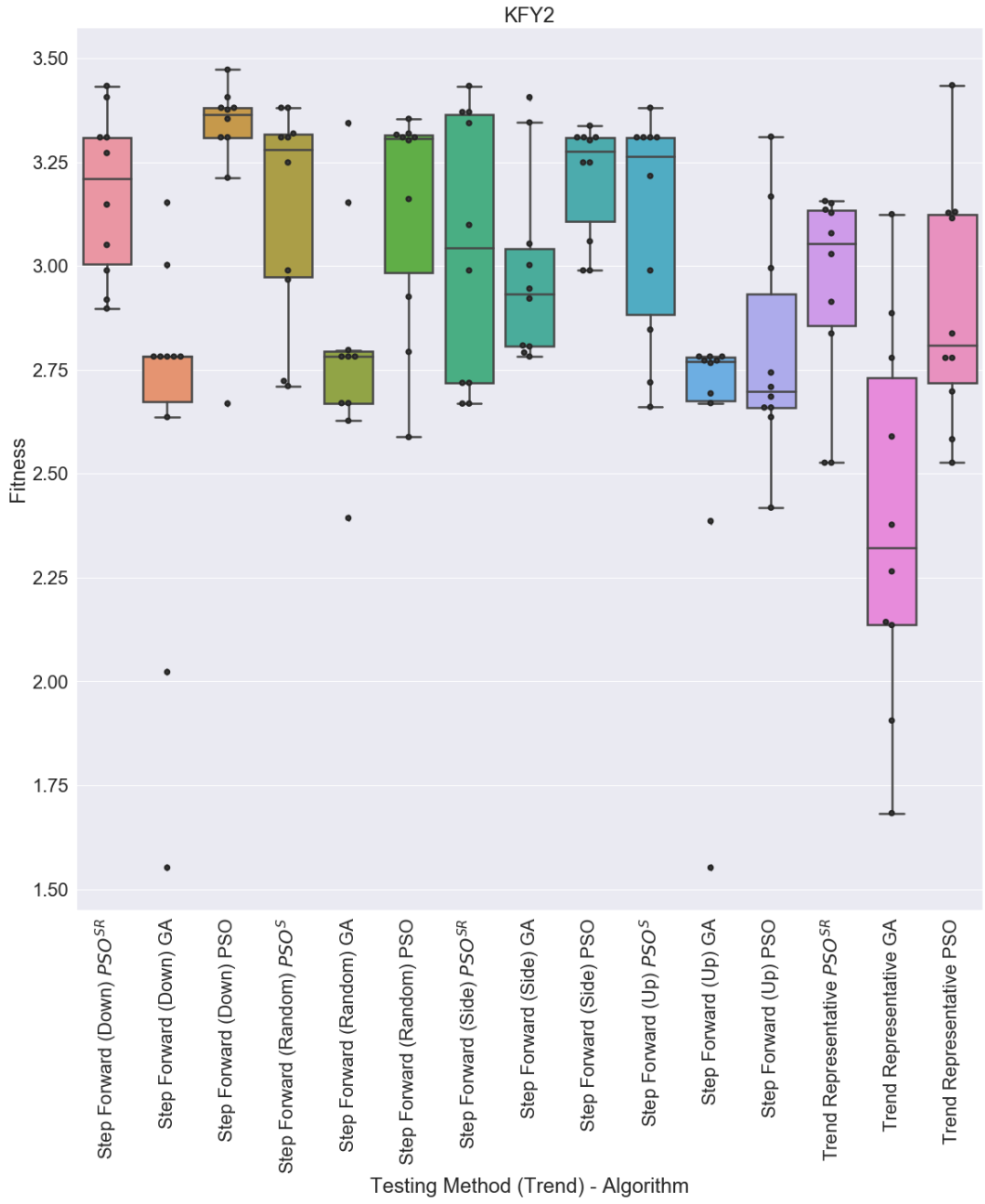

Figure 46: Algorithm box plots for KFY2

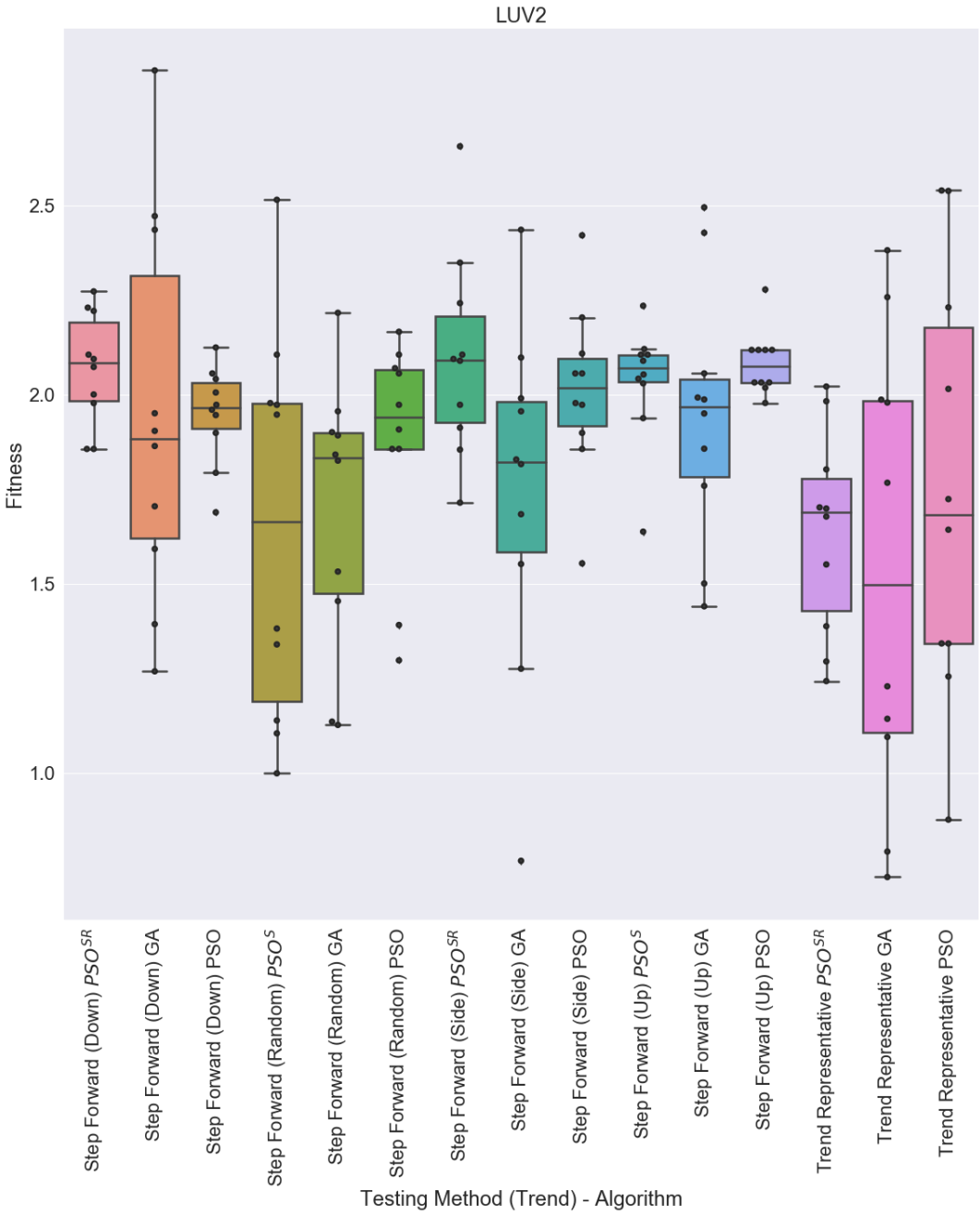

Figure 47: Algorithm box plots for LUV2

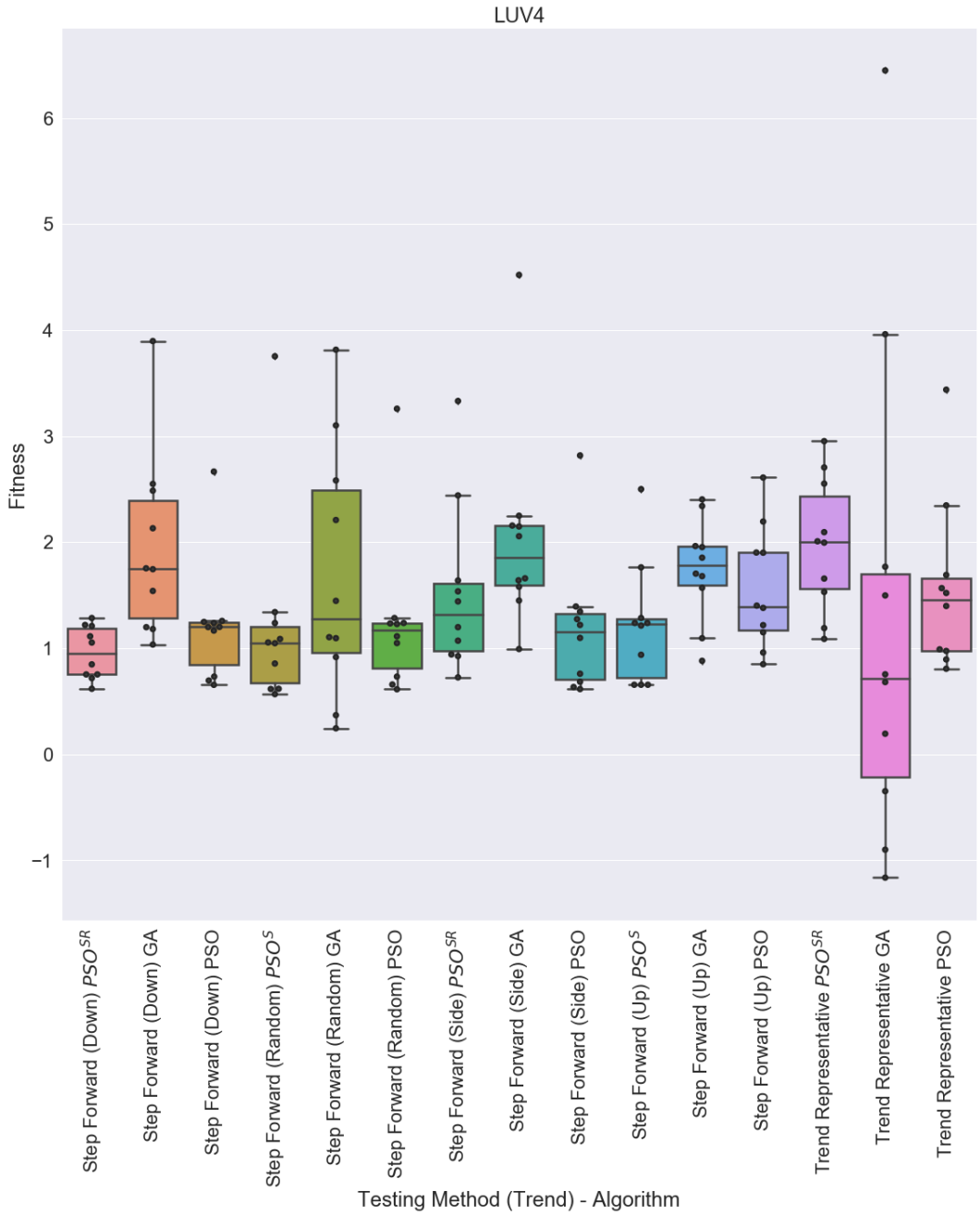

Figure 48: Algorithm box plots for LUV4

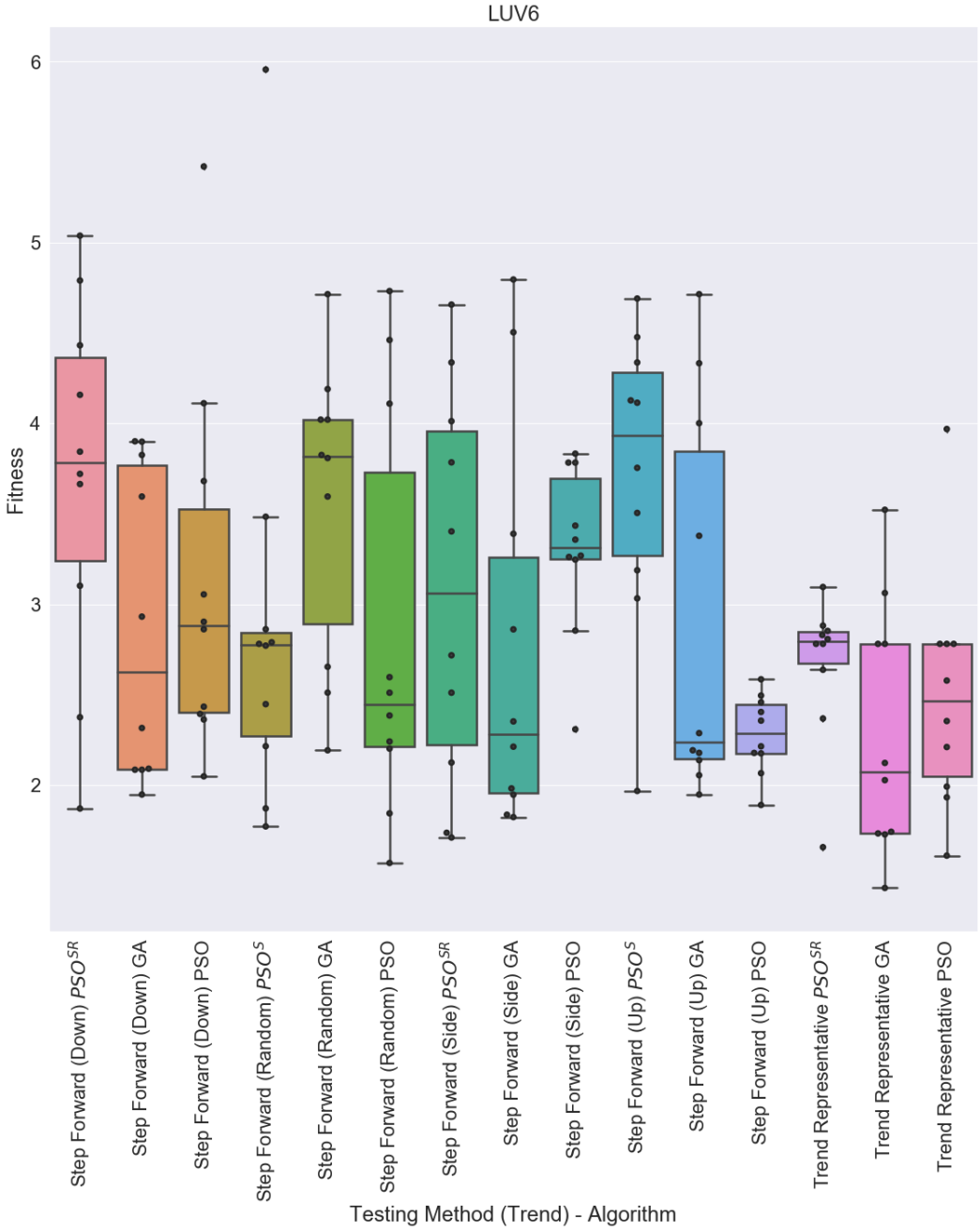

Figure 49: Algorithm box plots for LUV6

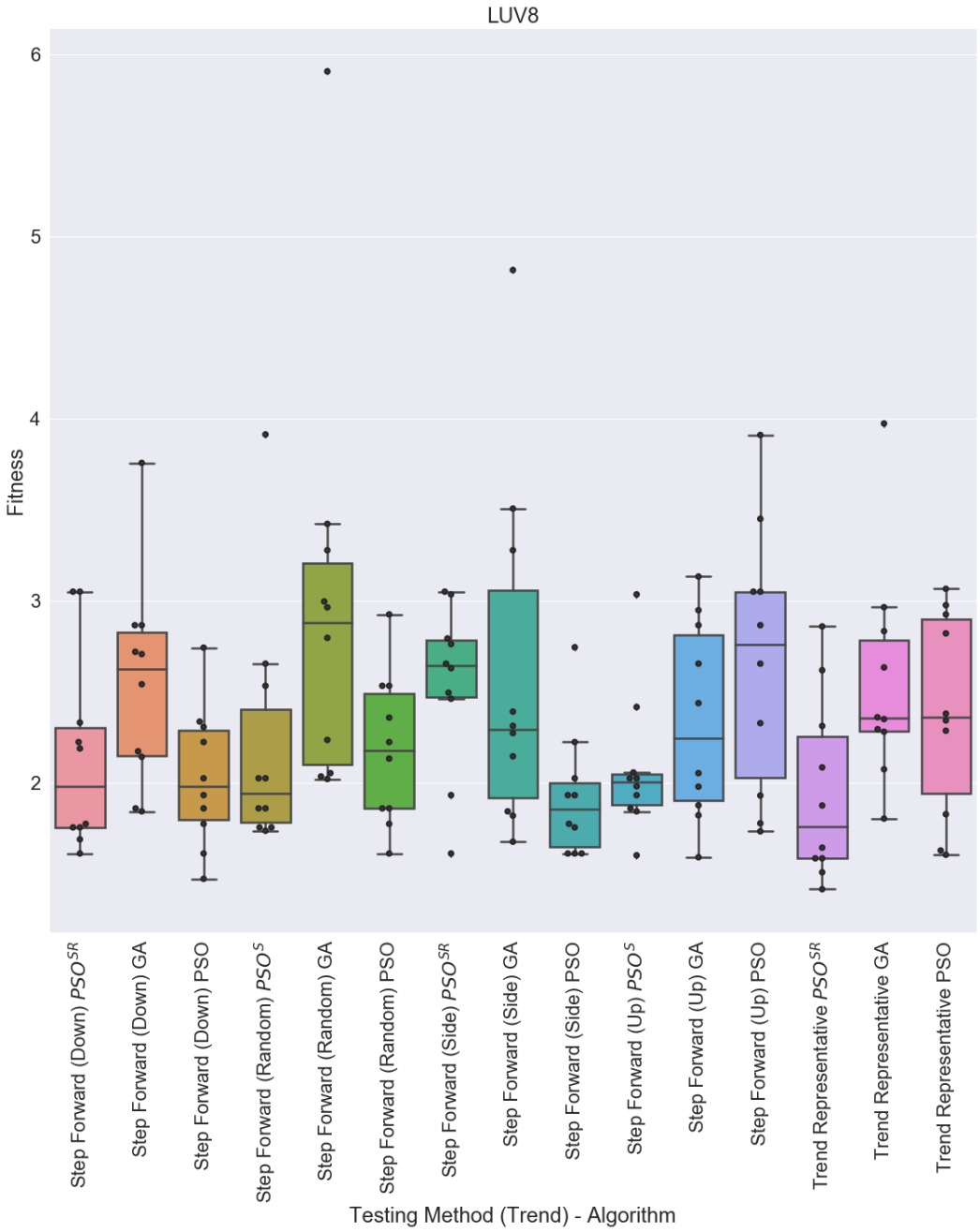

Figure 50: Algorithm box plots for LUV8

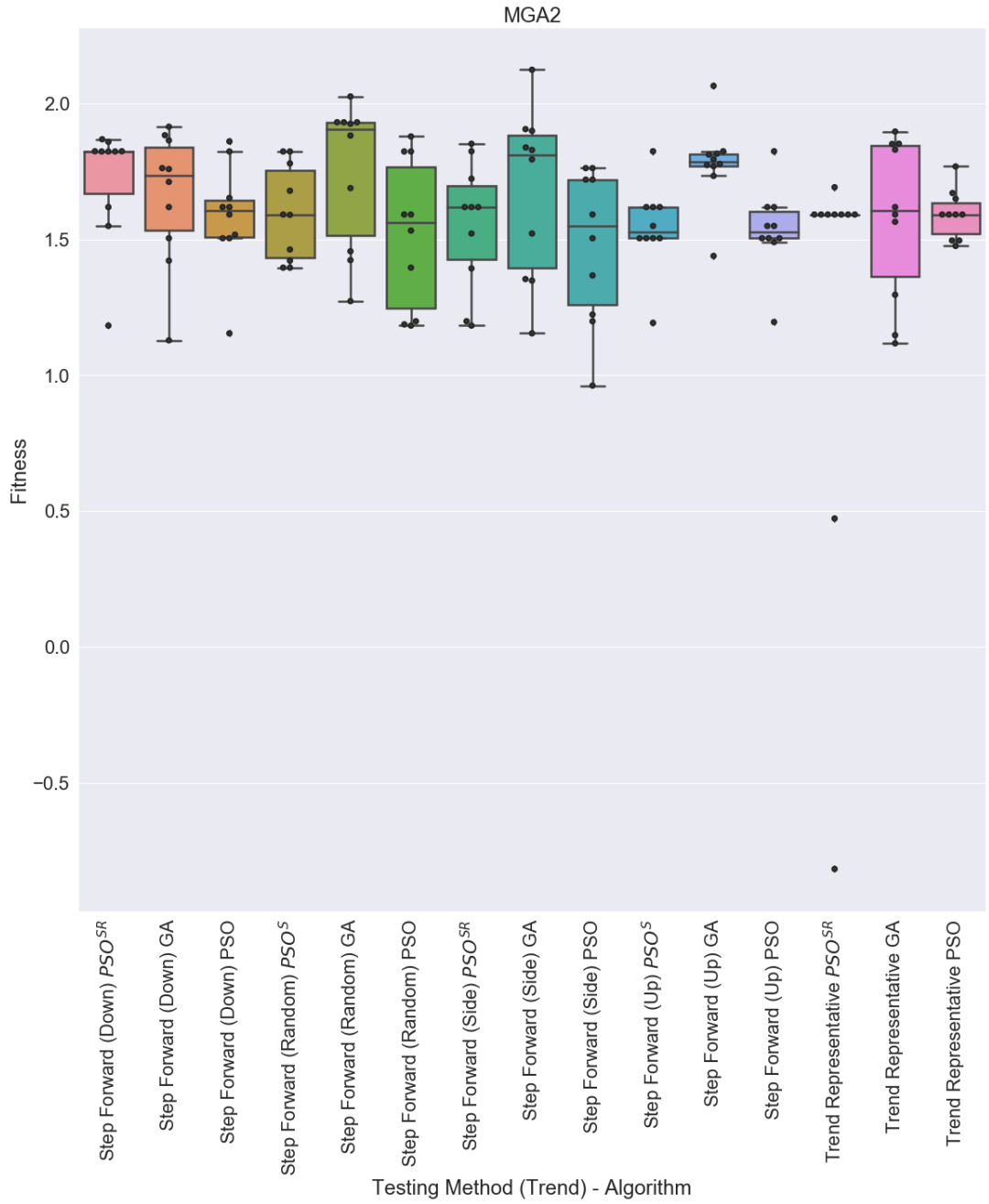

Figure 51: Algorithm box plots for MGA2

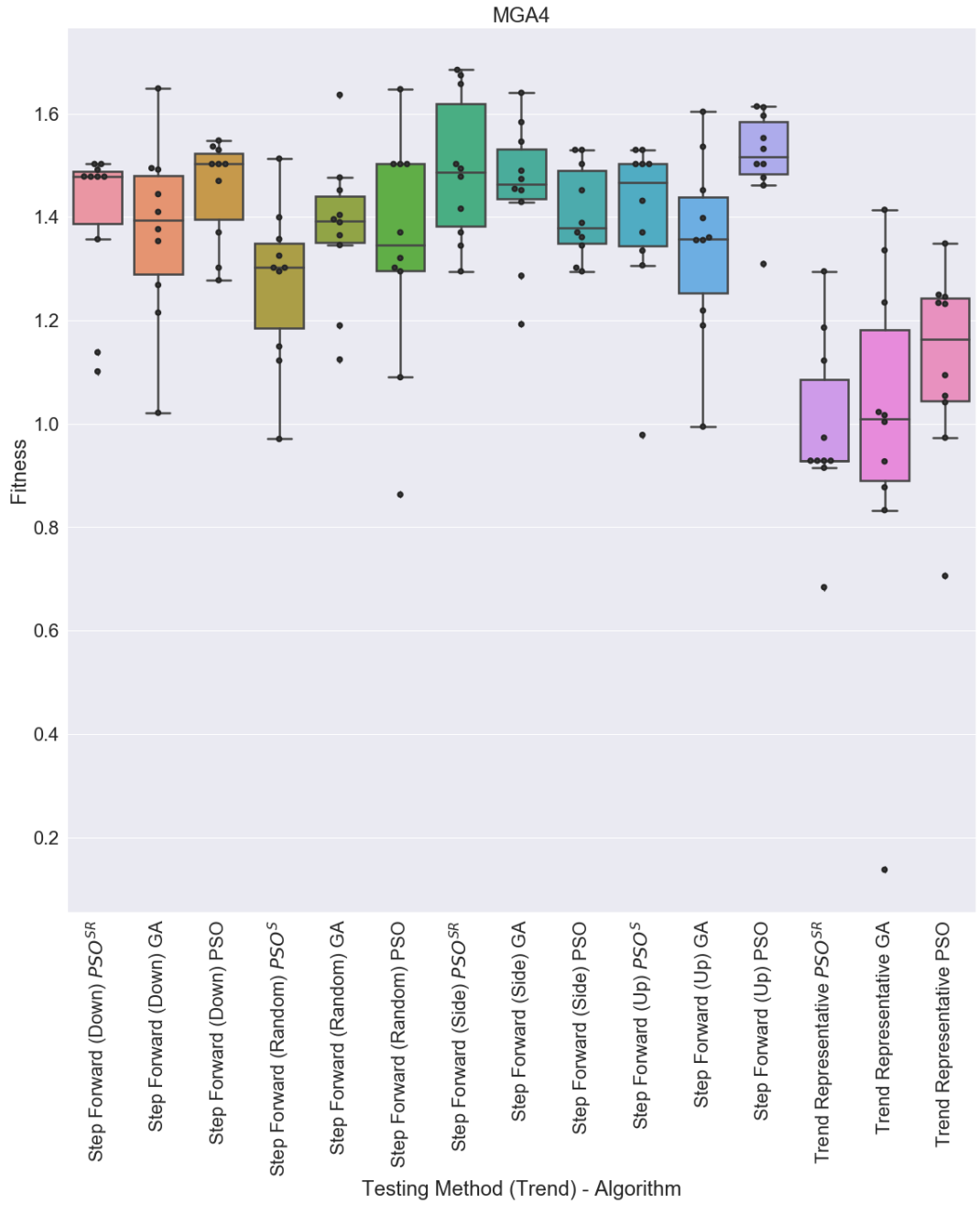

Figure 52: Algorithm box plots for MGA4

Appendix II: RadViz Plots for Multiobjective Algorithms, NSGA-II and MACD

In this appendix, we present the RadViz plots obtained for λ-PSO, λ-PSO$^{\text{SP}}$, λ-GA, NSGA-II and MACD while tackling market timing as a multiobjective optimization problem. Each plot represents the results obtained for a particular testing strand, the name of which can be seen at the top of the plot. An explanation of how to interpret RadViz plots can be seen in Figure 21, Chapter 7.

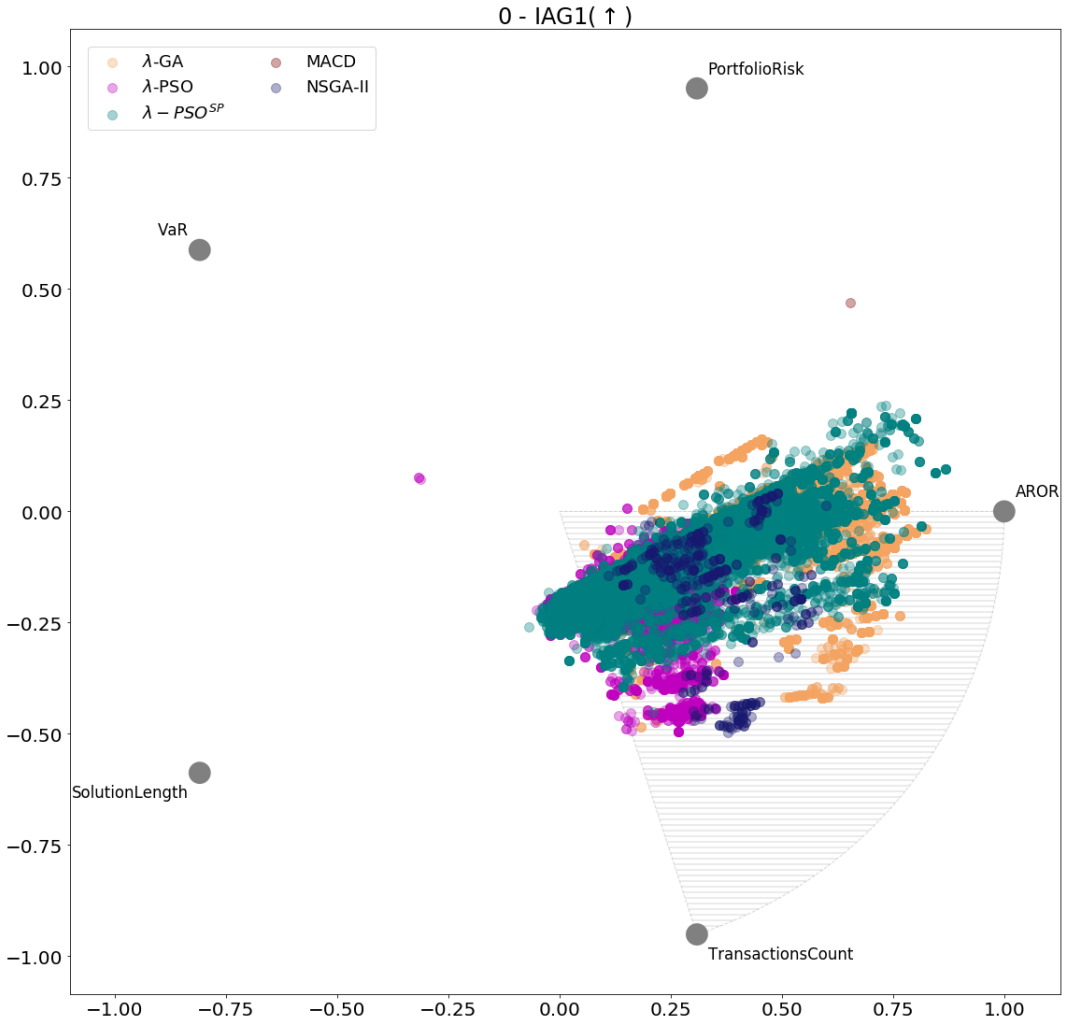

Figure 53: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand IAG1

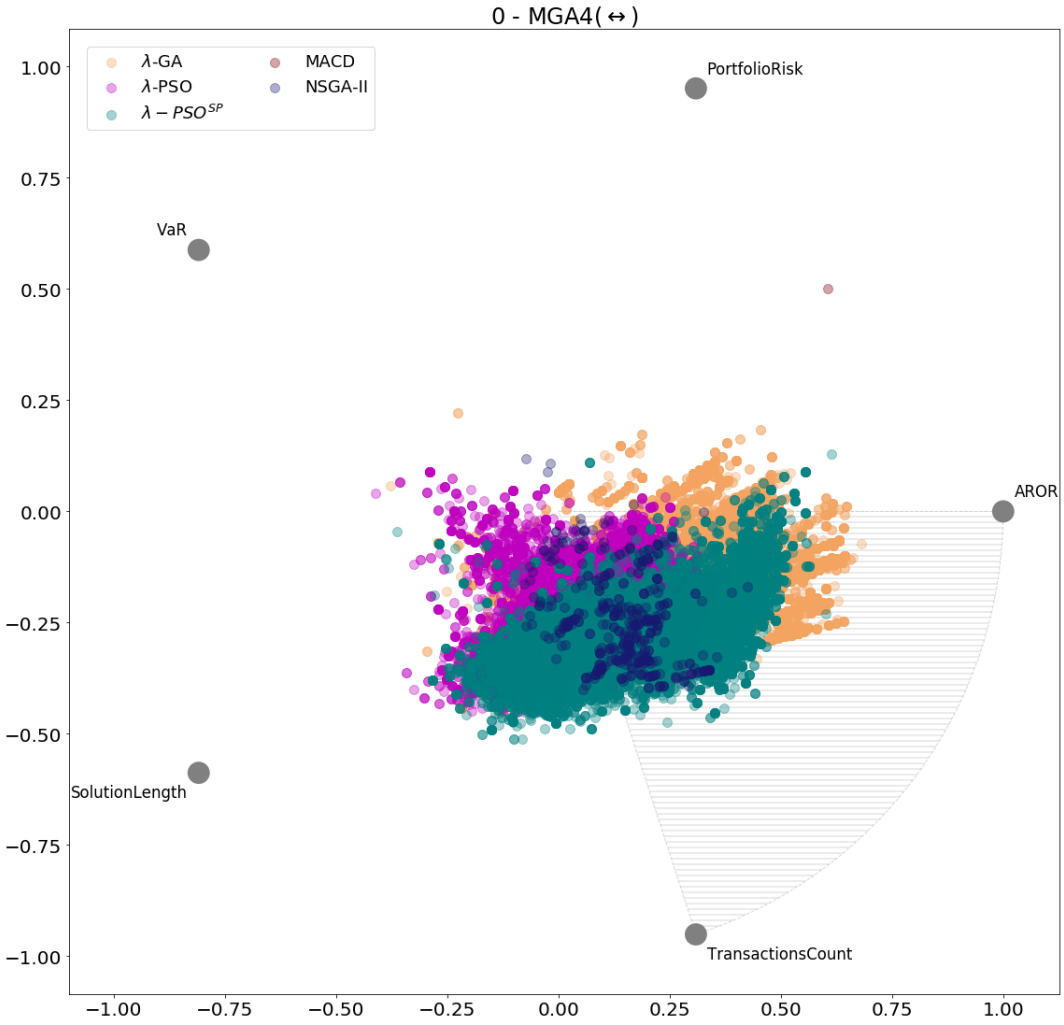

Figure 54: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand MGA4

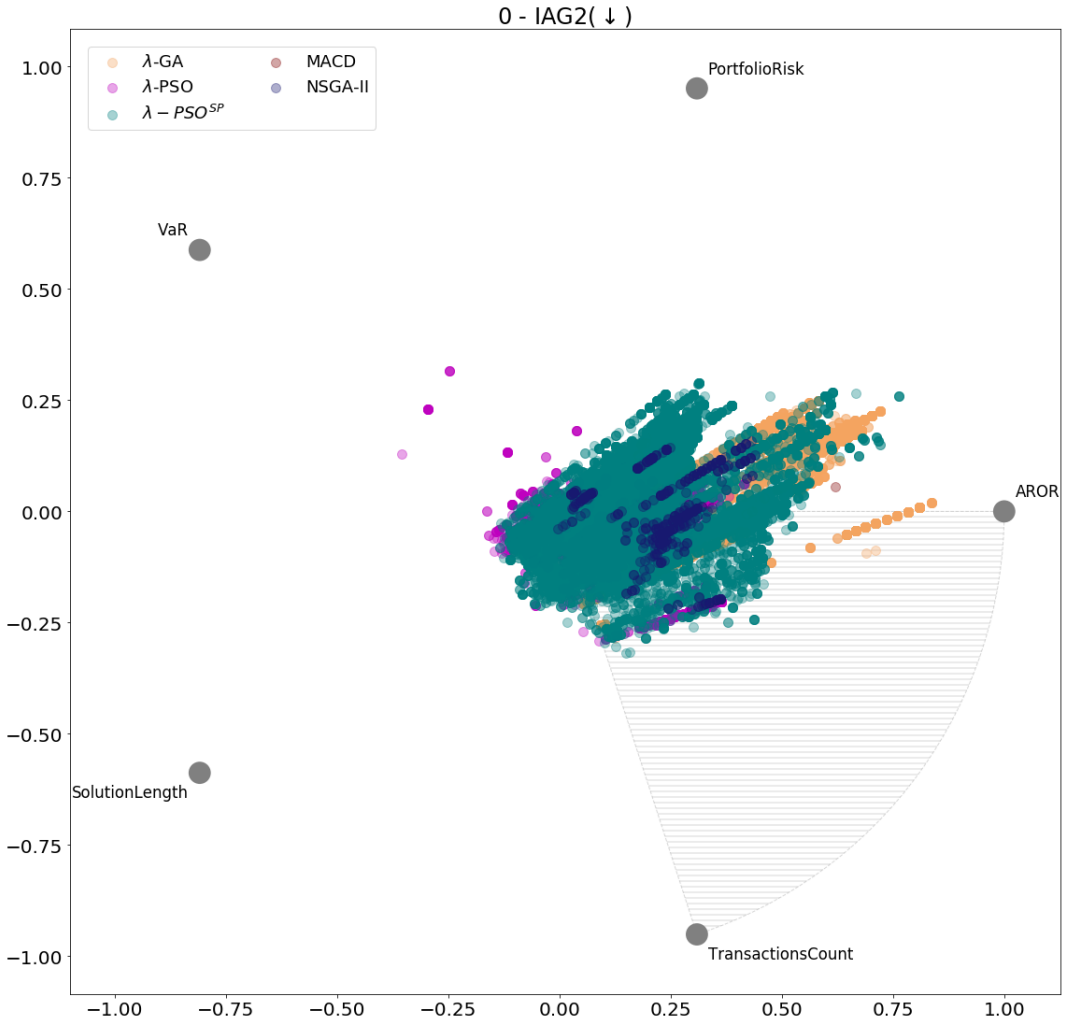

Figure 55: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand IAG2

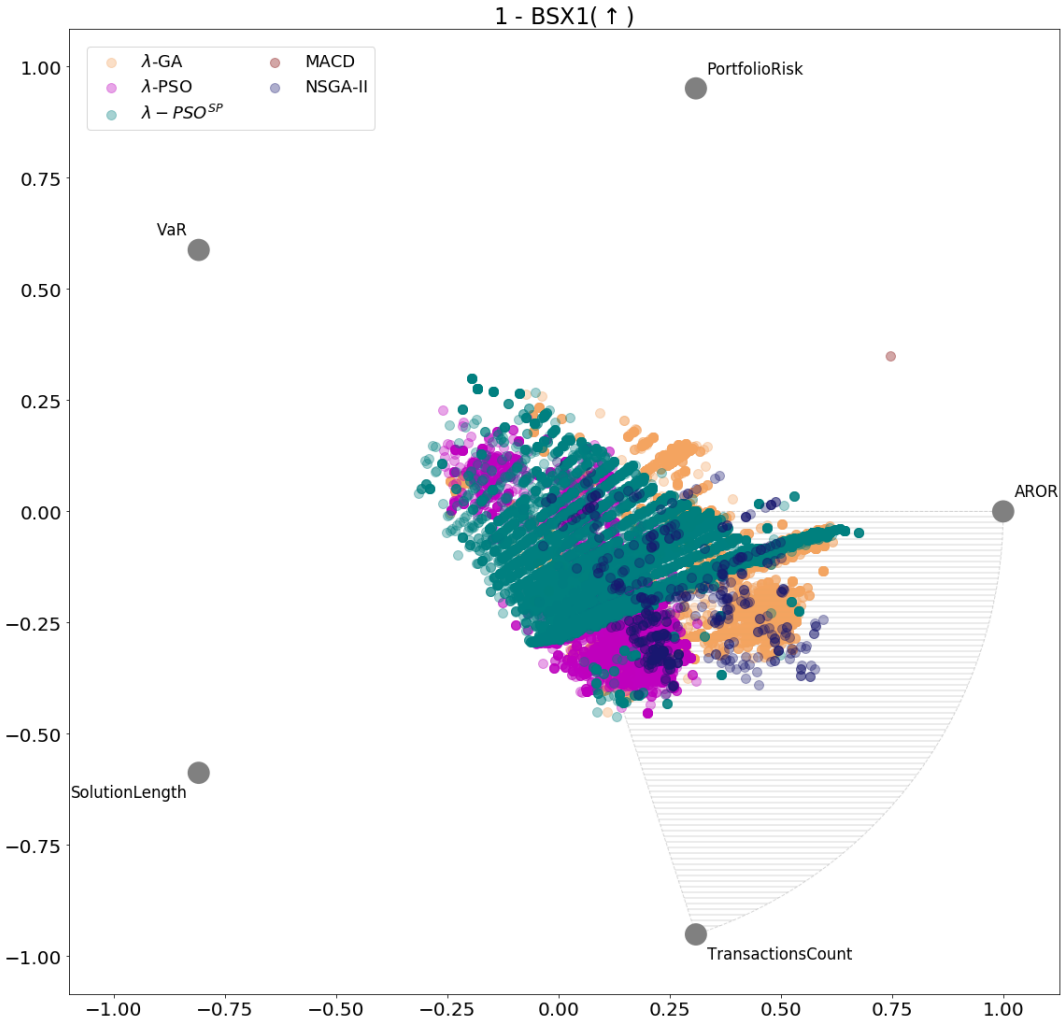

Figure 56: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand BSX1

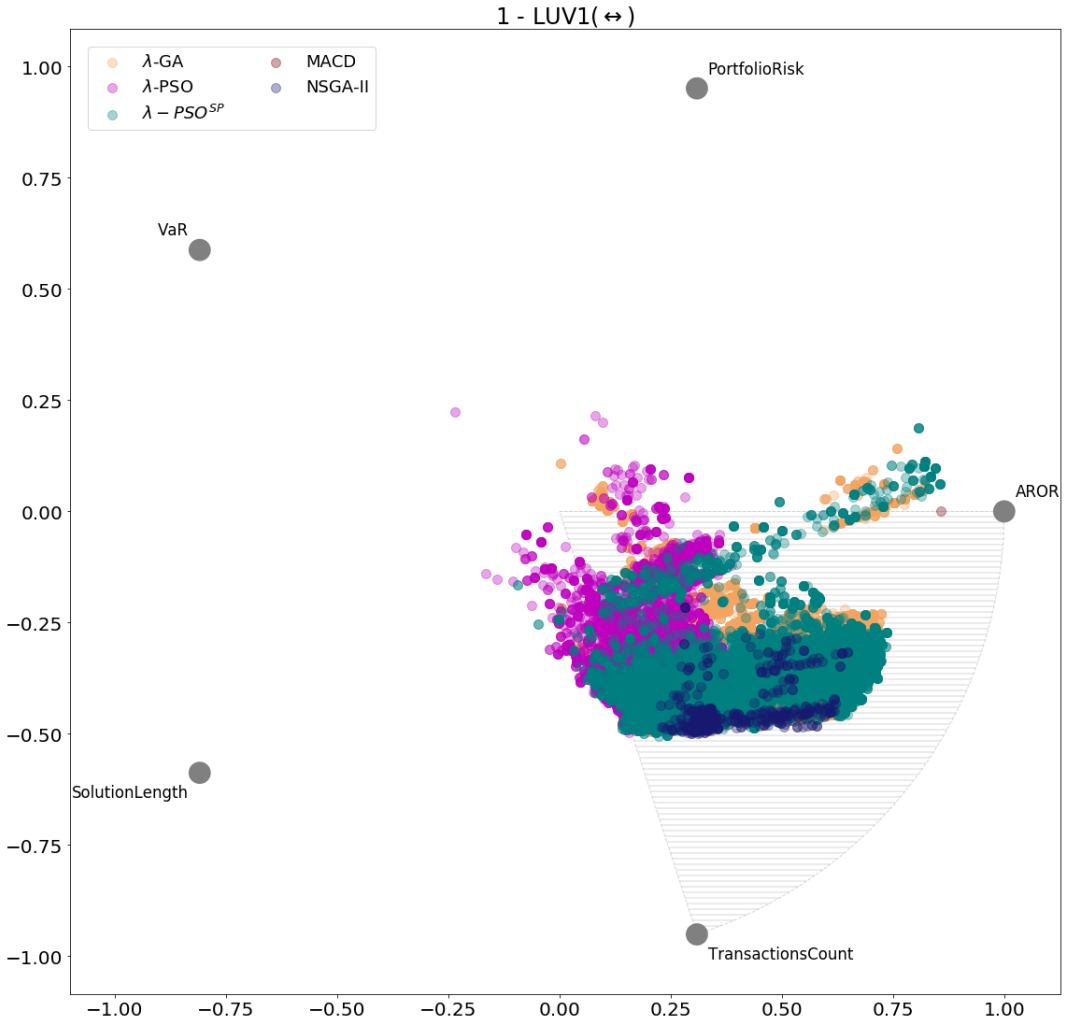

Figure 57: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand LUV1

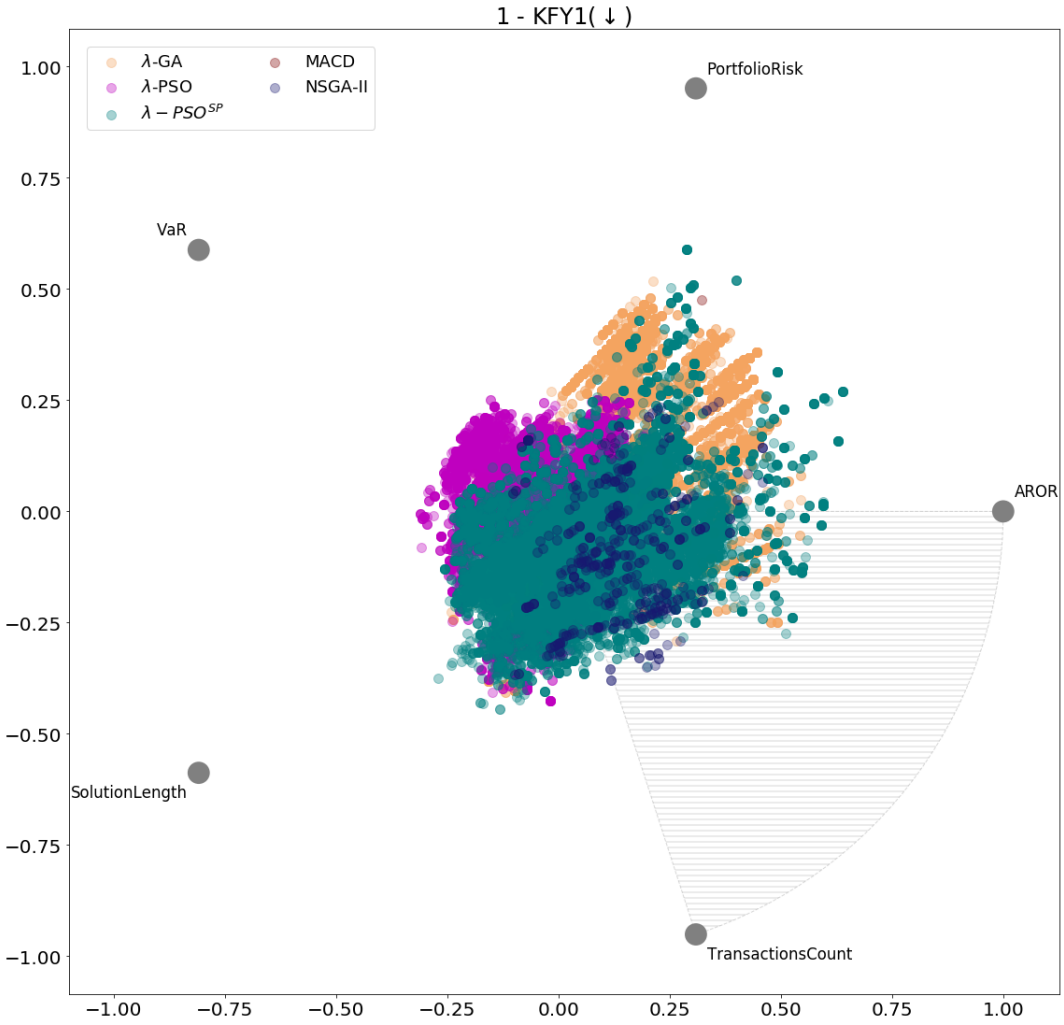

Figure 58: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand KFY1

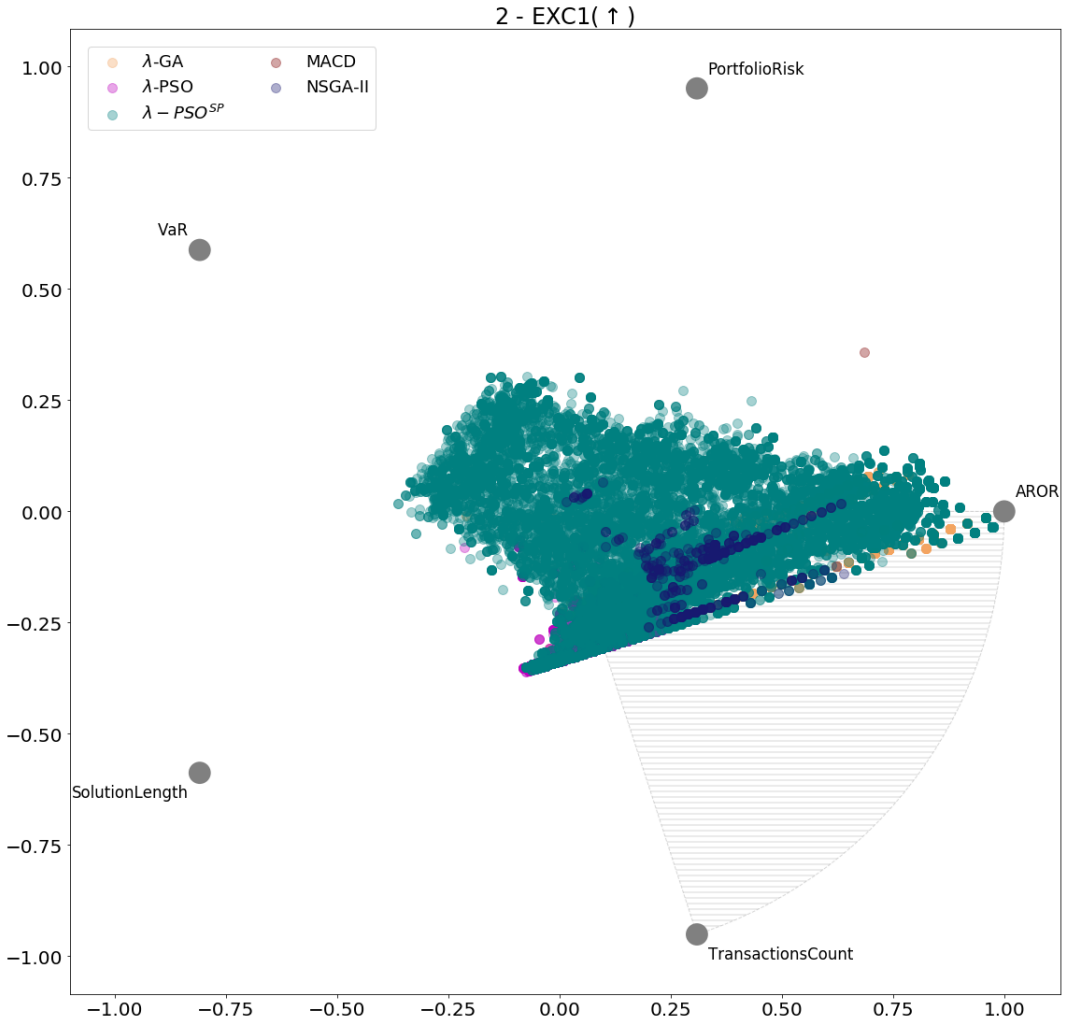

Figure 59: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand EXC1

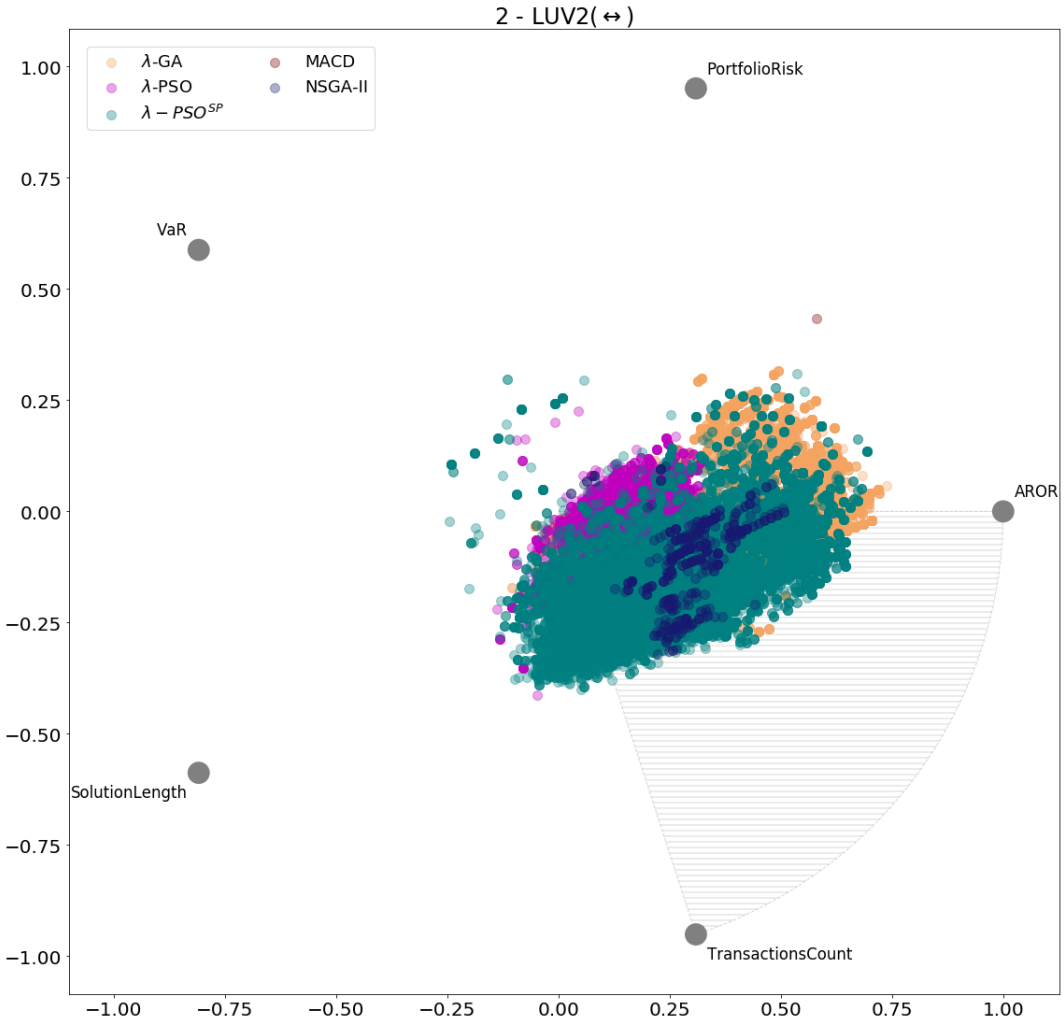

Figure 60: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand LUV2

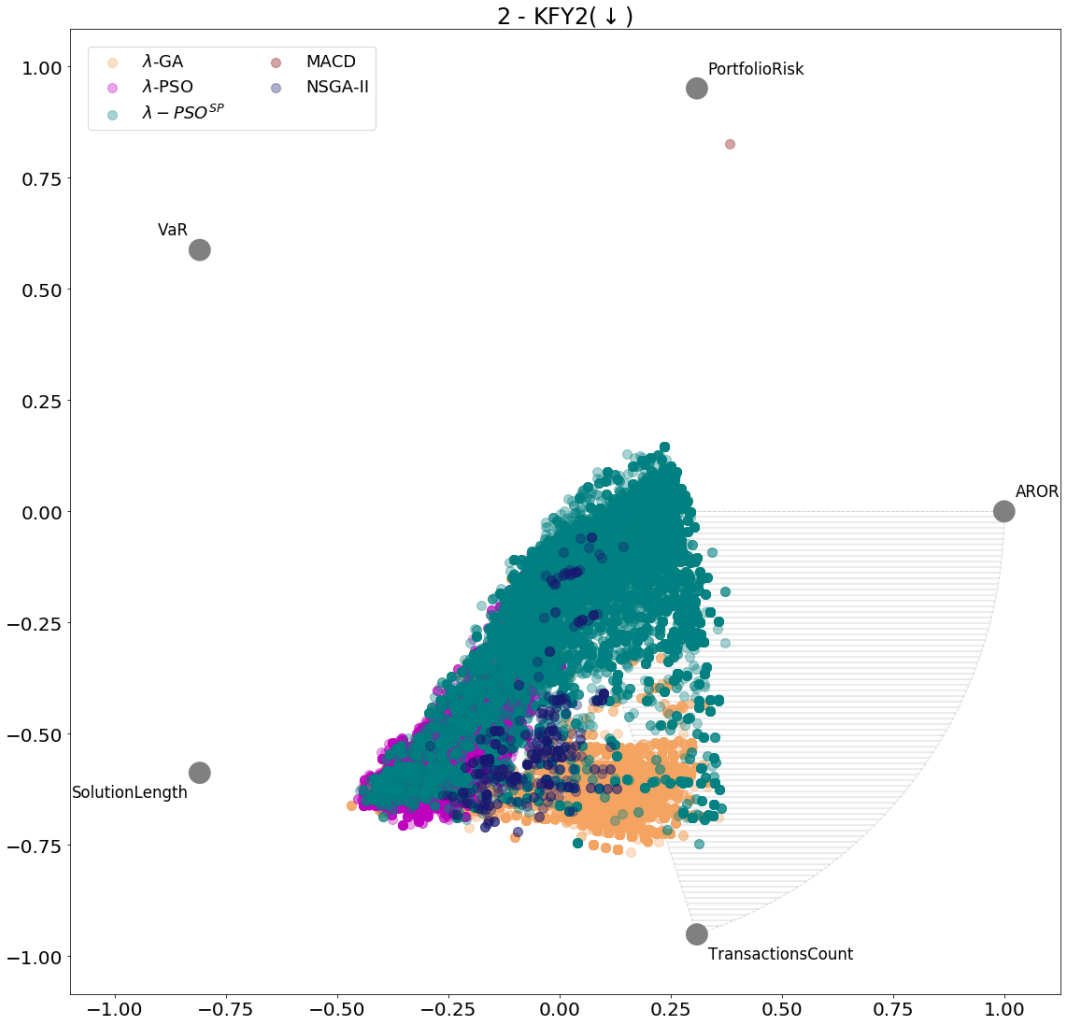

Figure 61: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand KFY2

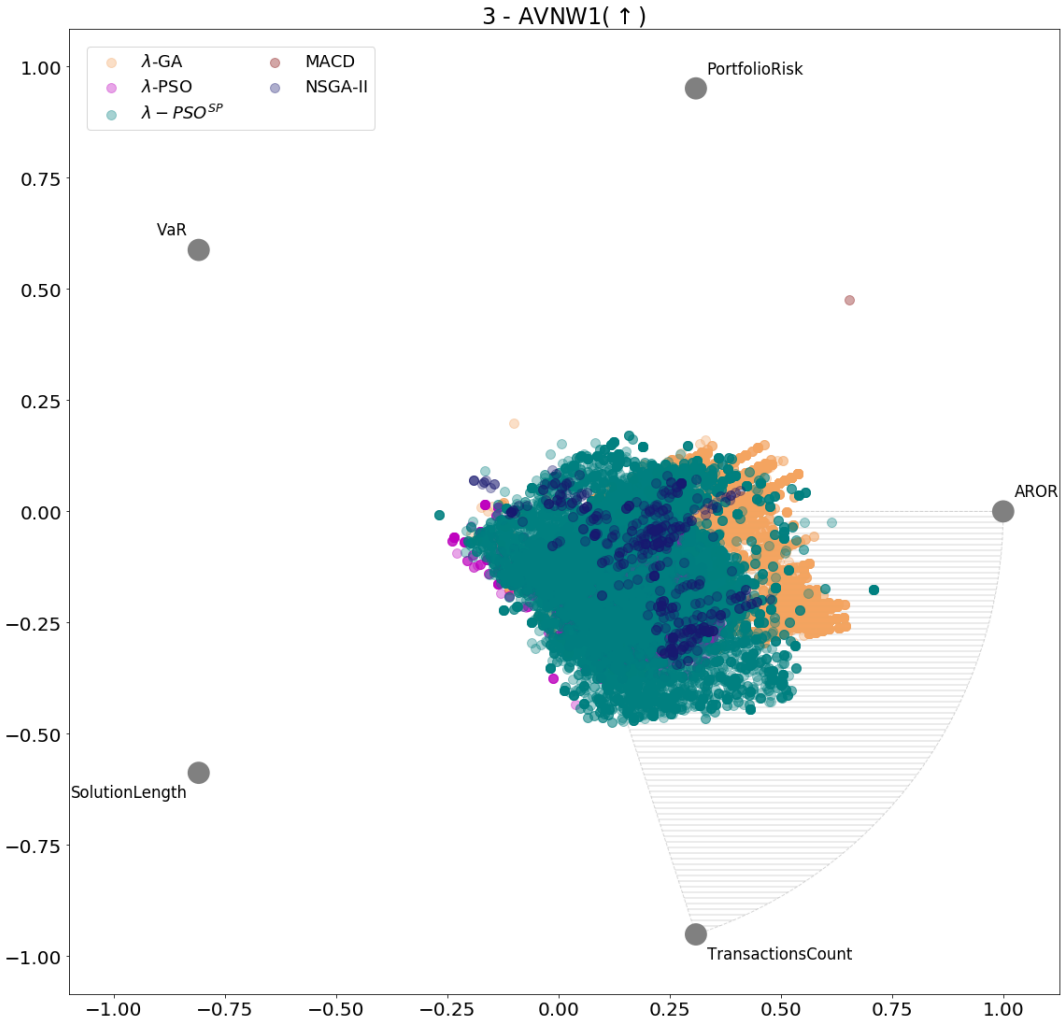

Figure 62: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand AVNW1

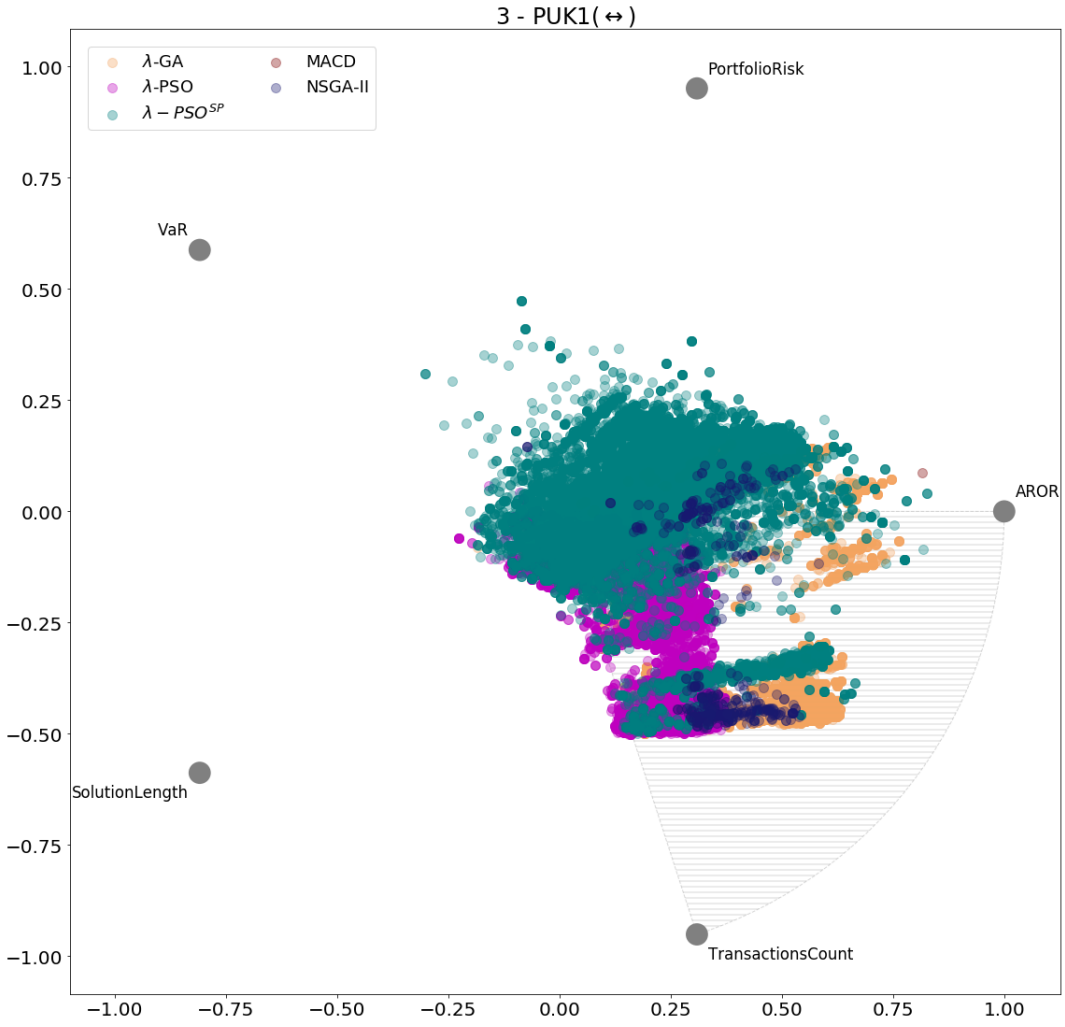

Figure 63: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand PUK1

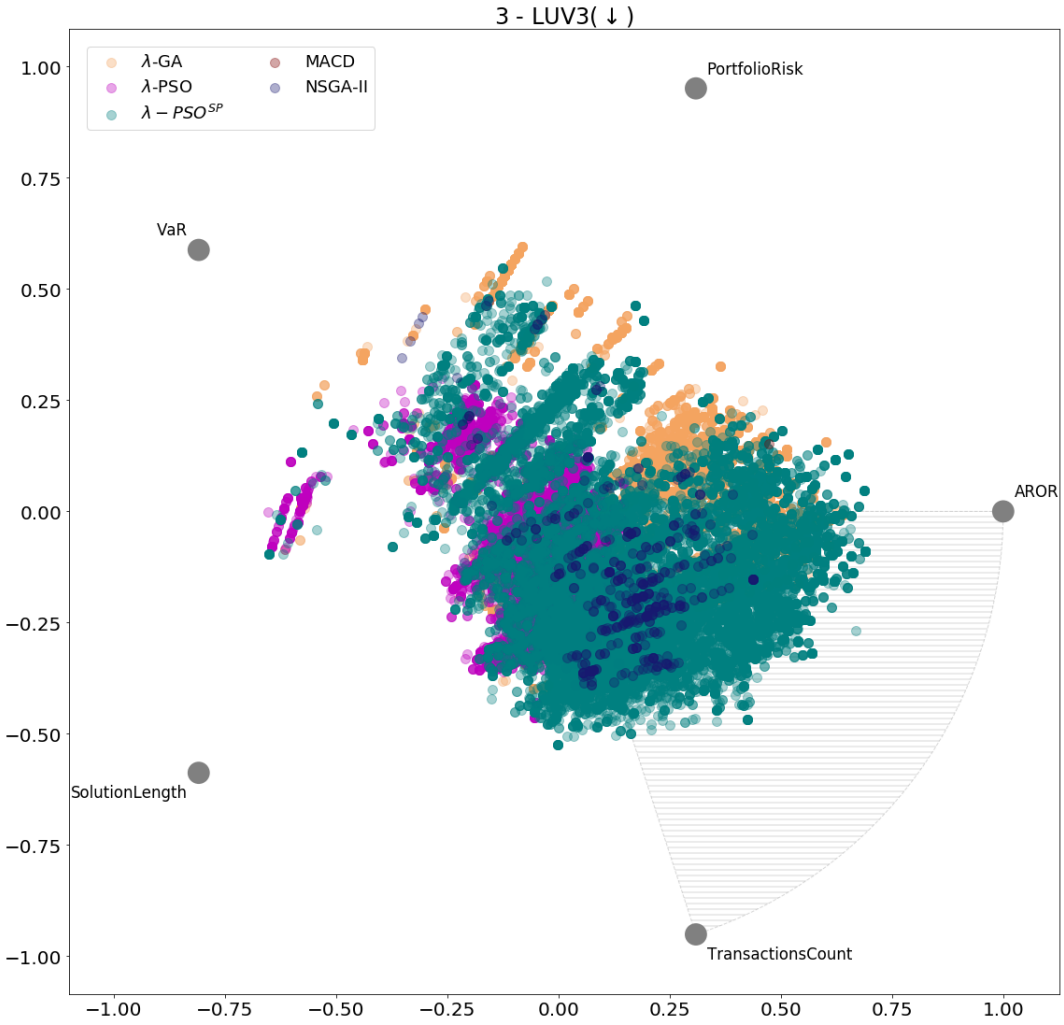

Figure 64: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand LUV3

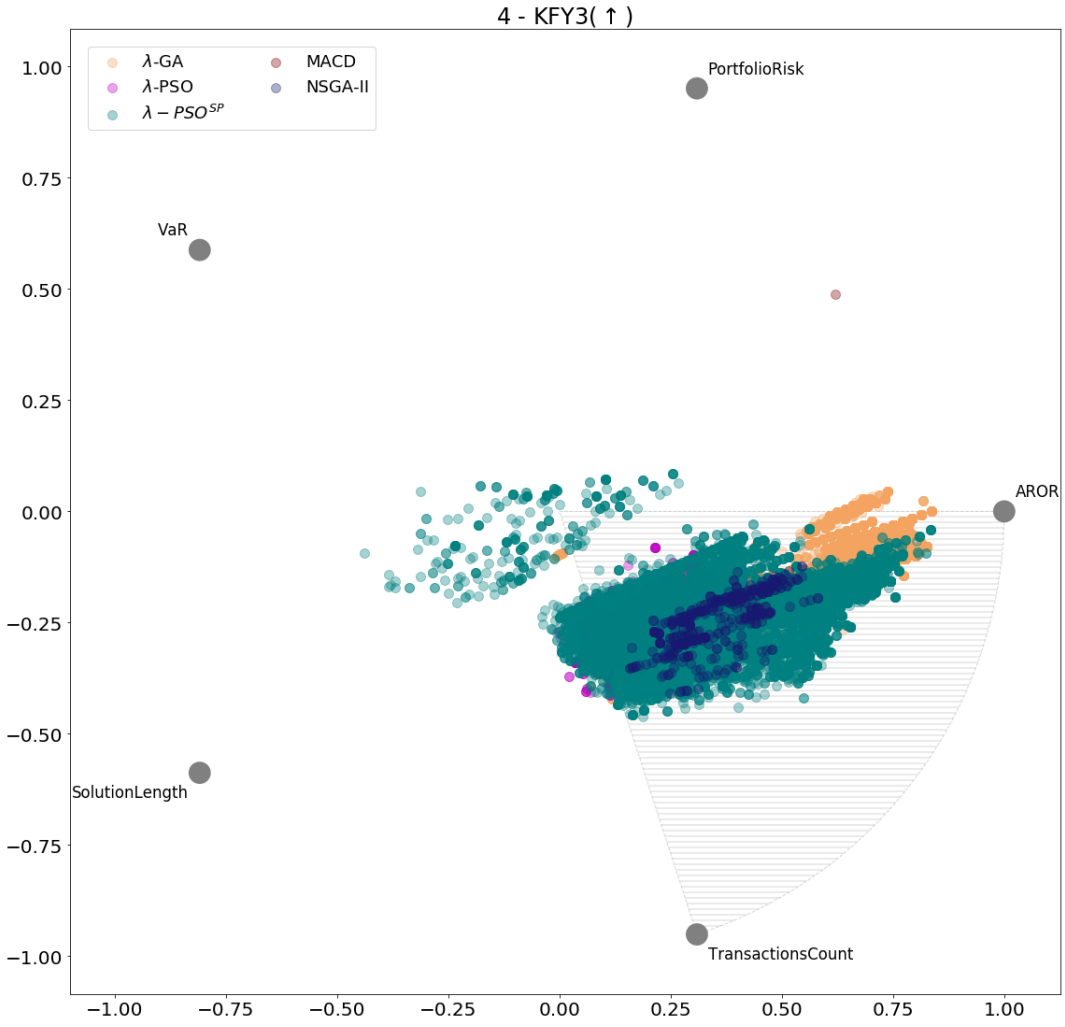

Figure 65: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand KFY3

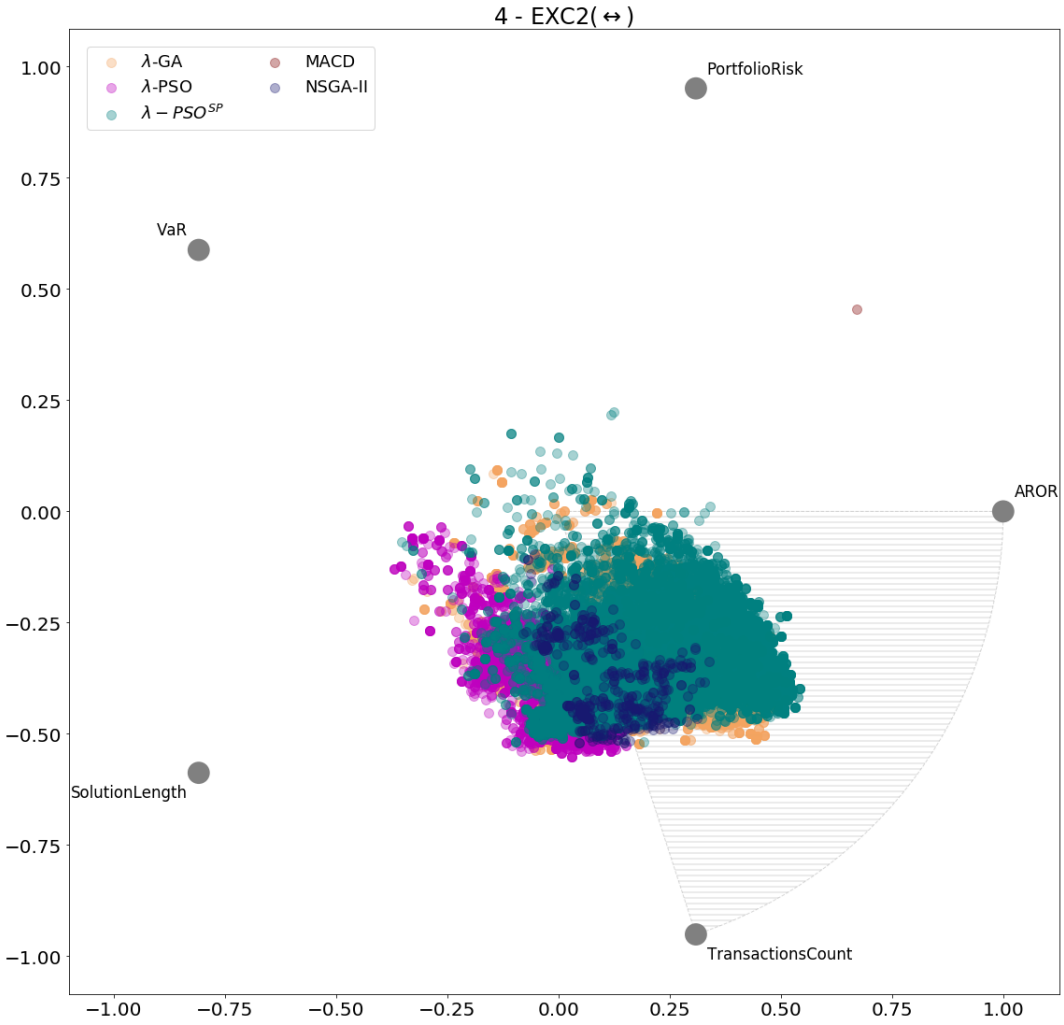

Figure 66: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand EXC2

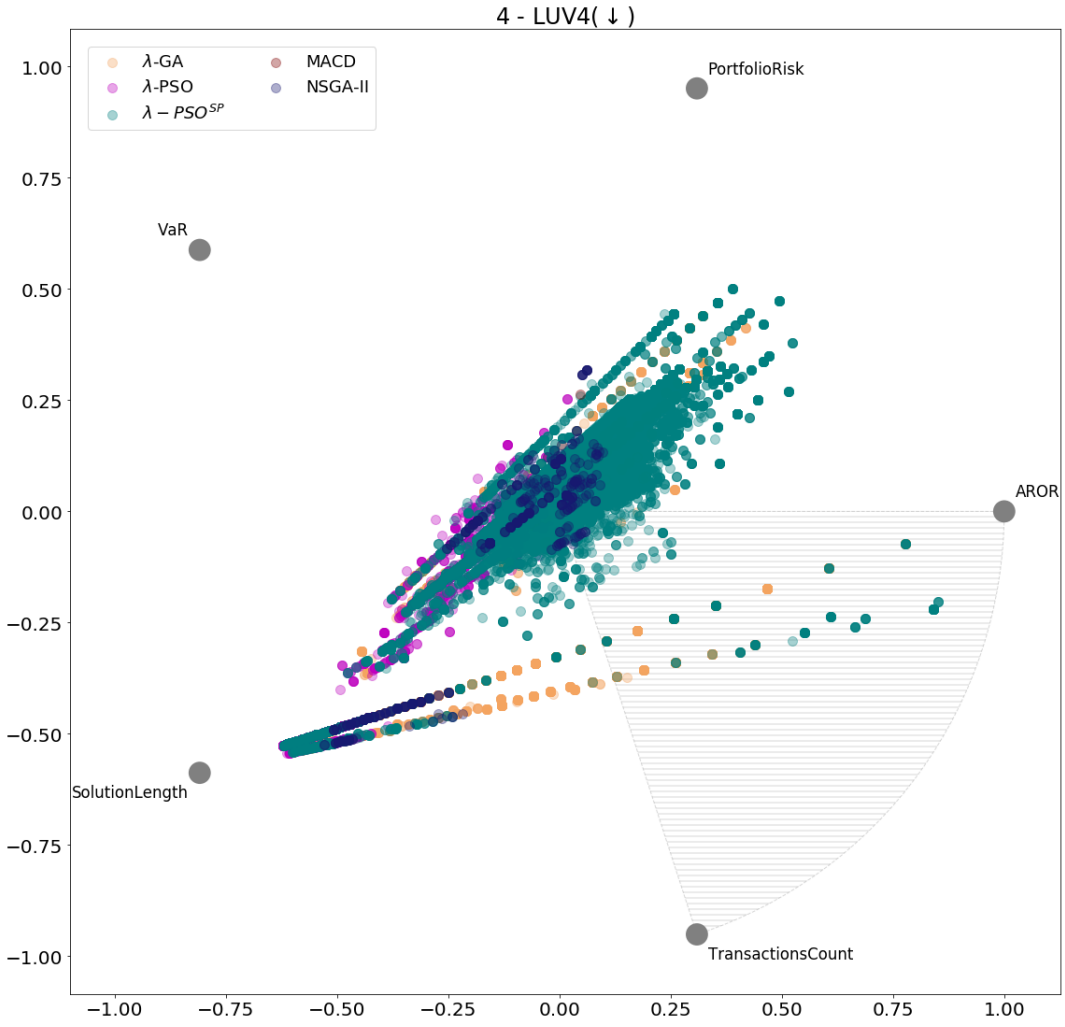

Figure 67: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand LUV4

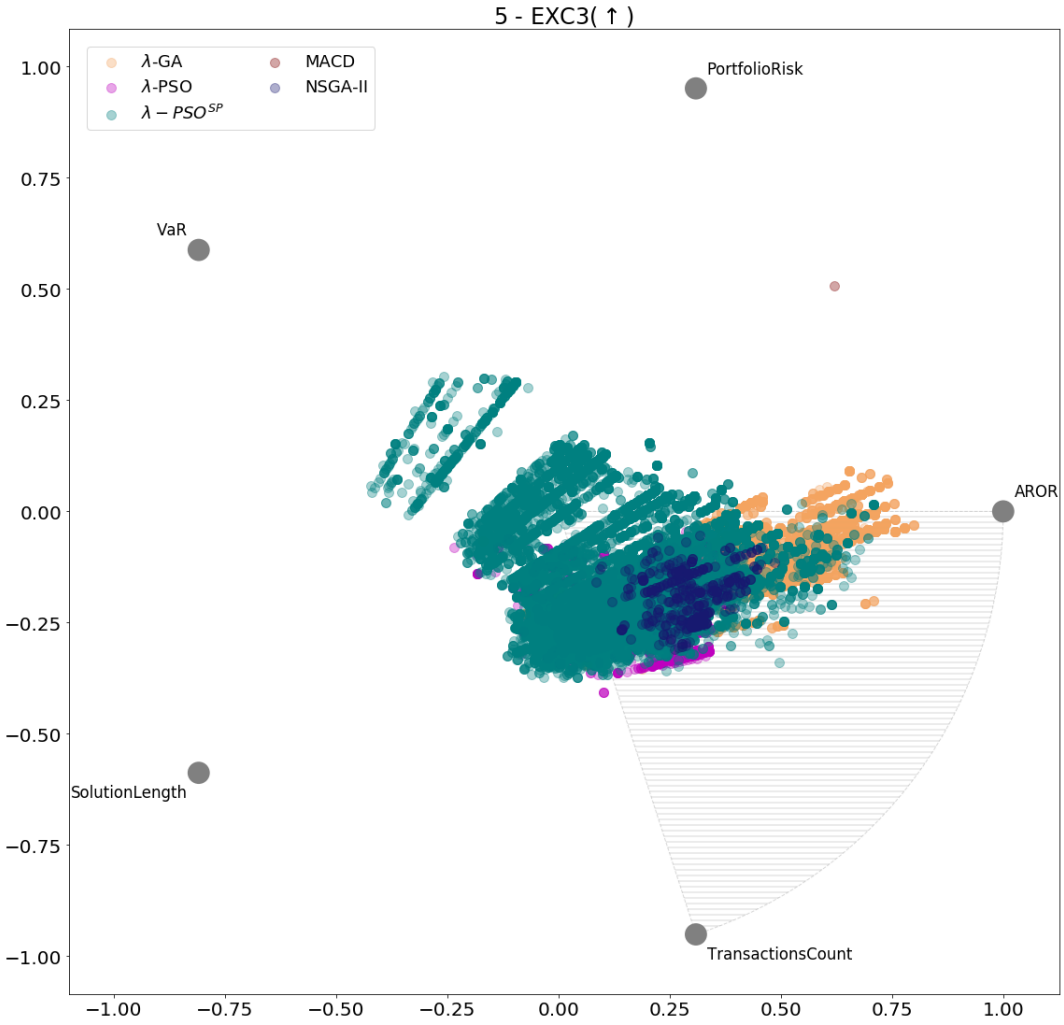

Figure 68: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand EXC3

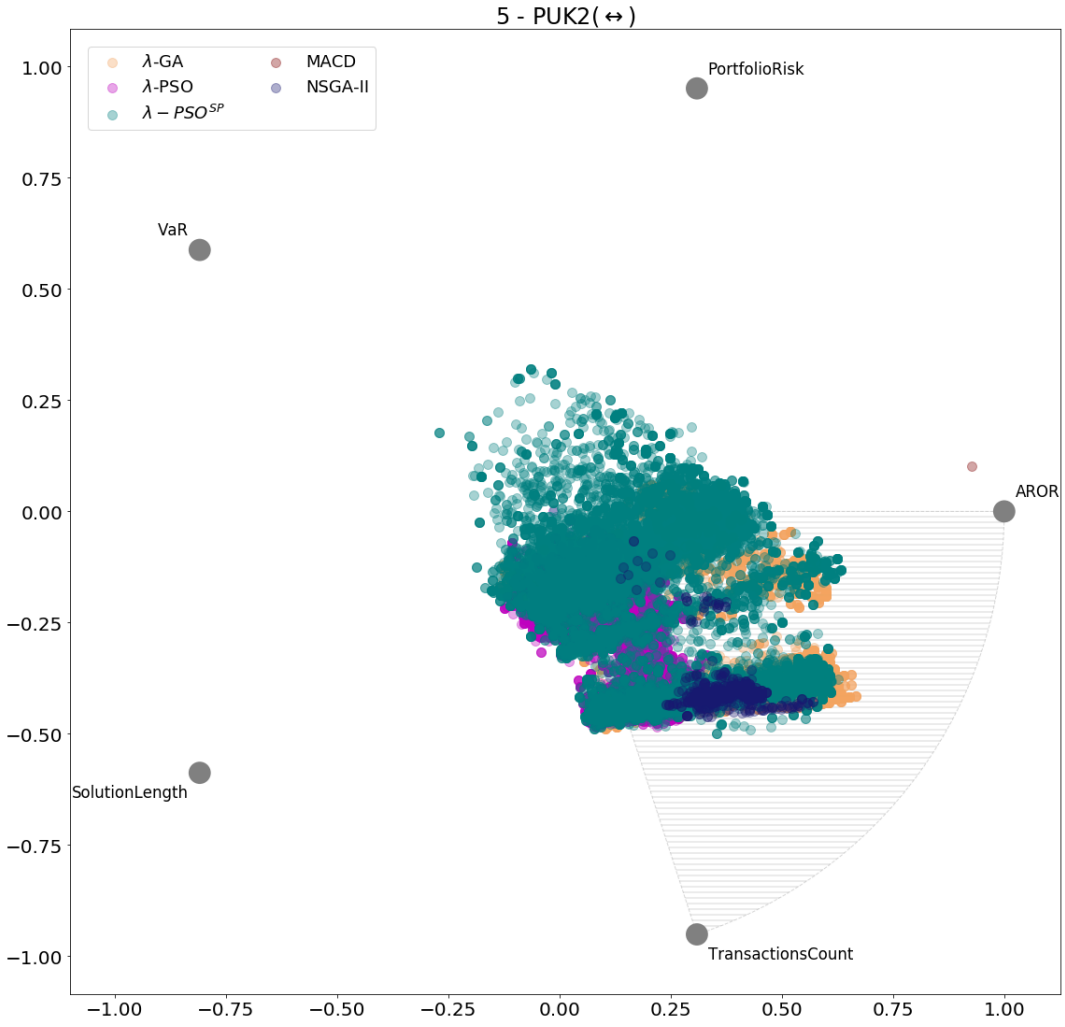

Figure 69: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand PUK2

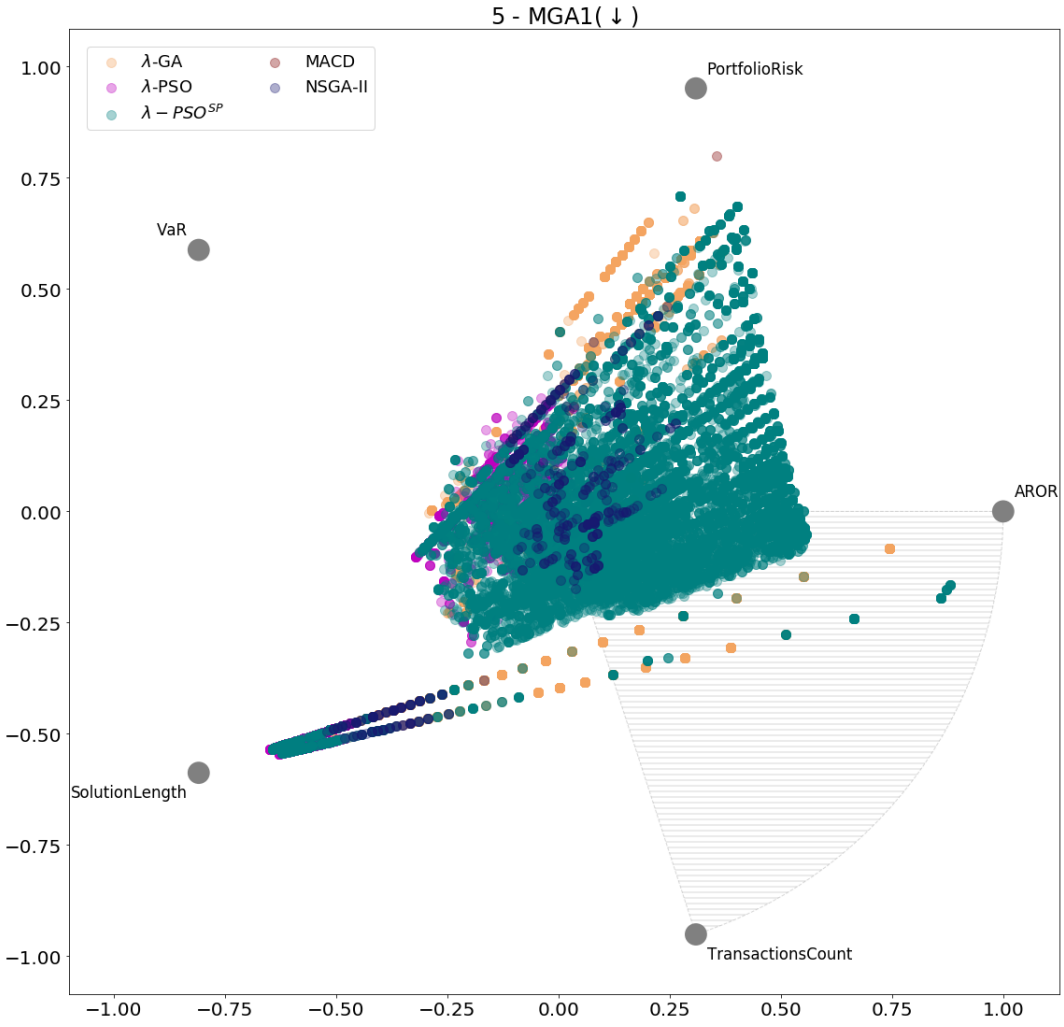

Figure 70: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand MGA1

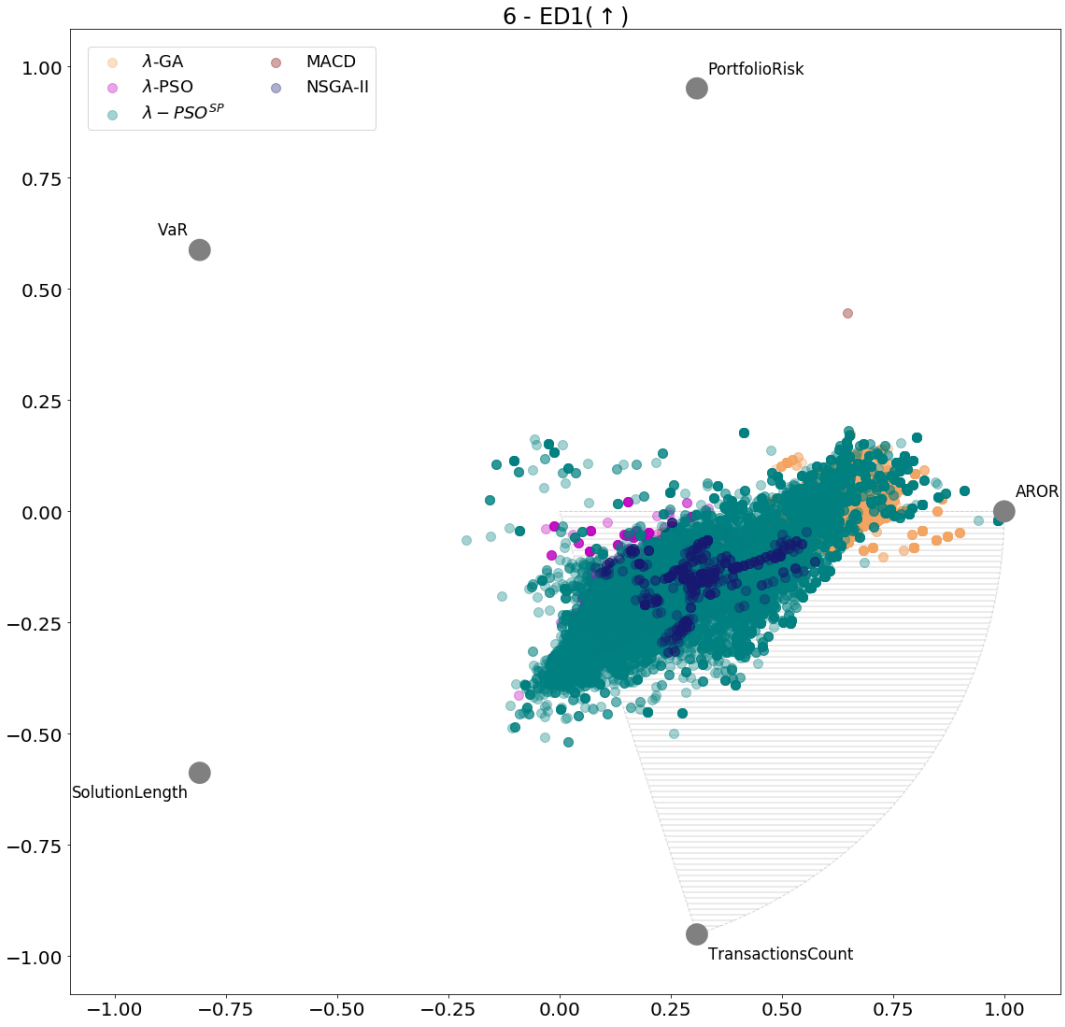

Figure 71: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand ED1

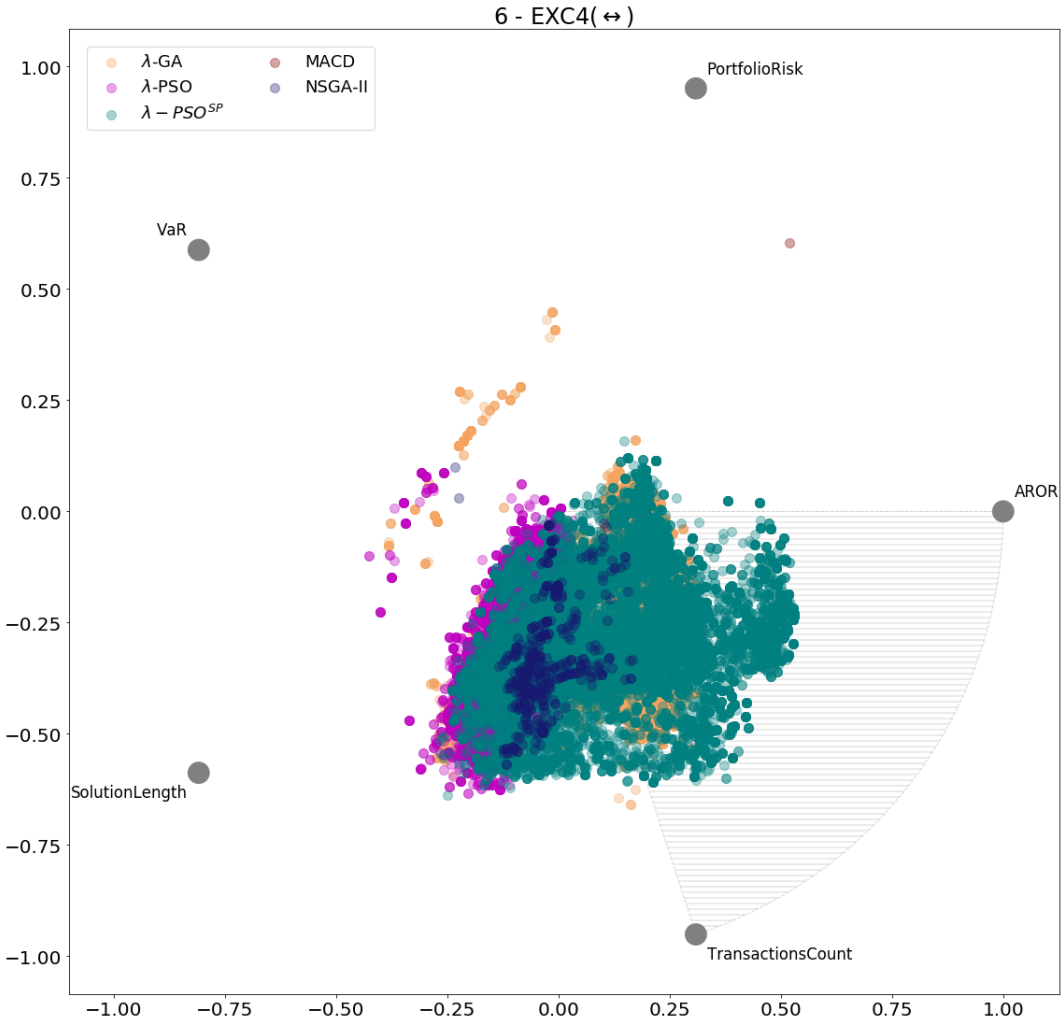

Figure 72: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand EXC4

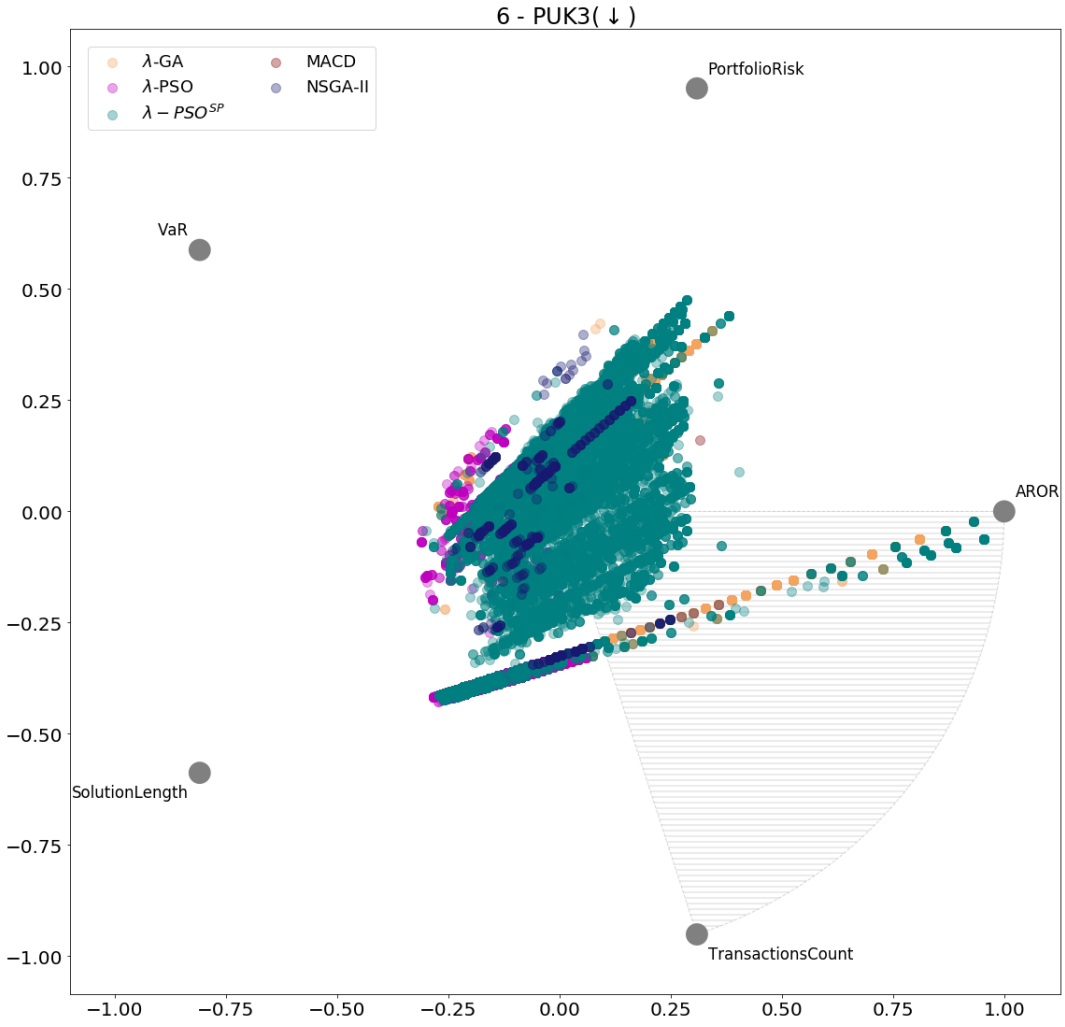

Figure 73: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand PUK3

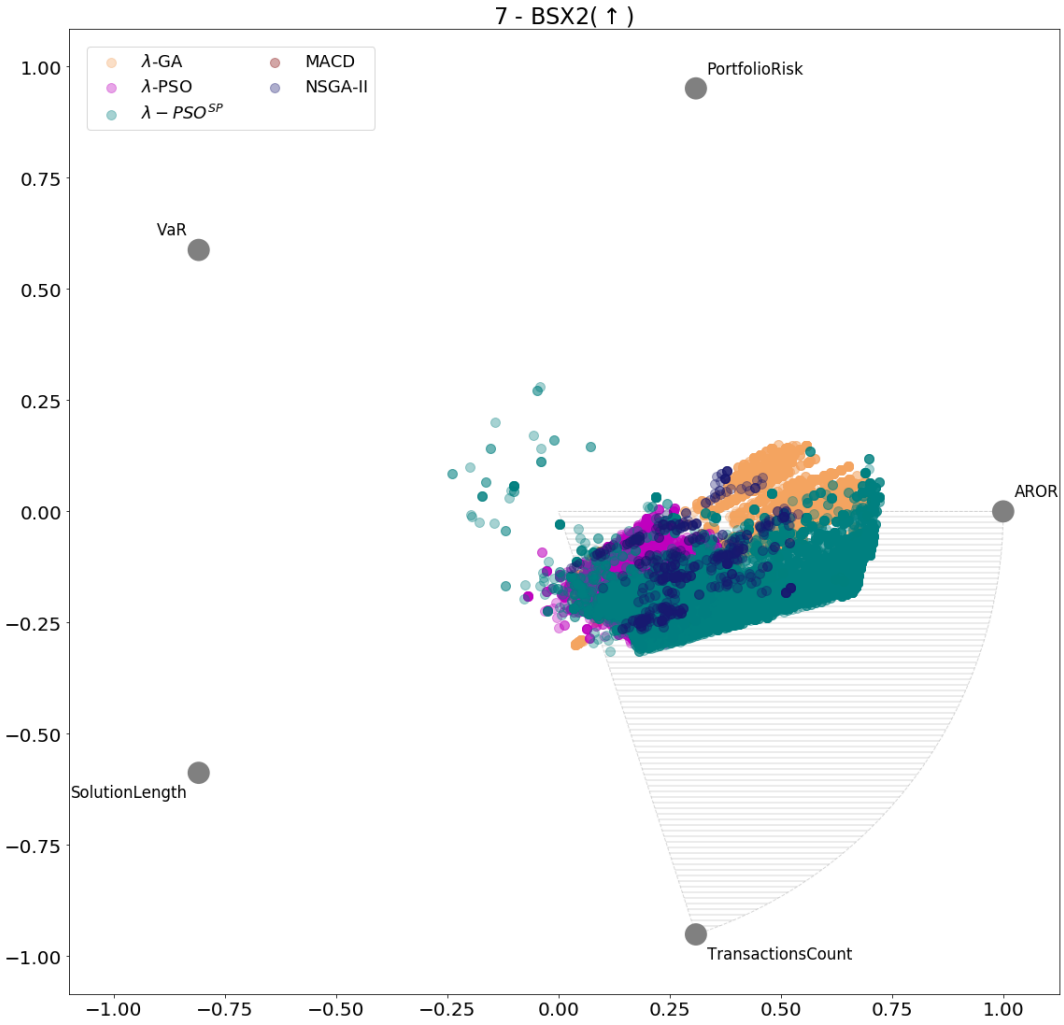

Figure 74: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand BSX2

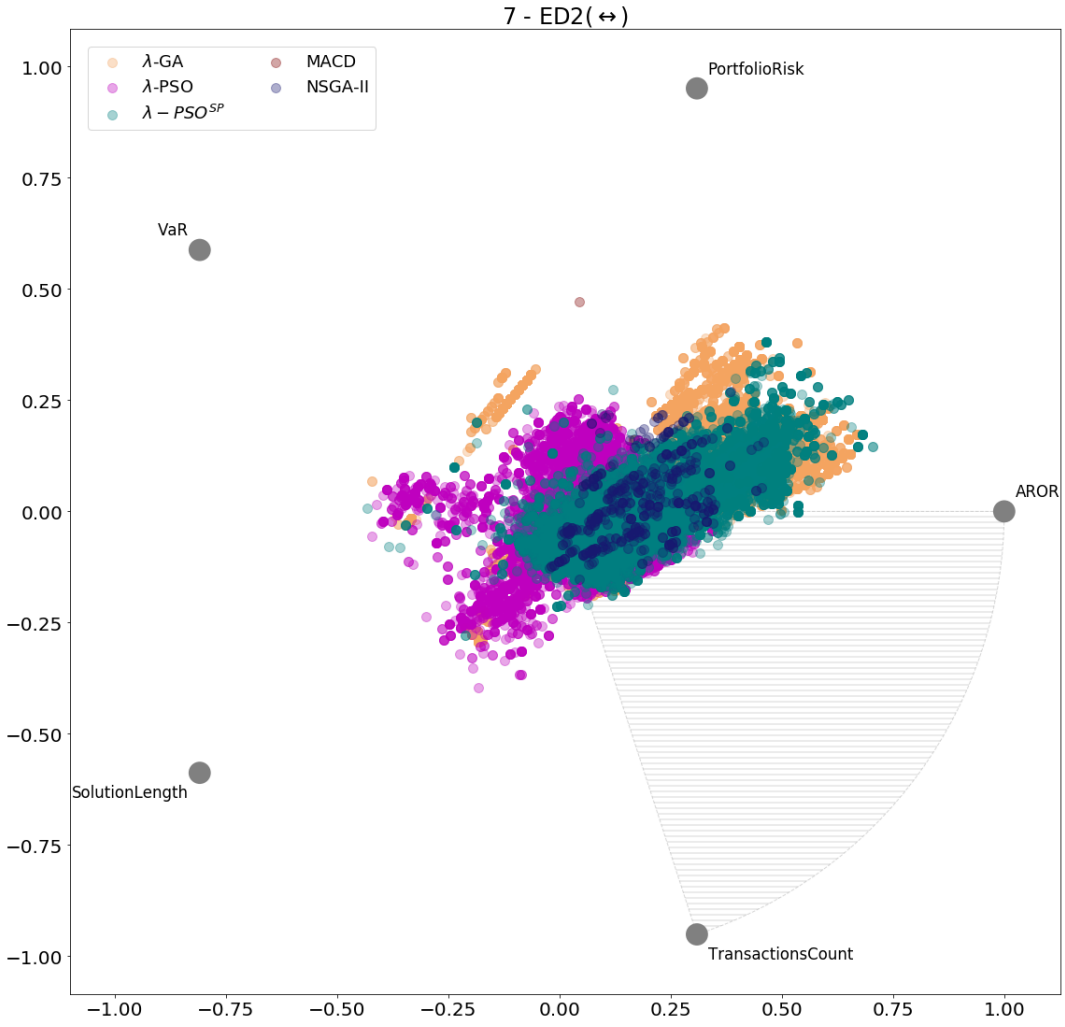

Figure 75: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand ED2

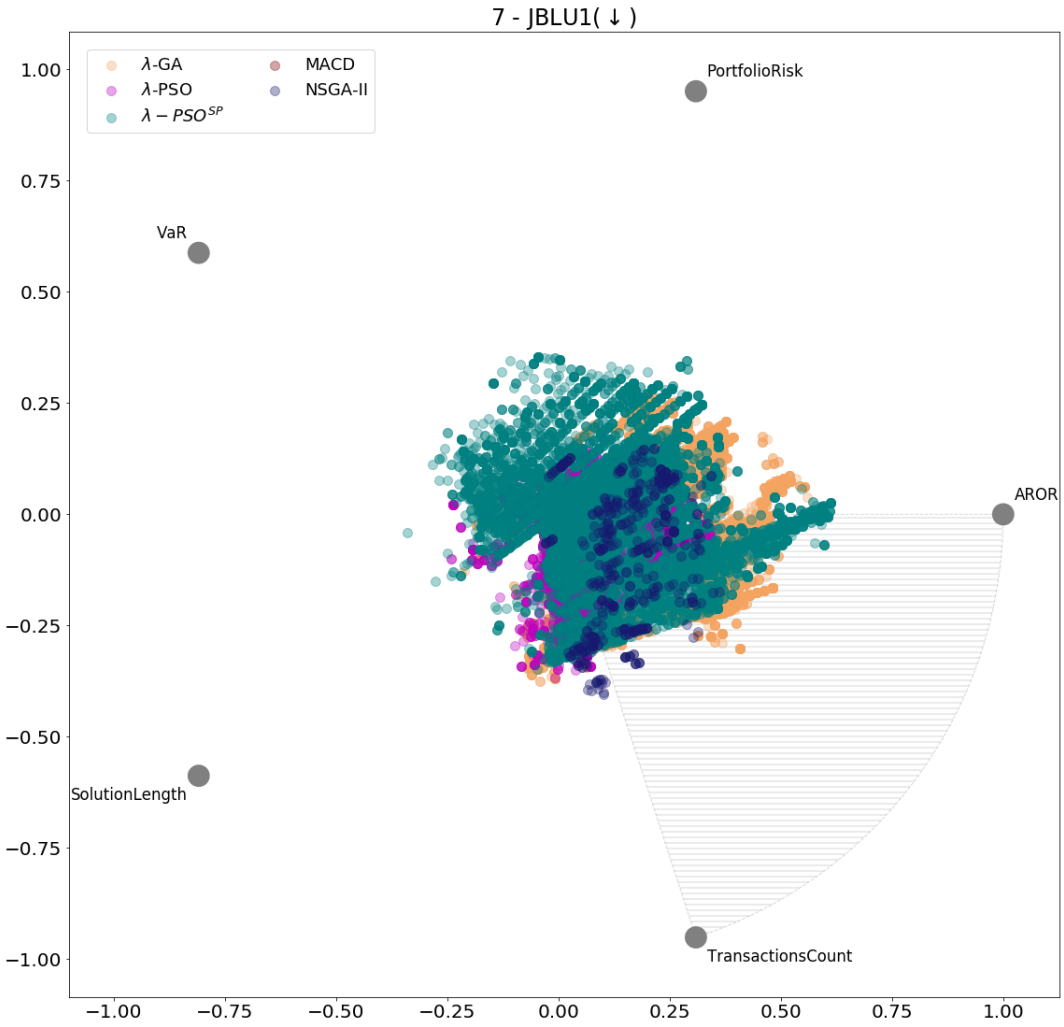

Figure 76: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand JBLU1

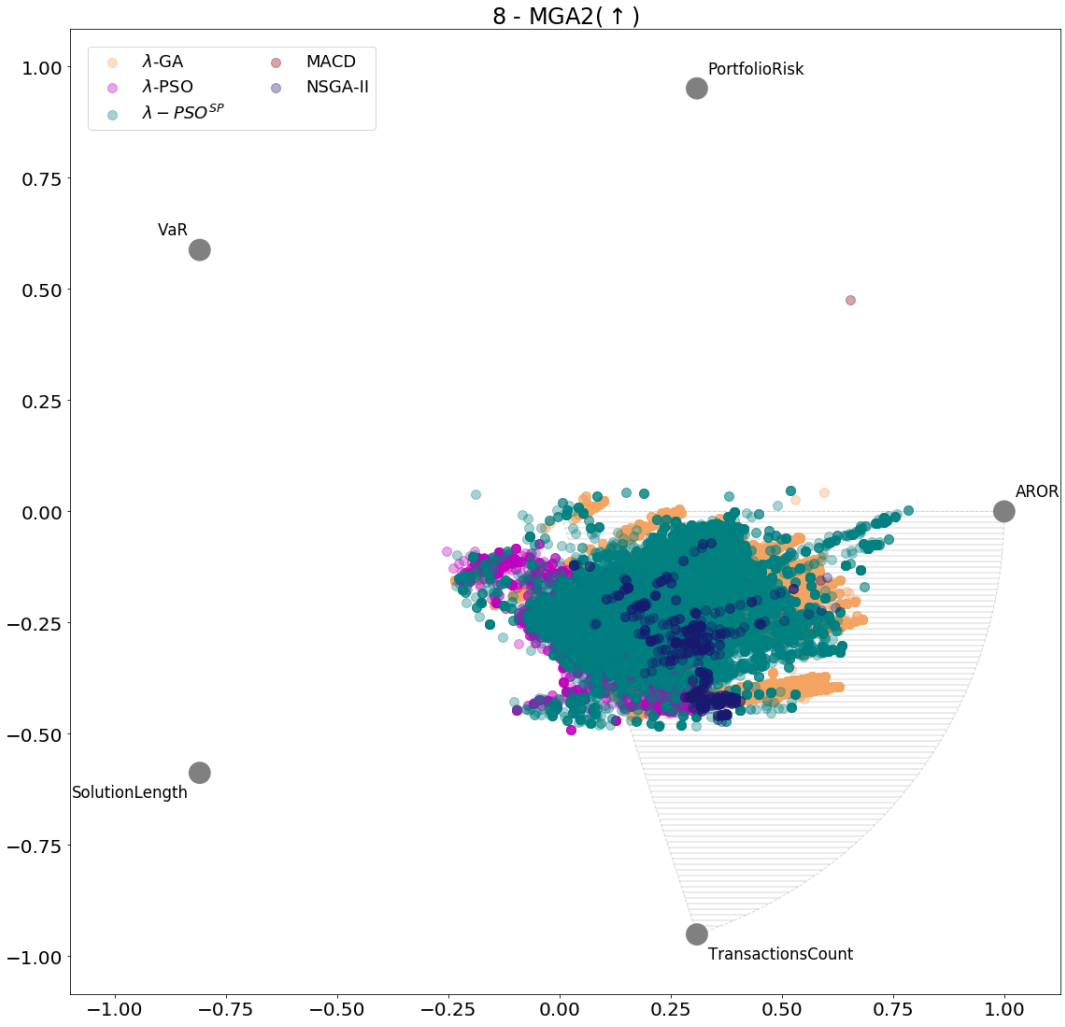

Figure 77: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand MGA2

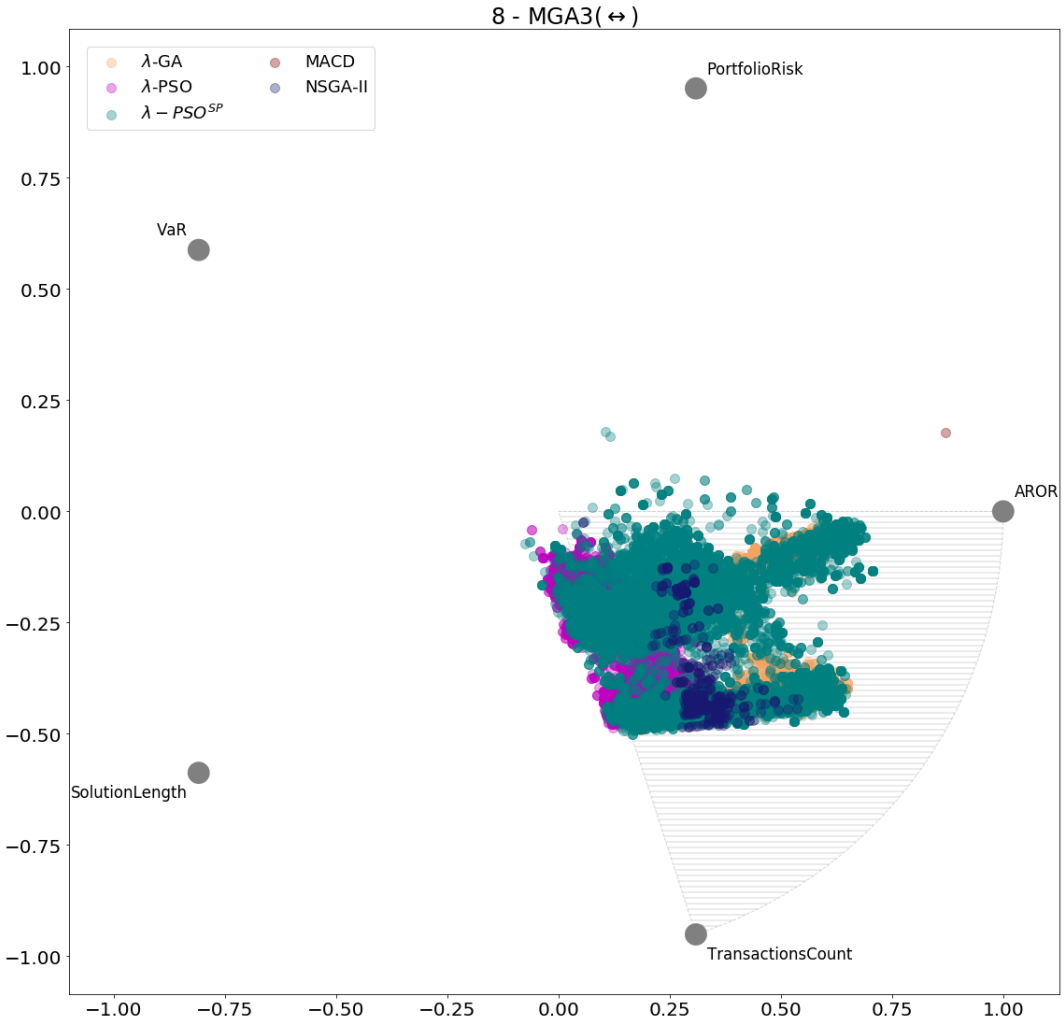

Figure 78: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand MGA3

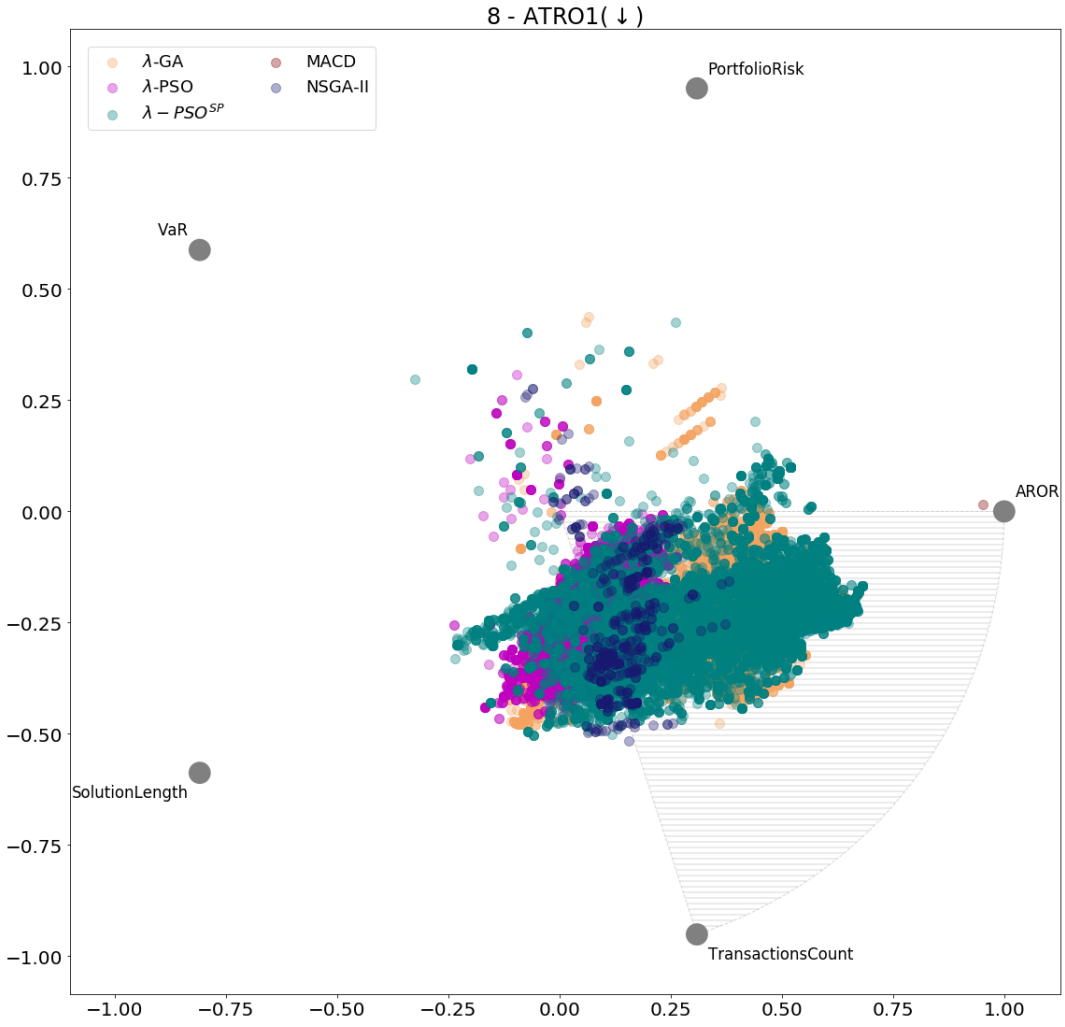

Figure 79: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand ATRO1

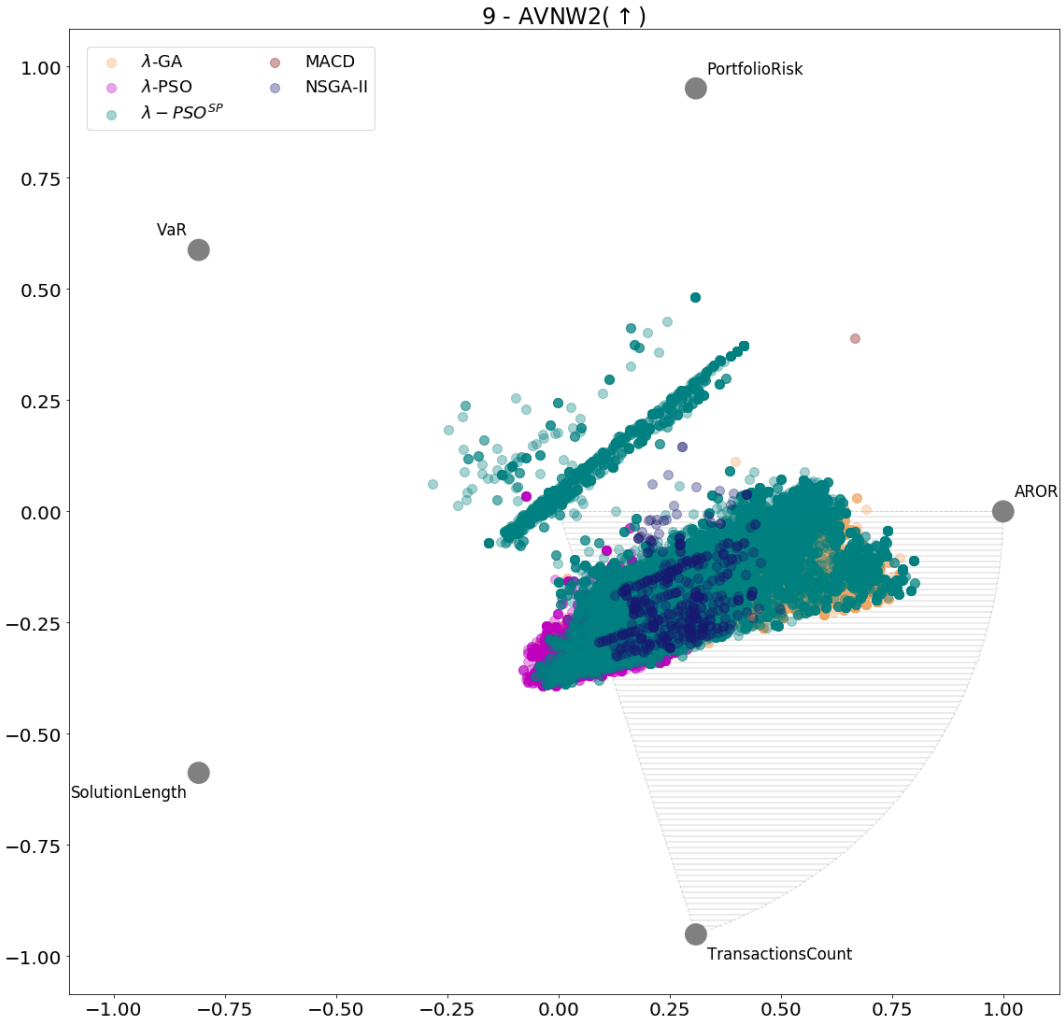

Figure 80: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand AVNW2

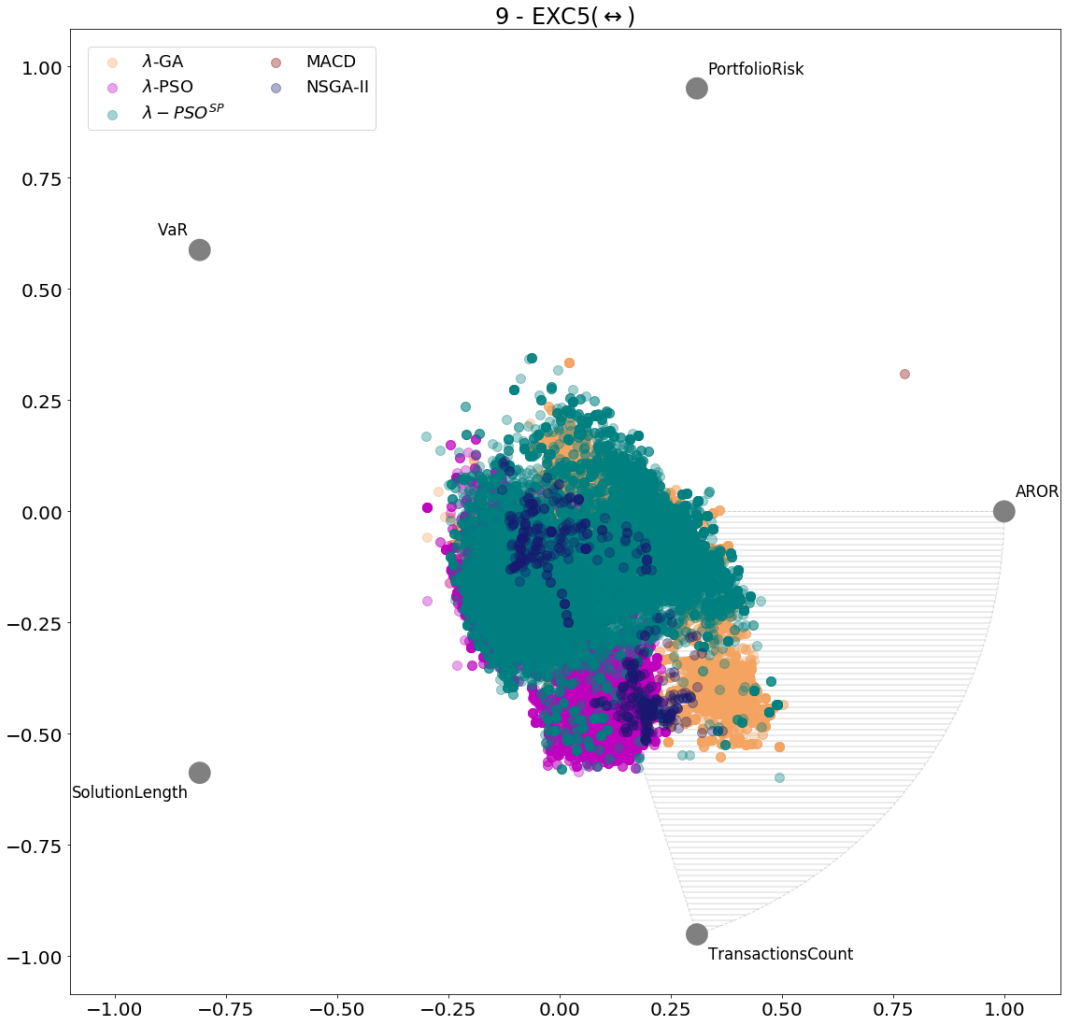

Figure 81: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand EXC5

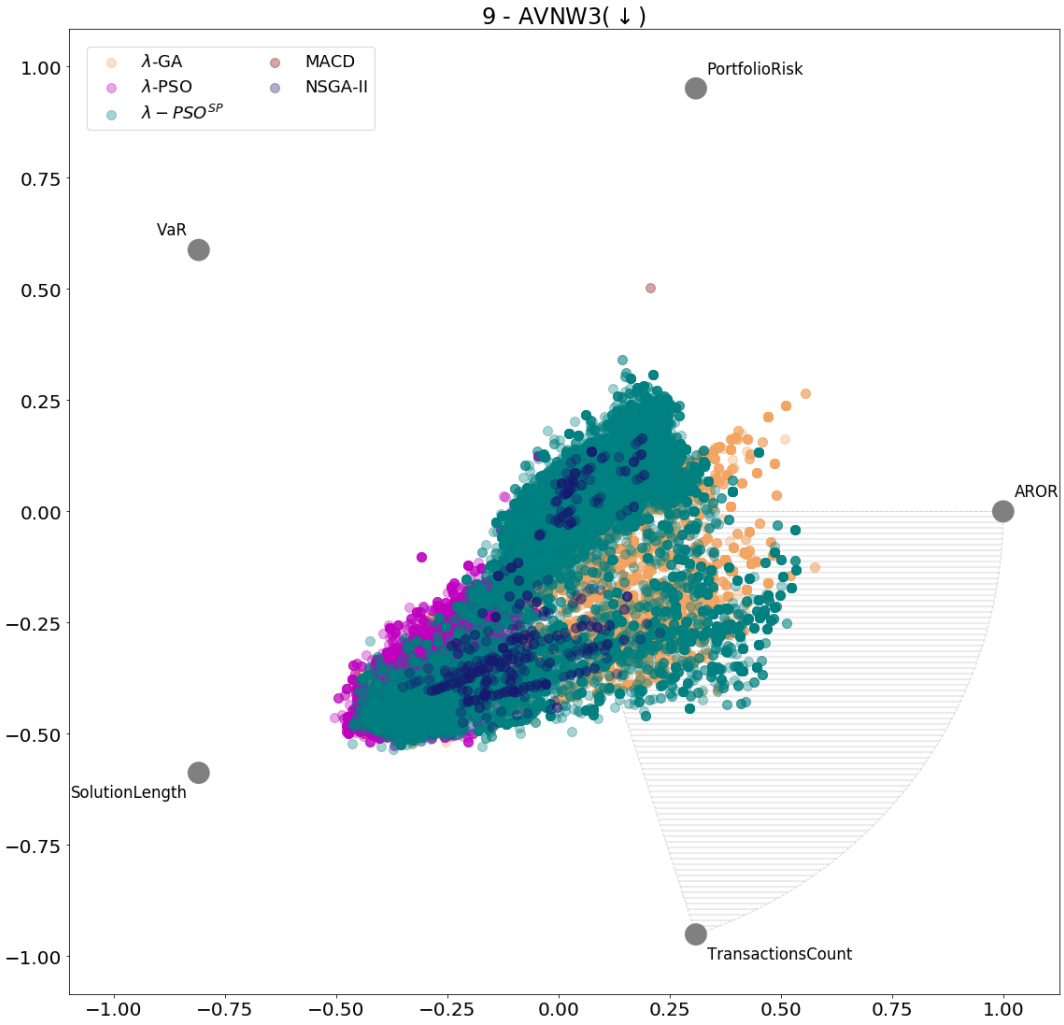

Figure 82: RadViz plot for λ-PSO$^{\text{SP}}$, λ-PSO, λ-GA, NSGA-II and MACD for test strand AVNW3