Martín, C., Quintana, D., Isasi, P. (2019). Evolution of trading strategies with flexible structures: A configuration comparison. *Neurocomputing*, 331, pp. 242-262.

DOI: 10.1016/j.neucom.2018.11.062

Evolution of trading strategies with flexible structures: a configuration comparison

Carlos Martín[a], David Quintana[a,*], Pedro Isasi[a]

[a]*Dpto. de Informática, Universidad Carlos III de Madrid, Avda. Universidad 30, 28911 Leganes, Spain.*

Abstract

Evolutionary Computation is often used in the domain of automated discovery of trading rules. Within this area, both Genetic Programming and Grammatical Evolution offer solutions with similar structures that have two key advantages in common: they are both interpretable and flexible in terms of their structure. The core algorithms can be extended to use automatically defined functions or mechanisms aimed to promote parsimony. The number of references on this topic is ample, but most of the studies focus on a specific setup. This means that it is not clear which is the best alternative. This work intends to fill that gap in the literature presenting a comprehensive set of experiments using both techniques with similar variations, and measuring their sensitivity to an increase in population size and composition of the terminal set. The experimental work, based on three S&P 500 data sets, suggest that Grammatical Evolution generates strategies that are more profitable, more robust and simpler, especially when a parsimony control technique was applied. As for the use of automatically defined function, it improved the performance in some experiments, but the results were inconclusive.

Keywords: evolutionary computation, genetic programming, grammatical evolution, trading

1. Introduction

Recent advances in Information Technology and Communications have favored the progressive automation of trading. The ability of algorithmic trading systems to process data at high speed and to identify regularities within or across markets, has led to their popularization since the 1990s [1]. While they offer advantages in terms of speed; accuracy; reduced costs or lack of human emotions, to mention a few, their impact on the financial markets as a whole is still controversial. Researchers have identified both positive elements, like higher efficiency in the price discovery process, and negative ones like excess volatility and higher adverse selection costs [2, 3].

According to Nuti et al. [4], these systems can cover either specific stages of the process, including pre-trade analysis; trading signal generation; trade execution or post-trade analysis, or

*Corresponding author

Email addresses: carlos.m.fernandez@alumnos.uc3m.es (Carlos Martín), dquintan@inf.uc3m.es (David Quintana), isasi@ia.uc3m.es (Pedro Isasi)

December 20, 2021

the whole cycle. In this work will be focused on trading signal generation, more specifically on market timing strategy generation.

The range of alternatives that can be used in this context is very wide, but most of them fall under one of two categories, fundamental analysis or technical analysis. The former is focused on fair value and uses financial statements as the main source of information, while the latter relies mostly on features of the price and volume data of securities called technical indicators. Among them, the most well known are moving averages and support lines. Besides these, other classic indicators are the opening, closing, maximum and minimum prices, volume and volatility [5]. These, together with a number of operators and thresholds, are used to define technical investment rules. Once these rules are specified, investors can use them to make decisions on the purchase and sale of securities in financial markets.

The matter of whether it is possible at all to find profitable trading strategies for publicly traded assets is still under debate. According to Fama [6] and his followers, stocks trade at fair value. This would make beating the overall market through traded asset selection or market timing on a risk-adjusted basis impossible. This, so called, Efficient Market Hypothesis (EMH) has three variants "weak", "semi-strong", and "strong" that differ on the strength of the claim. However, other authors disagree [7] and there is a whole industry based on the argument that active investing is profitable.

The search for profitable trading strategies has been driving research efforts for decades. Among the many different approaches that can be used to design trading rules (such as Particle Swarm Optimization, Genetic Algorithms, Artificial Neural Networks and Fuzzy Methods), there are some based on evolutionary computation that are especially interesting due to three key features: the process of rule generation is automatic, the resulting rules are interpretable, and their structure is flexible.

Evolutionary Computation (EC) belongs to a family of problem solvers, based on trial and error strategies, that is widely used in finance [8]. This framework includes a number of optimization algorithms inspired by the principles of natural selection. These are based on populations of candidate solutions that are improved over a number of iterations. These include, among others, Genetic Algorithms (GA), Evolution Strategies (ES); Genetic Programming (GP) and Grammatical Evolution (GE). This work will we based on the last two.

GP is a type of Evolutionary Algorithm (EA) that uses a guided random search technique (GRST) for search optimization. GP is a refinement of GAs popularized by Koza [9], who considers that the problem solving process is equivalent to the search (within the space of possible computer programs) of a particular computer program, or highly adapted individual, that solves it.

Over time, researchers have developed a number of variations on the canonical algorithm. Among them, we could mention:

- Tree-based Genetic Programming (TGP)
- Stack-based Genetic Programming (SGP)
- Linear Genetic Programming (LGP)
- Extended Compact Genetic Programming (ECGP)
- Multi-Tree based Genetic Programming (MTGP)

- Multiobjective Genetic Programming (MOGP)
- Cartesian Genetic Programming (CGP)
- Probabilistic Incremental Program Evolution (PIPE)
- Strongly Typed Genetic Programming (STGP)

Grammatical Evolution is an evolutionary computation technique that is closely related to GP. They share the common objective of finding an executable program or a program fragment that will achieve a good fitness value for the given objective function. In GE, Ryan et al. [10] make use of variable length genomes, represented as integer vectors, to elaborate definitions of grammars to generate a large range of programs, of arbitrary complexity, by applying genetic operators.

The body of literature on trading rule generation using either GP or GE is large. However, it is also very heterogeneous. Most of the papers differ in the choice of core algorithms, technical indicators or data samples. The latter is specially relevant as GP/GE-based strategies tend to perform better in some settings (bear markets/sideways trends) and worse in others (upper trends), making performance comparisons over different samples rather problematic. That means that the evidence supporting the superiority of any of these two competing approaches over the other in this domain is very limited. In addition to this, even basic discussion on whether using basic extension like ADFs or mechanisms aimed to promote parsimony makes sense, are still pending. This work intends to fill that gap in the literature presenting a comprehensive set of experiments using STGP and GE with similar variations.

The goal of this work is to compare the two most popular approaches used to generate trading rules from tree-based flexible structures based on evolutionary computation. In addition to that, the study will analyze the importance of ADFs and the impact of the addition of a complexity control mechanism to both GP and GE. The analysis of the results will be based on financial returns and the complexity of the investment rules generated under different configurations.

In order to make the results comparable, both algorithms will use the same set of functions and indicators. In a first approximation, we will consider the set defined by Lohpetch and Corne [11] and a standard population size. This initial work will be subsequently expanded to assess the sensitivity of both algorithms, together with their variations, when we increase the population size and the number of technical indicators used.

The work is organized as follows: section 2 presents a short overview of the relevant literature. Then, section 3 will provide a description of the aspects of GP and GE evaluated and used in this work. That will be followed by a description of the experimental setup in section 4, and the presentation of the main results in section 5. Finally, section 6 will be devoted to summary and conclusions.

2. State of the Art

Academic research on automatic trading rule generation using GP, GE and their variations has a two-decade tradition that started with a seminal paper by Allen and Karjalainen [12]. This section will be devoted to present a number of key papers that will be classified according to the core underlying algorithm.

2.1. Contributions based on Genetic Programming

Most of the literature is based on the canonical version of GP. The starting point would be the contribution of Allen and Karjalainen [12]. These authors sought to generate, through GP, technical trading rules for the S&P 500 index, using daily prices from 1928 to 1995. Their study concluded that the rules generated could not obtain, in the periods of test, consistent excess returns on investment over the naive strategy of buying at the beginning of the period and selling at the end, known as Buy and Hold (B&H), specially after considering transaction costs.

In their paper, Setzkorn et al. [13] created a set of market investment rules that optimized through the use of GP. The set of rules consisted of simple relationships between technical indicators, based on moving averages of different time lengths, from which they generated three types of signals: buy, sell or remain inactive. The data used came from the composite index of S&P 500 (02/01/90 to 18/10/01) that was divided into a training period of 3 years, a validation one of 4 years and a test sample of 4. The results showed that the complex rules behaved very well in training but poorly in the test period, whereas the simple ones did worse during training, but better in the test phase. The authors considered that the most likely reason for the poor performance of the algorithm was the exclusive use of moving averages as indicators, and they thought that increasing the range of alternatives would probably improve the results.

The same year, Thomasy and Sycara [14] published a paper where they evaluated whether financial information from internet news portals (Yahoo and Lycos Finance) had a predictive power that, used as an input to GP, could provide investment returns superior to B&H. In their experiments, the B&H strategy obtained a 126.21% yield throughout the trial period, while the return of the GP alternative was 164.36%.

Becker and Seshadri [15] performed a series of experiments using GP to beat the B&H strategy in the S&P 500 index, taking into account transaction costs and without counting dividends. Their approach differed that of Allen and Karjalainen in that they used monthly data instead of daily returns, increased the number of derived technical indicators, and used a complexity penalty factor in calculating individuals fitness in order to promote parsimony. In their conclusions, they affirm that they beat B&H at the same time that the complexity of the found rules was reduced drastically.

In another study, Neely [16] used GP, following Allen and Karjalainen [12], to construct optimum investment rules bearing in mind the ex-ante risk predicted for the S&P 500 index. The period that they considered ran from 1929 to 1995, with training periods of 5 years followed by selection periods of 2. The best rules obtained in the previous two phases were tested on the remaining data, taking into account transaction costs of 25 basis points. In their conclusions, they found no evidence that the rules obtained significantly exceeded the performance of the B&H strategy on an risk-adjusted basis.

Fyfe et al. [17] used GP to generate investment rules trained from long time series belonging to the S&P 500, S&P Automobile and S&P Bank indexes for the period that goes from 01/01/90 to 07/30/99. The rules were evaluated in terms of their basic yields and adjusted by including risk and operating costs. In the results, they observed that although the basic yields performed very well, in comparison with the B&H strategy, they did not surpass this once the risk adjustment and associated operating costs were considered.

Potvin et al. [18] proposed the use of GP to automatically generate short-term investment rules

in securities markets. Instead of using a compound index, their investment rules were adjusted using individual index values. In their study, they used stock price data and transaction volumes of 14 Canadian companies included in the TSE 300 index. The study period ranged from June 30, 1992 to June 30, 2000, for a total of 2003 days. In their conclusions, they showed that GP failed to beat B&H. However in a detailed analysis of the results, they found that GP outperformed it in 9 out of the 14 indexes.

Navet and Chen [19] investigated GP use on the New York Stock Exchange Market (NYSE). Based on the time series data of 100 index values during the period from 2000 to 2006, and dividing them into three sets of data: training (2000-2002), validation (2003-2004) and test (2005-2006), the authors explored performance of the investment rules obtained by GP, from a classification scheme that distinguished between values with high and low entropy. Results showed that GP only exceeded the random investment strategy in 2 out of 5 simulations. In their conclusions, the authors pointed out that the empirical evidence obtained suggested that predictability is neither a necessary condition nor sufficient to obtain profitability. The predictability test only identifies the existence of time patterns, but gives no further information on the ease, or difficulty, of discovering the patterns.

Lohpetch and Corne [20, 11] sought to identify a training strategy that would allow the generation of investment rules that outperform the B&H strategy robustly. They used a 31 year period of S&P 500 data from 1960 to 1991 (in line with [12, 15, 21]), in which they explored two different regimes for choosing and evaluating a rule after training. In the first strategy, after the training phase, the best rules were tested against the validation set, in the second strategy, the obtained rules were validated, first against a second set of data and then the rules acquired in this second phase were tested against the test set. The authors concluded that the GP was sensitive to the periods of data involved, and it was clearly better to use a validation set following the training one to choose the rule to evaluate. As for the length of the training period, they found that in the shorter periods B&H was dominant, whereas in the long periods the performance was lower (24 months and 18 months).

In the same vein as Potvin et al. [18], Esfahanipour et al. [22] extended the GP model, in two ways: first adjusting the risk and second without considering it. In their study, they implemented their solution taking into account the individual stock indexes of ten Iranian companies on the Tehran Stock Exchange market (TSE). In the GP model, in which the unadjusted risk was applied, the results obtained were similar to those of Potvin et al. In their results, they affirm that in the risk-adjusted GP model the investment rules generated surpassed B&H strategy in all market conditions, whether in the bull market or in the bear market. They concluded that their GP model managed to generate profitable investment rules for all investors, either for those who accept the risk or those who try to avoid it.

In his thesis, Jensen [23] chose a fairly recent sample (1997-2007) of the stock indexes DAX and Hang Seng, which he used to identify investment strategies using GPs in training periods of 3 and 5 years and of test of 1 and 3 years respectively. The author performed some experiments with 10-year training periods that he tested in a one year test. Unfortunately, the results were poor despite the fact that the GP algorithm was trained with a data set that included a complete economic cycle. In his conclusions, he affirmed that the rules of investment generated by the GP generally produce negative results in terms of yield and, more important, Sortino ratio. The author understands that these key figures may imply that GP fails in its attempt to beat B&H strategy. This was specially the case once he used a realistic transaction cost of 0.25%, instead

of the 0.10% that he used as a starting point.

Janice How et al. [24] used the GP algorithm procedure of Allen and Karjalainen to find technical trading rules. They chose four 7-year overlapping (in-sample) estimation periods (1979–1985, 1984–1990, 1989–1995 and 1994–2000) of daily data used as input to the algorithm. To avoid data overfitting, each in-sample period was broken down into a 5-year training period and a 2-year selection one. The rules generated were then tested out-of-sample over the remainder testing period, which always ended up on 31 December 2005. Like in Brock [25], Allen and Karjalainenl [12], Potvin et al. [18], they found that the rules obtained tended to signal the trader to stay in the market during periods of low volatility and positive returns, and stay out of the market during periods of high volatility and negative returns.

Gabrielsson et al. [26] explored the feasibility of evolving transparent entry and exit trading strategies, for the E-mini S&P 500 index futures market, in a high-frequency trading environment using GE. They compared the performance of models incorporating risk into their calculations with models that did not. In their conclusions, they affirm that the empirical results obtained suggest that profitable, risk-averse, transparent trading strategies for the E-mini S&P 500 can be obtained using GE together with technical indicators.

Luengo et al. [27], based on the application of a set of simple trading rules optimized by GP, looked for a method for generating input and output signals in the Spanish stock market under three different market scenarios: bull market, bear market and sideways market. In their results, they found that market global behavior had a great influence on the results of each method. Strategies based on GP performed best in sideways markets.

In their paper, Gypteau et al. [28] used an intrinsic time scale based on directional changes, which they combined with GP, to find an optimal trading strategy to forecast the future price moves of financial markets. They evaluated its efficiency and robustness as forecasting tool through a series of experiments and concluded that, with their approach, they we were able to obtain valuable information about forecasting performance.

Hongguang et al. [29] used an intraday time series with GP to fully exploit the short term forecasting advantage of technical analysis. They thought that the utilization of intraday data to train the trading rules could avoid jumping points in daily or monthly data, as relevant information is assumed to be fully digested during the market close periods. Although, their results showed that the trading rules generated by GP were able to accumulate trading profits within a certain period (morning session data are most likely to be exploited profitably in the afternoon session of the same trading day), the trading strategies trained with multi-day data could lead to severe losses in several of the following trading days. In their opinion, this indicates that trading trend tended to vary in different trading days and stayed the same in a single trading day. As input data, they used 21 days of Hushen 300 index future data, covering the time period from 29/09/14 to 03/11/14.

Pimienta et al. [30] proposed a computational system that combined a conventional market method (technical analysis), GP and multiobjective optimization to generate an automated investment system. It is interesting to note, that they put forward a mechanism for automatic detection and removal of outliers, in order to minimize distortions in the evaluation of candidate buying and selling rules. The proposed automated investment system was applied to six stocks (BBAS3, BOVA11, CMIG4, EMBR3, GGBR4, and VALE5) from BOVESPA, in a test window of 514 working days, between 02/05/13 and 02/02/15. In the results, they show that their system

was able to obtain financial returns considerably above the stock variation price on the same period, and it outperformed other two automated investment strategies. In addition, it was able to obtain significant profit even in situations of strong depreciation of the asset.

Yang et al. [31] presented a framework for developing a sentiment feedback strength based trading strategy using GP. In their study, they chose the S&P 500 ETF as a representation of the U.S. broad market performance, collecting daily historical return of these indexes through a Bloomberg terminal from July 31, 2012 to January 30, 2015. They found, that the sentiment indicator based on a GP approach yielded superior market returns, with low average monthly maximum drawdown, over the period analyzed. When they compared Sterling ratio, and other risk measures, the proposed sentiment indicator based strategies was superior to the technical indicators and the traditional B&H strategy.

2.2. Contributions based on variations of Genetic Programming

Even though the bulk of the literature is based on the standard version of GP, several authors relied on variations that extend its capabilities or mitigate some of its limitations.

Mousavi et al. [32], for instance, used GP to develop a dynamic portfolio trading system to capture dynamics of stock market prices through time. The proposed approach takes an integrated view on multiple stocks and generates a rule base for dynamic portfolio trading, based on the technical indicators. In their research, they developed a multitree GP forest in order to extend the GP structure for extracting multiple trading rules from historical data. Based on the Iranian and Canadian stock exchange markets, their results showed that the proposed model significantly outperformed other traditional models of dynamic and static portfolio selection, in terms of portfolio return and risk adjusted return.

More recently, two papers presented solutions based on STGP. This approach imposes data type constraints and uses generic functions and data types, which overcomes some issues related to the closure requirements of the canonical version of GP.

Manahov et al. [33] developed profitable stock market forecasts for a number of financial instruments and portfolios using a special adaptive form of STGP. The STGP-based trading algorithm produced one-day-ahead return forecasts for groups of artificial traders. The in-sample period in the experiment consisted of 11,405 daily observations (24 May 1962–14 September 2007) for the S&P 500 index and IBM and General Electric stocks. The out-of-sample period included 1515 daily observations (17/09/07 to 17/09/13). In their conclusions, they pointed out that they found little support for the Marginal Trader Hypothesis but some evidence in favor of the Hayek Hypothesis.

Agapitos et al. [34] used a memory-enabled program representation in STGP and compared it against the standard representation of a GP in a number of financial time-series modelling tasks. They based their experiments in data from S&P 500 EUR/USD and Nikkei and used a complex function set that included mathematical functions, boolean logic, memory access, relational operators and conditional sentences, terminal sets with technical indicators and ERCs. They concluded that memory-enabled programs generalize better than their standard GP counterparts in most data sets of the problem domain.

Finally, it is worth mentioning the contribution of Berutich et al. [35], which presented a robust PG approach to discover profitable investment rules that were used to manage a portfolio

of Spanish securities. The authors explored a Random Sampled Fitness GP method (RSFGP). The core algorithm is basically like the standard GP but instead of calculating fitness over the complete data set, relies on randomly selected segments with the aim of increasing robustness. In their conclusions, they explained that the RSFGP system was capable of dealing with different types of markets, achieving a return on the Spanish investment portfolio of 31.81% for the 2009-2013 test period, compared to 2.67% of yield obtained by the IBEX35 index.

2.3. Contributions based on Grammatical Evolution

The number of papers published applying GE in this domain is more limited. This not surprising, as the body of literature on GP in general is also significantly larger.

Brabazon and O'Neill [36] explored the possibility of using this instrument to generate investment rules aimed at beating a B&H reference strategy on the money market. Their results outperform the benchmark on five out of the six test sets using a small set of technical indicators, a metric to penalize commercial risk, and a relatively low amount of foreign exchange data from the London market from 10/23/92 to 13/10/97.

Dempsey et al. [37] used a GE-based evolutionary automatic programming methodology, to discover technical investment rules targeting the S&P 500 and Nikkei 225 indexes. The authors explored two approaches, one with a single population of rules that adapts throughout the data time series, and another by which a new population is created for each generation step. In their study they obtained positive returns, with clear advantages in the case of the adaptive population of rules. However, in the S&P 500 index, they found that there were very few opportunities to beat the reference index. On the other hand, in the Nikkei 225 index, GE generated investment returns with an average improvement of 74% over the index.

In their paper, Contreras et al. [38] presented a GE-based trading system. They tested its performance on historic returns for a set of companies of the Spanish market using 2012 data. In addition to that, they compared the GE system with a previous work [39] where the authors applied a GA-based approach. In their results, the trading system implemented with GE obtained a 14% return, while the GA-based alternative obtained a loss of approximately 20%. The analysis, with an extended set of nine selected companies from Spain, showed that the general profitability was greater than the B&H strategy.

Schmidbauer et al. [40] constructed and tested a framework for trading rule selection that curbs the data-snooping bias of performance evaluation. At the core of their approach was the concept of a-priori robustness, in which a trading rule performing well on the original time series of prices should also perform well when exposed to an alternative scenario, exhibiting similar features but without being identical to the original time series. They deployed an evolutionary computation tool based on a grammar guided GP for the selection process and a multi-objective fitness criterion which involved the original as well as modified time series. The method was tested for FOREX trading of Euro/USD exchange with intra-day data from February to June 2011. Their findings suggested that a-priori robustness criterion gives less spurious results, and that prevents in-sample overfitting. Although, they found evidence that out-of-sample profit could be increased, they could not yet achieve profitability.

Table 1, summarizes the most important information, as it classifies the mentioned pieces of research according to the core algorithm, GP or GE. It also shows whether the authors used a

complexity control mechanism other that a mere constraint on the size of the individual trees, be it in terms of complexity or depth. As it as mentioned before is apparent that the most popular core algorithm is GP, which is used in 20 out of the 24 considered works. The use of parsimony promotion strategies is only reported in two papers of the sample, and it is remarkable that no one explored the convenience of adding ADFs. All these possibilities will be benchmarked in this paper.

Table 1: Related works. EC strategies and techniques used

Related Work	GE	GP	Parsimony	Related Work	GE	GP	Parsimony
Allen and Karjalainen(1999)		X		Esfahanipour et al. (2011)		X	
Setzkorn et al. (2002)		X		Contreras et al. (2013)	X		
Thomasy and Sycara (2002)		X		Schmidbauer (2014)	X		
Becker and Seshadri (2003)		X	X	Gabrielsson et al. (2014)		X	
Neely (2003)		X		Mousavi et al. (2014)		X	
Brabazon et al. (2004)	X			Gypteau et al. (2015)		X	
Fyfe et al. (2005)		X		Luengo et al. (2015)		X	
Potvin et al. (2006)		X		Manahov (2015)		X	
Dempsey et al. (2006)	X			Hongguang et al. (2015)		X	
Navet and Chen (2008)		X		Agapitos et al. (2016)		X	
Lohpetch et al.(2009)		X	X	Berutich et al. (2016)		X	
Jensen (2010)		X		Pimienta et al. (2017)		X	
How (2010)		X		Yang et al. (2017)		X	

3. Genetic Programming and Grammatical Evolution

3.1. Genetic Programming

GP is an stochastic optimization metaheuristic where a population of programs (individuals) is iteratively improved according to an objective function also known as fitness function.

Even though the algorithm has a number of limitations, like its computational cost, or the fact that it cannot guarantee global optima, it has very important advantages. Among these, we could mention the fact that it is easily paralleled; works with large solution spaces and complex fitness landscapes; it is easy to adapt it to different problems and, above all, it offers interpretable solutions.

Most of these traits are shared with other evolutionary computation alternatives like genetic algorithms, evolution strategies or differential evolution, to name a few. However, it offers a key advantage vs. the others when it comes to trading strategies: the structure of the rules doesn't have to be set in advance. Instead optimizing parameters of predefined rules, GP evolves the whole rule.

The representation of an individual, in this case an investment rule, is the approximation followed to construct it.

GP traditionally represents individuals as syntactic tree structures. Trees have the advantage that they can be easily evaluated recursively and have a direct correspondence with the s-expressions of Lisp. Each non-terminal node of the tree has an operator function and each terminal node contains an operand. The mathematical expressions contained in the tree are simple, it is easy to evaluate and it is easy to apply different genetic operators to the tree itself.

Figure 1 shows an example of an investment rule tree representation whose expression-s is:

(Or (>Mx2 M2) (Or (>Minimum UR) (<M10 LR)))

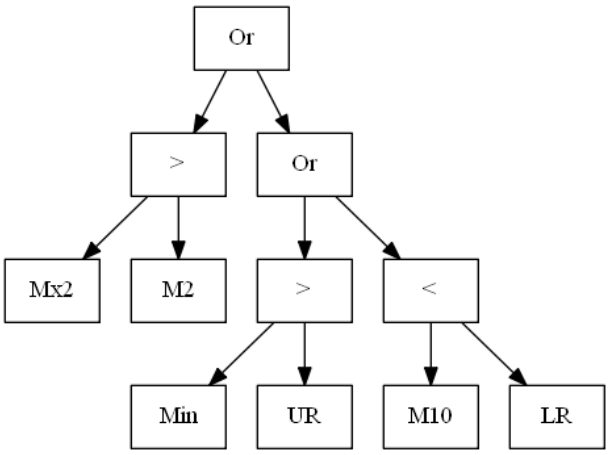


Figure 1: Trading Rule with its Tree Representation

That can be easily interpreted as: whenever the two months average (M2) is below the second resistance indicator of the maximum moving average of the last 3 months (Mx2), or when the minimum of the session is greater than the upper trend line (UR), or the 10 months average (M10) is lower than the lower trend line (LR), then the best strategy is to enter the market and otherwise to leave it.

Figure 2, illustrates the behaviour of the rule during 2007. There, we see the evolution of the S&P 500 index. The lighter line represents whether the rule suggests being in the market or in cash. As we can see, the rule recommends staying out of the market for about the first month, and then the indication varies over time. According to the strategy defined by the rule, the investor should invest during the last month. The fact that the rule suggests a limited number of transactions contains the costs, hence increasing the chances of beating the index.

The basic process behind GP is initializing a population of individuals from a set of available primitives, then evolving it for a number of iterations until a stopping criterion is met and, finally, return the best individuals. The internal steps of the algorithm consist of: a) evaluation of candidate solution in terms of a fitness function, b) selection of a set of individuals of the population following some criterion based on the fitness function and c) creation of new individuals through the application of certain operators (crossover, mutation, reproduction) to the previously selected individuals.

As for initialization in GP, like in other evolutionary algorithms, the individuals that make up the initial population are generated randomly[1].

[1]Initialization does not have to be always random, a GP tree can be initialized from a non-randomized point filling

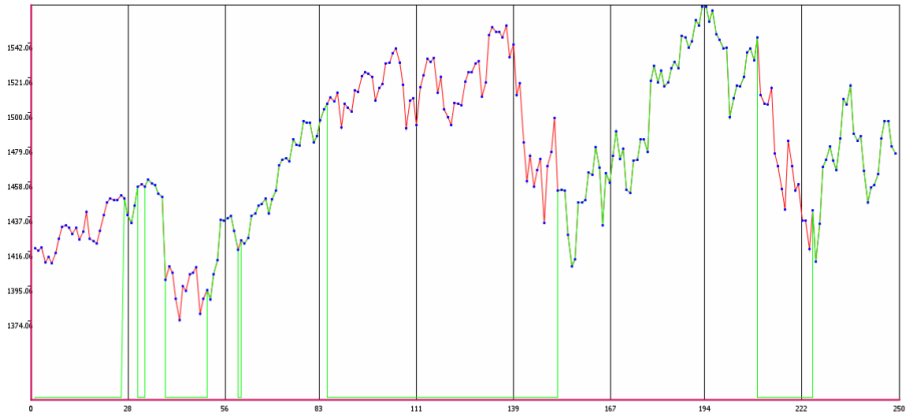


Figure 2: Behaviour of Trading rule (Or (>Mx2 M2) (Or (>Minimum UR) (<M10 LR))) during 2007

The choice of the algorithm and initialization mode is important since, for GP to function effectively, it is required that most of the function sets meet an important property known as a closure, which in turn can be broken down into the consistency property of types and evaluation safety. The consistency of types is necessary because the subtree-type crossover can mix and bind nodes arbitrarily during the evolutionary process. As a result, it is necessary that any subtree could be used in any of the argument positions of each function in the function set, since it is always possible that a subtree crossover generates that combination [42].

In this work, we will use the uniform initialization algorithm. This approach, introduced by Bohm and Geyer-Schulz [43], gets its name from the fact that the tree generated is uniformly exact. The algorithm requests the size of the tree to be generated, and guarantees that it will create a uniformly chosen tree from the complete set of all possible trees of that size, given a specific set of functions.

The process of generating candidate solutions for new populations is driven by the application of mutation and crossover operators. These are complemented with the reproduction operator, that selects an element from the existing population and copies it into a new one.

In the implementation that we use for the experiments, the application of genetic operators is managed in a breeding pipeline. This pipeline is a chain of selection and breeding operators whose function is to draw individuals from an existing population to generate individuals for a new one. Given a split of probabilities for crossover, mutation and reproduction, the process picks randomly an operator and supplies candidate solutions for the next generation.

In each generation, the final population has the same size as the starting population. Replacement follows a (μ, λ) strategy [44]. We begin with a population λ, assess the fitness of individuals and replace all individuals of the population except the most fit μ. To complete the number λ, each of the μ individuals has to produce μ/λ new individuals by mutation. μ represents the surviving parents and λ the children they produce. The substitution operation ends with the replacement

the initial population with an individual who, though not a solution, is believed to be a good starting point. Such seed may have been produced by a previous GP or user-built. Aler, Borrajo, Isasi [41]

of the discarded individuals with the newly created children. In (μ, λ) the number of offspring created is far greater than the number of parents.

We use a an enhanced version of genetic programming (STGP) [45], which enforces data type constraints. STGP permits crossover and mutation of trees only with the constraint that each node's return type "fits" with the corresponding argument type in the node's parent; further, the root node's return type must "fit" with the tree type.

However, most node-building algorithms require that for each accessible type used by the function established on the node, there must be at least one terminal (ideally, a non-terminal one) in the set of functions that returns that type. This is due to the fact that many algorithms have the requirement of being able to generate any terminal, regardless of the current situation of the type where the terminal has to be hung, to generate nodes.

3.2. Grammatical Evolution

This EC technique introduced by Ryan et al. [46] is closely related to GP. The aim and the main loop are basically the same, but there are differences in representation and the way solutions are generated and updated.

In GP, the genotype of an individual is represented by an s-expression in a tree structure, and it is directly manipulated by genetic operators. This same tree is subject to the evaluation function and, therefore, phenotype and genotype are basically the same thing. In GE, on the other hand, there is a clear difference between these two. GE genotypes are vectors of integers (codons) that encode the selection of rules from a predefined context free grammar, and to which traditional genetic operators, such as mutation and crossover, are applied. The phenotype, however, is usually represented by a tree structure that is evaluated recursively. In order to assess fitness, a genotype-phenotype mapping relationship is established. Here, the trees are not intended for the individual's genotypic representation. They operate as temporal structures used in the course of mapping and subsequent evaluation of individuals.

As already mentioned, in Koza's conventional GP style, the set of functions must meet the closure requirement: all functions must be able to accept as arguments the output of all other functions, both terminal and non-terminal. In GP this is achieved using a single data type, such as the use of double precision floating point type. This limitation is not suffered by GE, as the problem is managed by grammars usually defined through formalisms expressed in Backus-Naur form (BNF). This strategy allows the search space to be restricted by incorporating domain knowledge of the problem and facilitating an efficient type control.

GE representation affects the type of problems that can be optimally solved. According to Thorhauer and Rothlauf [47], the representation introduced significant biases. In their study, GE shows a bias towards narrow and deep-type tree structures that outperform standard GP when optimal solution conform to them. Conversely, solutions that require denser trees are easier to solve through GP.

Regarding the initialization of the population, in GE it is based on a grammar definition in BNF format that describes the syntax of the type of program used to solve the problem together with the set of terminals and non-terminals to be used for this purpose.

From there, the construction algorithm generates random vectors of integers, each one in the

range from 0 to a maximum value defined by the number of productions of the grammar rule with the largest number of them minus one. Each position of the vector indicates the rule of the grammar to be selected after applying to its value the modulus (remainder operation) defined by the number of possible values of the rule considered.

Every grammar rule R_i represents its individuals as an element belonging to a finite group in $\mathbb{Z}_{ni}$, where n_i is the number of possible productions of the rule i. One aspect to be taken into account is that the generated vector of integers will contain values that exceed the module and, therefore, are part of the equivalence class of one of the elements of the finite group. For example:

$$117 \equiv 5 \pmod 7 \text{ belongs to the equivalence class } |3|$$

Due to this characteristic, different integer vectors can correspond to exactly the same individual with the same mapped corresponding representation tree.

Therefore, all these individuals form part of an equivalence class of an individual |s|, which is represented by the equivalence classes of the modules of the grammar rules production associated with each position of a vector.

Let's take as an example of a GE vector that represents the trading rule depicted in Fig. 1 and the corresponding grammar in BNF form of table 2.

Table 2: Grammar used in Fig. 1

Nº	Modulus	Grammar Rule
1	1	<Rule> ::= <bool>
2	5	<Bool> ::= (**And** <bool> <bool>)| (**Or** <bool> <bool>)
		<Bool> ::= (**Not** <bool>)
		<Bool> ::= (**>** <exp> <exp>) | (**<** <exp><exp>)
3	16	<Exp> ::= (**Opening**)|(**Closing**) | (**Max**) | (**Minimum**)
		<Exp> ::= (**M2**) | (**M3**) | (**M5**) | (**M10**)
		<Exp> ::= (**Roc3**)| (**Roc12**)
		<Exp> ::= (**Mx1**) | (**Mx2**) | (**Min1**) | (**Min2**)
		<Exp> ::= (**UR**) | (**LR**)

Suppose we got the following pseudorandom sequence of integers ranging from 0 to 255:

064 176 183 171 196 211 073 067 222 129 007 031

To build the tree representation, we take the first number of the integer vector; 064, and we calculate its $\pmod 1$ associated with the root grammar rule, so the rule <bool>is taken. This is a non terminal rule so we take the next integer, 176, which in $\pmod 5$ is 1 and therefore the second production of bool is chosen (Or <bool><bool>). Or is already a non terminal function and is placed in the root of the individual tree. The children of the "Or" node have to be built as well, so we take the next integer 183 which in $\pmod 5$ corresponds to 3, and the rule (><exp><exp>is taken. With the next integer, 171, we select the expression (Mx2) a terminal element appended to the tree. The process continues building the tree in preorder, until all the rules are expanded or the integer vector is exhausted.

The latest integer vector is an isomorphic equivalent to the canonical individual formed by the numbers:

000 001 002 011 004 001 002 003 014 003 003 015

Outside of the fitness assessment context, in which individuals are represented as trees, individuals are simple binary strings to which the standard operators of an AG [48] are applied, like crossing and mutation by a single or multiples points, with the difference that these individuals are composed of chromosomes that are vectors of integers of variable length. Therefore, the operators in GE can be implemented very efficiently, and the performance penalty when making the assignment is also low. However, the syntactic and type constraints imposed by evolutionary grammars achieve this efficiency at the expense of a high proportion of invalid individuals. This problem is addressed in mutation by wrapping the chromosome and interpreting it in a circular fashion and, in the case of crossing, applying it on the codon limit, instead of an arbitrary position. Both mechanisms can greatly reduce the number of invalid individuals [10, 49].

Operators in GE have a different effect from traditional GP, which tend to preserve most of the individuals structure, and are designed to have a predictable effect. In GE, on the one hand, minor changes in the genotype can lead to massive changes in the phenotype. On the other hand, changes in some parts of the genotype may have no effect on the phenotype.

To compensate for the problems introduced in crossing and vector mutations, GE introduces two repair strategies within the genotype itself.

The first strategy, called duplication, is based on selecting a sequence of integers from the vector itself, from two calculated random indexes, and adding it to the end of the vector. The second, truncation, is used to determine how many integers of the chromosome vector are consumed in the creation of the tree, and truncate the rest of integers of it that have not been used. We use both mechanisms.

In terms of applicability to automatic trading rule generation, GE and GP would be equivalent, as they have the potential to provide the same solutions. They both can generate tree-like functional structures based on combinations of predefined building blocks.

3.3. Extensions

The two core algorithms discussed above have extensions in common designed to improve their performance. We will consider two aimed at promoting simplicity and modularization among solutions.

One of the characteristics of GP, derived from the variable size representation nature of individual's trees, is that individuals tend to grow rapidly with generations in a process known as bloating. This feature has several undesirable effects: the first one is that individuals are slower to evaluate, the second one is that it makes individuals more difficult to interpret. In addition to that, it provides poor generalization and in some cases the solutions tend to be very far from the optimum.

A second major problem of complex solutions is overfitting. In the context of trading strategy generation, larger solutions offer a higher potential to model more complex relationships. This often results in rules that offer very high performance on training samples that, unfortunately,

do not generalize well. There are many scenarios where simpler structures provide much better results on new data.

GP traditionally limited individuals setting an upper bound on the maximum depth. This approach is too restrictive and the tendency today is punishing the most complex individuals to reduce the likelihood of selection by introducing a penalty on their fitness value. This is called parsimony pressure and can be established in two main ways:

- Linear Parsimony Pressure. It takes into account the aptitude value and complexity of individuals. Complexity is introduced in the calculation of individual aptitude with a penalty effect. The problem with this approach is that the correct balance of aptitude and complexity must be known in advance.
- Non-parametric Parsimony Pressure. It does not consider the aptitude value and complexity of individuals, but it chooses one according to who is more apt and less complex. An example is the Lexical Parsimony Pressure. It is introduced in the selection operator so that, in case of a tie in the fitness value, the least complex individual wins the tournament.

In our work we will use the latter mechanism of complexity control.

Finally, automatically defined functions (ADFs) are a standard way of creating reusable programming modules both in GP [50] and GE [51]. According to Ferreira [52], the motivation behind the implementation of ADFs in GP, is the belief that ADFs allow the evolution of modular solutions and, consequently, improve performance [53].

ADFs are usually implemented as additional trees associated with the individual main one. This means, that individuals consist of several trees that define a structure of functions in which certain trees call other trees that act like subfunctions. ADFs evolve in parallel to main trees and may or may not be called. When an ADF function is called, firstly its children are evaluated, then their return values are saved and, finally, the corresponding ADF tree itself is evaluated. ADFs are constrained by the number of arguments that they take. This parameter is defined a priori, and cannot be changed during evolution.

4. Experimental Design

As we mentioned in the introduction, the aim of this work is doing a comprehensive benchmarking exercise of the two most popular algorithms for trading rule generation based on evolutionary computation. This exercise will cover, not only the core algorithms, GE and GP, but also the impact of adding ADFs and complexity control mechanisms. Finally, we will study the sensitivity of the different configurations to population size and the function set.

In this section we describe the experimental setup, including all the aspects related to parametrization of algorithms, data used, and the statistical significance evaluation protocol.

4.1. Trading rule representation

The trading rules generated by the algorithms will provide binary recommendations. As it will be discussed later, they will suggest either being in the market, earning market return, or out of the market, earning the risk-free return. Therefore, only long positions will be allowed.

Generating the structures at the core of the specific encoding used by GP and GE requires detailed discussion of two components: the basic building blocks, and the combination rules. The former requires describing the function sets and the terminal sets, while the latter entails introducing the grammar to be used.

At a later stage the individuals will be encoded as parse-trees in GP or as strings of integers in GE, but both share the traits discussed below.

4.1.1. Function and terminal sets

Trading rules will be encoded as trees in GP and as vectors of integers in GE. In our work, both the terminal elements and the non-terminal functions mirror the ones used by Lohpetch and Corne [20]. One difference, however, is that we work with daily returns rather than monthly ones. Terminal elements include technical indicators and past sessions prices, while non-terminal ones consist of logical and relational operators (*and*, *or*, *not*, > and <).

The technical indicators are:

- Opening, closing, high and low daily prices for the index (*Opening*, *Closing*, *Max* and *Minimum*).
- 2, 3, 5 and 10-month moving averages: moving averages are the simple averages of closing prices over a predefined number of time periods. They are widely used indicators in technical analysis to detect trends and smooth the price series (*M2*, *M3*, *M5* and *M10*).
- Rate of Change Indicator (ROC) (3-month and 12-month): the price rate of change is a technical indicator of momentum that measures the percentage change in price between the current price and the price *n* periods in the past (*Roc3* and *Roc12*).
- Price Resistance Indicators: price points in the market which are expected to be difficult to break. We consider the two previous 3-Month moving average minima, lower resistance indicators, and the two previous 3-month moving average maxima, upper resistance indicators (*Mx1*, *Mx2*, *Min1* and *Min2*).
- Trend Line Indicators: a lower resistance line based on the slope of the two previous minima plus an upper resistance line based on the slope of the two previous maxima. Trendlines are used to show direction and speed of price changes (*UR* and *LR*).

4.1.2. Grammar

As we intend to use strongly-typed GP, the rules designed to combine the elements of the function sets both among them and with the terminal sets will be applicable to GP and GE alike.

The grammar built to describe the investment rules is based on a single IF-THEN-ELSE expression of traditional languages. If the expression is evaluated as true, it will be interpreted as a signal to buy, otherwise it returns false which represents an instructions to sell. The grammar thus generated follows the very simple approximation of Lohpetch [20] which can be expressed

in the BNF form as:

$$
\begin{aligned}
&< Rule >::=< bool > \\
&< Bool >::= (\textbf{\textit{And}} < bool >< bool >)|(\textbf{\textit{Or}} < bool >< bool >) \\
&< Bool >::= (\textbf{>}< exp >< exp >)|(\textbf{<}< exp >< exp >) \\
&< Exp >::= (\textbf{\textit{Opening}})|(\textbf{\textit{Closing}})|(\textbf{\textit{Max}})|(\textbf{\textit{Minimum}}) \\
&< Exp >::= (\textbf{\textit{M}2})|(\textbf{\textit{M}3})|(\textbf{\textit{M}5})|(\textbf{\textit{M}10}) \\
&< Exp >::= (\textbf{\textit{Roc}3})|(\textbf{\textit{Roc}12}) \\
&< Exp >::= (\textbf{\textit{Mx}1})|(\textbf{\textit{Mx}2})|(\textbf{\textit{Min}1})|(\textbf{\textit{Min}2}) \\
&< Exp >::= (\textbf{\textit{UR}})|(\textbf{\textit{LR}})
\end{aligned}
\tag{1}
$$

Some configurations require using automatically defined functions (ADFs). In those circumstances we extend the previous grammar (1) with the definition:

$$< bool > ::= (ADF1 < bool >< bool >)$$

and we add the component required to express the ADF itself.

$$
\begin{aligned}
&< Rule >::=< bool > \\
&< Bool >::= (\textbf{\textit{And}} < bool >< bool >)|(\textbf{\textit{Or}} < bool >< bool >) \\
&< Bool >::= (\textbf{>}< exp >< exp >)|(\textbf{<}< exp >< exp >) \\
&< Exp >::= (\textbf{\textit{ARG}0})|(\textbf{\textit{ARG}1})
\end{aligned}
$$

4.2. Fitness function

We define the fitness of strategies simply as the sum of all adjusted transaction benefits $\Delta r = r$ as it is done in [13]. Following authors like [15], we will rely on a continuous compound yield. The main advantage of using this approach in this context vs. an alternative based on raw returns is the fact that it is easy to scale forward. This simplification reduces the complexity of computing market returns over n periods from $O(n)$ multiplication operations to $O(1)$ additions. This lowers the computational cost very significantly, while the approximation is still very good.

The continuous compound yield of an investment rule is given by the expression:

$$r = \sum_{t=1}^{T} r_t \cdot I_b(t) + \sum_{t=1}^{T} r_f(t) \cdot I_s(t) + n \cdot ln\Big(\frac{1-c}{1+c}\Big) \tag{2}$$

where:

- $r_t = ln(P_t) - ln(P_{t-1})$ is the continuous composite yield. P_t is the price at time t. r_t calculates the return on investment over time when, according to the rule obtained, the investor should be in the market.
- $I_b(t)$ indicates a buy signal. It adopts the value "1" if the rule suggests buying at time t and "0" otherwise.
- r_f calculates risk-free return on investment when, following the rule obtained, the investor is out of the market.

- $I_s(t)$ is the sell signal. It is the opposite value of $I_b(t)$. It adopts the value "1" if the rule recommends selling at time t and "0" otherwise. It is the opposite of $I_b(t)$.
- The third component in the equation models transaction costs (assumed to be 0.25%). There, n denotes the number of transactions expressed as a purchase signal followed by a sales signal (any open position is closed on the last day), and c is the cost of a transaction expressed as a fraction of the price.

r_f can be calculated in different ways. Jansen[23] computes it as:

$$r_{f_{(t)}} = ln\frac{(1 + r_f, monthly)}{\delta} \tag{3}$$

Where $(1 + r_f, monthly)$ refers to the monthly interest in the money market and δ to the number of days in which the market is open. In this work we follow the same approach but we consider, for simplicity, calendar days instead. It is unlikely to have any relevant impact, specially in a low interest rate environment. As Jansen [23] mentions "the calculation of performance is not very precise and in any case the yields obtained by the GP in this concept are marginal. Even some authors discard them".

Other authors like Allen and Karjalainen [12] or Lohpetch and Corne [20], evaluate solutions in terms of excess investment return over B&H strategy $\Delta r = r - RBH$. Here, RBH is the investment return obtained buying at the beginning of the period and selling at the end, which is formally defined as:

$$RBH = \sum r_t + ln\Big(\frac{1 - c}{1 + c}\Big) \tag{4}$$

where, as it was discussed in regards to Equation (2), r_t is the market return during period t, and c represents the one-way transaction cost.

Our approach is almost the same, as the only difference is the subtraction of a constant, but the suggested representation has practical advantages as it has a slightly lower computational cost, and lower probability of dealing with negative fitnesses.

4.3. Parametrization

There are many parametrization possibilities for GP and GE algorithms. In our study, we have adopted, in general, common values found in the literature, and in other cases the choice was made based on exploratory tests.

The baseline experiments for both GP and GE-based configurations use populations of 500 individuals that are evolved over 50 generations. We implement elitism, carrying over the best strategy from every generation to the next.

Another element in common, that should be taken into account, is a mechanism designed to enhance variability. The initialization of the population or its mutation might lead to the same individual appearing more than once. In order to keep the population as rich as possible, should

this happen, we replace the repeated individual with a new one. The number of attempts to generate a replacement trading rule is limited to 100.

4.3.1. GP parametrization

The initialization of the population in GP is performed using the uniform algorithm. We define an initial target value of complexity of a minimum of 5 nodes and maximum of 25.

The method of selection chosen is the simple tournament with two candidates. The selection probability of terminal nodes is 20% and that of non-terminals 80%.

We used a multi-breeding pipeline with the following 3 sources and probabilities:

- Crossover. With a 0.8 probability, two individuals are selected by tournament and are crossed generating two other individuals that are introduced into the new population. In the crossing operator, a desired tree complexity limit of 50 nodes and a maximum depth of 7 levels is established. The crossover performs a standard subtree crossover: from each of the selected individuals a node is selected from their trees and the two subtrees rooted by those nodes are swapped.
- Reproduction. With a 0.1 probability, a single individual selected by tournament is copied directly into the new population.
- Mutation. With a 0.1 probability, a single individual obtained by tournament is mutated and introduced into the population. Mutation performs a standard subtree mutation: a node of the selected individual tree is selected and the subtree rooted by that node is replaced in its entirety by a randomly-generated tree. The minimum number of desired nodes, in the resulting individual, is set to 10 and the maximum to 50.

The new individuals are added to the population using a (μ, λ) replacement strategy [44], keeping population size constant across generations.

4.3.2. GE parametrization

GE basically relies on the same parametrization discussed for GP. The differences are the following:

- The initialization of individuals is defined by geometric series with a minimum initial complexity value of 5 and a growth probability of 0.85.

In GE the multi-breeding pipeline has also 3 sources with the following probabilities associated to them:

- Crossover. With a probability of 0.85, two individuals are selected by tournament and are crossed by a one-point crossover, generating two other individuals that are introduced into the population. The vector crossover procedure picks two random indexes i and j in each of two individuals A and B and then swaps the string of genes from the index start point to the end of the vector, A_i, ..., A_{end} and B_j, ..., B_{end}. Finally, to the obtained individuals a truncation mechanism is applied to determine how many integers of their vector genotype are consumed, from the grammatical definition, in the generation of the tree, eliminating the rest of integers that have not been used.

- Duplication. With a probability of 0.05, a duplication mechanism is applied to a single individual selected by tournament. In the duplication, a sequence of integers of the individual's own vector, defined by two random indexes, is selected and appended to the end of the list. The truncation technique is then applied.
- Mutation. With a probability of 0.1, an individual selected by tournament is mutated with a probability of 0.05 by randomly modifying one of its genes between the defined maximum and minimum values (-128, 127). We used uniform mutation, allowing a circular doubling (wrapping) of genes vector, up to 16 times, in the grammar translation process of the mutated individual. The 0.95 rest of the time, individuals are copied without modification. Finally, the mutated individuals, previously truncated, are introduced into the population.

4.4. Data sets and experimental protocol

In this subsection we introduce the sample to be used together with the experimental design, and the statistical significance testing protocol.

4.4.1. Sample

The experimental study relies on the combination of two datasets covering 12-year worth of data, from 2004 to 2015: one corresponds to Standard & Poor's 500 index and the second to the data necessary to calculate the daily risk-free return. The former was obtained from the commercial provider Datastream, and the latter from the Federal Reserve Bank of Atlanta, available at URL https://fred.stlouisfed.org/series/TB3MS.

4.4.2. Experimental design

Financial series often show structural changes that affect their predictability. In order to make the benchmarking exercise more generic, we test the configurations based on GP and GE under three different scenarios. To this end, the mentioned 12-year datasets have been split into three consecutive 4-year ones. For each of them, we created two samples: a training set, that covers the first three years, and a test sample with the fourth one. This results in the following subsets:

- 2004-2006 training 2007 test.
- 2008-2010 training 2011 test.
- 2012-2014 training 2015 test.

For each of them, we will test the strategies resulting from the combination of GP and GE with the potential use of ADFs and Lexicographic Parametric Parsimony growth control. That is, for each sample, we will benchmark:

- (Gp). Basic Genetic Programming.
- (GpPa). Genetic Programming with Lexicographic Parametric Parsimony.
- (GpAdf). Genetic Programming using ADFs.
- (GpAdfPA).Genetic Programming using ADFs and Lexicographic Parametric Parsimony.
- (Ge). Basic Grammatical Evolution.

- (GePa). Grammatical Evolution with Lexicographic Parametric Parsimony.
- (GeAdf). Grammatical Evolution using ADFs.
- (GeAdfPa). Grammatical Evolution with ADFs and Lexicographic Parametric Parsimony.

The basic experimental study is supplemented with a sensitivity analysis, to consider the impact of larger populations and additional technical indicators. In this regard, we have tested the following combinations:

- 500 individuals.
- 3000 individuals.
- 500 individuals plus extended technical indicators.
- 3000 individuals plus extended technical indicators.

As a result of applying the Cartesian product to the mentioned possibilities, we obtain a total of 96 different configurations to be tested. Given the stochastic nature of the algorithms, in an effort to reach a high level of significance, all the experiments are repeated 2,000 times. This adds up to a total of 192,000 experiments carried out.

4.4.3. Statistical significance evaluation protocol

The results obtained have been evaluated following the protocol used by García et al. [54] which is set out below:

The contrast of the normality of the population investment returns (continuous random variables) is performed using the Kolmogorov-Smirnov test, applying the Lilliefors correction. If the normality of the observations is rejected, then the non-parametric test of Wilcoxon's sign ranges is applied. Otherwise, the homoscedasticity of the variances is checked by the Levene test. If there is homoskedasticity, the t-test is applied, and if there is heteroskedasticity, the Welch test is used.

Algorithm 1: Statical significance testing protocol

```
if returns follow Normal distribution (Kolmogorov-Smirnov test with the Lilliefors correction) then
    if variances are homogeneous (Levene test) then
        A t-test is performed.
    else
        A Welch test is executed.
    end if
else
    A Wilcoxon test is applied to compare medians
end if
```

In order to do this experimental work, we have used the Java-based Evolutionary Computation (ECJ) framework, together with control and visualization tools developed ad-hoc and R scripts for the statistical analysis. The hardware used in the simulations was composed of two four-core I7 computers with 16 gigabytes of memory that have produced about 200,000 output log files.

5. Experimental Results

5.1. Baseline

The first set of experiments of this benchmarking exercise will set the baseline for the two core algorithms and their variations. We will compare the performance of both GP and EG vs. B&H strategy using populations of 500 individuals and the function set introduced in the previous section. The reported compounded returns have been computed according to Equations 2 and 4 discussed in 4.2. In addition to returns, we will consider other aspects such as complexity or reliability.

Table 3: Results Summary. Return on test samples over 2000 runs.

	2007 (Train 2004-06)		**2011 (Train 2008-10)**		**2015 (Train 2012-14)**	
	Mean	**Variance**	**Mean**	**Variance**	**Mean**	**Variance**
Ge	0.0424	0.00057	0.0079	0.0018	-0.0155	0.00047
GePa	0.0432	0.00049	0.0124	0.0014	-0.0160	0.00055
GeAdf	0.0419	0.00039	-0.0081	0.0035	-0.0153	0.00041
GeAdfPa	0.0413	0.00028	-0.0040	0.0032	-0.0168	0.00058
Gp	0.0359	0.00077	-0.0633	0.0054	-0.0149	0.00028
GpPa	0.0366	0.00065	-0.0485	0.0051	-0.0148	0.00035
GpAdf	0.0367	0.00082	-0.0481	0.0051	-0.0154	0.00034
GpAdfPa	0.0363	0.00072	-0.0470	0.0048	-0.0153	0.00038
B&H	0.0310		-0.0171		-0.0101	

Table 3 summarizes the main experimental results regarding financial performance on the test sample. For each configuration, we report the mean return and variance over 2000 runs. As we can see, GE beats GP in the first two time periods. This comes together with more consistent results, as evidenced by variances. However, this dominance is reversed in both fronts in the final period. The addition of the parsimony control mechanism generally results in higher returns, but the advantages of adding ADFs are less consistent.

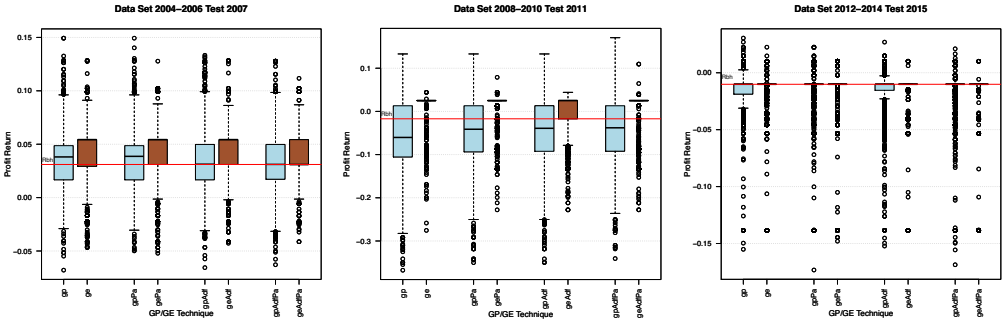
Table 4: Statistical significance of differences on test samples.

	Significance 2007 / 2011 / 2015							
	Ge	**GePa**	**GeAdf**	**GeAdfPa**	**Gp**	**GpPa**	**GpAdf**	**GpAdfPa**
GePa	-/--/=							
GeAdf	=/=/-	=/=/=						
GeAdfPa	++/=/=	++/=/=	=/--/=					
Gp	++/++/=	++/++/=	++/++/=	=/++/=				
GpPa	++/++/=	++/++/=	++/++/=	=/++/=	-/--/			
GpAdf	++/++/=	++/++/=	++/++/=	=/++/=	=/--/	=/=/		
GpAdfPa	++/++/=	++/++/=	++/++/=	=/++/=	=/--/	=/=/	=/=/	
B&H	++/++/--	++/++/--	++/++/--	+/++/--	++/--/--	++/--/--	=/--/--	=/--/--

Related to the above, Table 4 shows the statistical significance of the differences reported in Table 3 for the three consecutive periods. When the algorithm in the column offers a mean/median return that is higher than the one in the row at a significance level of 1%, we use the symbol

++. A similar difference in the opposite direction is represented with --. In case the difference is significant at the 5% conventional level, we use - or +, as appropriate. Finally, if we can not rule out the possibility of equality, we show =.

The best strategies by setup and period for the baseline configuration are reported in Appendix A. For those configurations that use ADFs, both the main rule, Tree 0, and the ADF, ADF1, are provided. In some instances, even though ADFs were evolved, the rule did not make use of them, hence the lack of reference to ADF1. The behavior of these rules is illustrated in Appendix B. The lighter line in the figures represent the recommendation. It either tracks the market, in case the recommendation is staying in the market, or remains at the bottom in case the investor should be out of it.



(a) 2007 (Train 2004-06) (b) 2011 (Train 2008-10) (c) 2015 (Train 2012-14)

Figure 3: Return distribution by configuration. Test samples over 2000 runs.

The distribution of returns by technique and period is reported in Figure 3. There, we can see three panels showing box-plots for the different configurations. Each panel illustrates the distribution of returns on test samples (years 2007, 2011 and 2015) over 2000 experiments. As a reference point, we have added horizontal lines that indicate the return that would have been obtained following a B&H strategy. It is worth noting that the distributions are rarely symmetrical. They tend to be skewed, showing longer left tails that represent poor strategies that are very unprofitable. This is specially apparent in the last period which, as we see in Table 3, is particularly challenging.

Table 5: Results vs. Buy-and-Hold on test samples with 500 individuals and core terminal set over 2000 runs.

	2007 (Train 2004-06)				**2011 (Train 2008-10)**				**2015 (Train 2012-14)**			
	Prediction		**Complexity**		**Prediction**		**Complexity**		**Prediction**		**Complexity**	
	Better	**Equal**	**Depth**	**Nodes**	**Better**	**Equal**	**Depth**	**Nodes**	**Better**	**Equal**	**Depth**	**Nodes**
Ge	63.70%	10.95%	3.24	7.56	84.80%	2.20%	2.89	5.81	7.95%	70.45%	3.66	8.68
GePa	65.45%	14.10%	2.97	6.38	88.85%	1.65%	2.79	5.31	4.55%	79.45%	3.19	6.88
GeAdf	55.05%	31.25%	2.96	6.09	74.70%	0.85%	2.74	5.11	0.50%	87.65%	2.89	5.48
GeAdfPa	49.75%	40.35%	2.73	5.06	77.50%	1.05%	2.67	4.86	0.30%	87.30%	2.87	5.33
Gp	53.60%	0.35%	6.68	32.02	28.90%	3.45%	6.85	36.29	12.45%	50.25%	6.17	24.29
GpPa	52.80%	1.00%	6.02	24.30	37.45%	7.35%	6.20	22.08	6.95%	71.30%	5.71	16.87
GpAdf	50.15%	2.45%	8.38	34.80	37.20%	6.20%	8.63	39.33	10.00%	59.50%	7.77	28.18
GpAdfPa	50.05%	2.55%	6.89	25.14	36.20%	8.50%	8.14	35.16	6.65%	70.55%	7.13	21.73

Table 5 provides detailed information regarding the performance of the various configurations

vs. B&H. For each of them, we report the percentage of trading strategies that beat B&H on test, together with the percentage of experiments that result in strategies that provide the same return. The latter figure, reported in column "Equal", is usually the result of strategies that end up mirroring the naive strategy on the test sample. This information is often overlooked in the literature, but it is very relevant because it provides insights on the reliability of the different approaches as trading strategy generators.

The results reported in table 5 show that in 2007 GE obtains rules that either match of beat B&H between 74.65% and 90.10% of cases, while GP does it slightly below 54% of times. GE obtains better results in the period 2008-2011, where it beats or improves B&H between 78.55% and 90.50% of cases, while GP makes it significantly below 50%, between 32.35% and 44.70%. It should be noted that in both the 2004-2007 and the 2008-2011 periods, the standard version of GE and the version with parsimony control show a specially good relative performance.

In 2012-2015, the situation is significantly different. In that period, the strategies produced by the two core algorithms generate very few purchase or sale signals, and end up effectively following a B&H strategy. When that is not the case, both GE and GP are outperformed by B&H far more often than the other way around. This means that, even though the average results seem very poor, this is mostly caused by the long left tail of the distribution of returns already mentioned when we described Figure 3. In practice, for all the configurations but two, the probability of matching B&H was between 71% and 88%.

The second set of columns reports the average complexity of the solutions obtained by the different approaches. The first column indicates the average depth of the trees generated, and the second one the total number of terminal and non-terminal nodes. It is clear that the complexity of the rules is significantly higher when we use GP versus GE as the core algorithm. For example, in the period 2008-2011 the average number of nodes for GP is 33.22 compared to 5.27 of GE. The average depth of solutions shows a similar pattern as it reaches 7.46 for GP vs 2.77 for GE. As for the use of the parsimony operator, it reduces the complexity of the GP-based solutions by 9.19 nodes and their depth by 0.57. If we consider GE, the effects of the operator have the same direction, but they tend to be more limited because it is already far less complex than GP. It reduced the average number of nodes by 0.38, and the mean depth by 0.09.

Regarding the computational cost of this experimental work, the computational complexity of GE and GP depends on several factors:

- The genetic operators applied (crossover, selection and mutation), which in turn depend on their implementation.
- The representation of the individuals and the population (size).
- Number of generations.
- Number of experiments performed.
- Fitness function.

In our experiments, the fitness function overwhelms the computational cost of genetic operators and representation. As a result, they may be ignored and we can characterize the complexity as

of linear order:

$$O(jgn) \tag{5}$$

Where g represents the number of generations, n the population size and j the number of experiments carried out.

Figure 4 summarizes the total accumulated cost (in seconds) over the three samples, by configuration. It is apparent that the standard version of GP required more computation time than GE (29% more). Lexicographic Parametric Parsimony reduced the computational cost of both techniques, but GP profited the most (24% vs 8%). The difference between the two core algorithms increases very substantially once ADFs are considered. The versions based on GP that included this feature turned out to be significantly more expensive than the ones based on GE.

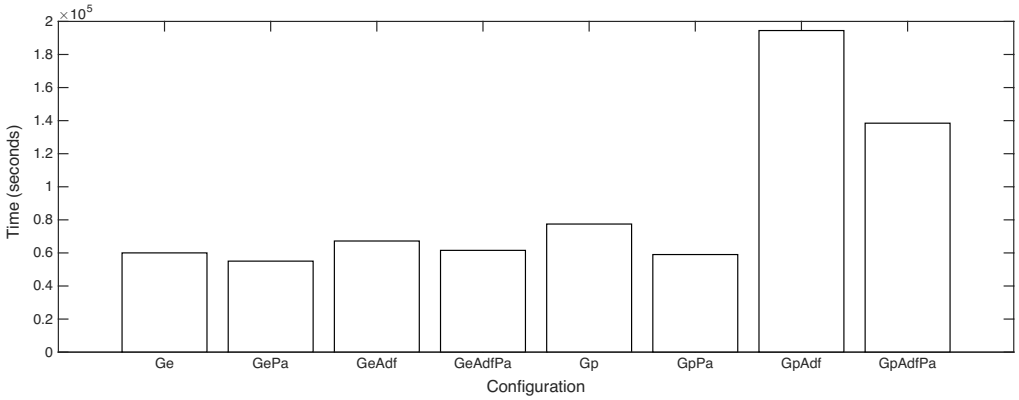


Figure 4: Computational cost. Total accumulated cost (seconds) over the three samples by configuration. Experiments ran on an Intel I7-6500U 2.5 Ghz with 16 Gbytes of DDR3 RAM.

As a summary of this part of the study, we can conclude that, according to our experiments, GE, particularly both in the standard version and using parsimony control, is more robust and generates simpler solutions than GP. At the same time, it beats the B&H strategy significantly more often than GP.

5.2. Sensitivity analysis

In order to enrich the analysis, we test the sensitivity of results to two extensions, population size and terminal set, both separately and combined. This defines three additional sets of experiments whose main results are summarized in Table 6.

5.2.1. Extended set of technical indicators

In this section we test the sensitivity of results to the number of technical indicators. The aim is exploring whether the algorithms can exploit this new information to find better trading rules. In addition to this, we intend to see whether there are patterns regarding the capabilities of the different configurations to extract additional profitability. To this end, we extend the richness of

Table 6: Sensitivity analysis. Average returns on test samples over 2000 runs.

	Extended Function Set			Extended Population			Extended Pop. & Func. Set		
	2007	**2011**	**2015**	**2007**	**2011**	**2015**	**2007**	**2011**	**2015**
Ge	0.04720**	-0.05556	-0.02056	0.03757	-0.00790**	-0.01490	0.05495**	-0.01171**	-0.01605
GePa	0.04596**	-0.05745	-0.01986	0.03706	0.00419**	-0.01567	0.05375**	-0.00344**	-0.01700
GeAdf	0.04651**	-0.07846	-0.02040	0.04131**	0.00409**	-0.01650	0.04967**	-0.00611**	-0.01927
GeAdfPa	0.04613**	-0.08342	-0.01917	0.04288**	0.01231**	-0.01667	0.04879**	-0.00293**	-0.01935
Gp	0.05671**	-0.06797	-0.01503	0.03779**	-0.07297	-0.01906	0.05992**	-0.06867	-0.01799
GpPa	0.05391**	-0.06631	-0.01501	0.03428**	-0.04932	-0.01654	0.05704**	-0.05537	-0.01812
GpAdf	0.05576**	-0.06192	-0.01559	0.04242**	-0.05781	-0.01754	0.06024**	-0.05685	-0.01490
GpAdfPa	0.05535**	-0.05679	-0.01500	0.04100**	-0.05727	-0.01659	0.06019**	-0.05784	-0.01438

** Significant vs. B&H at 1% *Significant vs. B&H at 5%

the function set suggested by Lohpetch and Corne, keeping the population size at 500 and the rest of execution parameters at the same values that we used in the baseline.

The extended set will include, in addition to the indicators already used in the baseline, new moving averages and pivot points. Pivot points [55, 56, 57] are used to determine the supports, resistances and directional changes in stock prices and represent the market position in relation to previous sessions. In addition to the standard pivot points that will be used in the analysis, there are other types like Fibonacci pivot points, Demark pivot points, Woodie pivot points or Camarilla points. Specifically, the new indicators were:

- Resistance indicators R1, R2, R3 and support S1, S2, S3 calculated as follows:

  The pivot value is calculated as P = (O + H + L + C) / 4. Where P is the pivot, O is the opening price, H is the maximum value, L is the minimum value and C is the closing value, all of them referred to the previous session. The first support value (S1) and resistance (R1) are obtained from the observation of the price range above the pivot point (H - P), and below it.

  - $R1 = P + (P - L) = 2xP - L$
  - $S1 = P - (H - P) = 2xP - H$

  The second resistance values (R2) and support (S2) are above and below the previous support and resistance values (R1 and S1).

  - $R2 = P + (H - L)$
  - $S2 = P - (H - L)$

  And likewise with the third resistance and support values that are expressed as a function of R1 and S1.

  - $R3 = H + 2x(P - L) = R1 + (H - L)$
  - $S3 = L - 2x(H - P) = S1 - (H - L)$

- M4, M6, M7, M8, M9, M11, M12. Moving averages of closing prices over a period of four, six, seven, eight, nine, eleven and twelve months prior to the current month, corresponding to those of 100, 150, 175, 200, 225, 275 and 300 sessions respectively.

If we compare the figures reported in Tables 3 and 6, we observe a high dispersion of direction changes in results depending on the period and basic algorithm considered. For the 2004-2007 dataset (train 2004-2006, test 2007), adding the new terminals results in a widespread improvement in returns. The scale of the change is specially remarkable for the configurations based on GP, which outperform the GE alternatives in every case. We observed that the proportion of strategies that beat B&H in 2007 grows across all configurations. That is also the case for GE configurations when we consider the number of strategies that match B&H. In the case of GP with parsimony control and ADFs, the proportion of strategies that beat B&H reaches 73.95%, up from the 50.05% baseline result.

For the rest of the periods the results of the terminal set extension are not as good. In 2011 neither GP-based nor GE-based configurations are longer able to beat B&H in terms of average return. There seems to be a general decrease in performance that affects GE to a greater extent. The same applies to 2015. Interestingly, for this years we observe a sharp decrease in the number of strategies that beat B&H that is specially acute in 2011 for GE (in the order of 30% to 40%, depending on the configuration), while the proportion of strategies that follow the benchmark goes up. A potential explanation for this behavior would be the fact that increasing the number of indicators expands the solution space. Given that we kept constant the population size and the number of generations, the algorithms might not have been able to converge to the best solutions. That might explain why they were not able to exploit conveniently the new information.

As in the previous test, the complexity and depth of the individuals obtained by GP is significantly greater than that reached by the solutions of GE. The average complexity of the solutions generated by GP and GE is similar to that calculated for the baseline configurations, in both number of nodes and depth.

5.2.2. Extended population

In this test, the modification introduced was increasing the number of individuals from 500 to 3000. The results, as expected, show clear signs of over-fitting. All of the GP and GE-based configurations obtain higher returns in training. However, once we consider the test sample, the performance is not as good.

The prediction of the algorithms in the test phase undergo variations: for example, in the 2007 test sample has worsened for the GE and slightly improved for the GP configurations that make use of ADFs (GpAdf and GpAdfPa). In 2011 most of the configurations got worse results, only the approaches based on GE with ADFs, GeAdf and GeAdfPa, improved. Having said that, the variations in absolute terms are very small. Finally in 2015, GE remains almost the same, but GP is affected negatively. As for the performance vs. B&H, the evidences is, once again, mixed in the sense that GE and GP also beat it in the period 2004-2007; only the variations related to GE do it in the period 2008-2011, and both GP and GE fail to do it in the period 2012-2015.

The complexity and depth of the individuals obtained by GP are significantly larger than that reached by the GE: 7.63 versus 3.61 in depth and 34.94 versus 8.55 in complexity. Comparing the results with the Baseline, complexity of the solution achieved in both GP and GE increases with the number of individuals, and we obtain the following results: in GP the number of nodes increases between 6.05 and 6.20, and from 1.89 to 3.11 in GE. If we consider depth, the average value for GP grows between 0.56 for the versions without parsimony control vs. 0.61 for those that implement it. The impact in GE is similar, as these increases are 0.79 and 0.49 respectively.

In summary, GE still provides the highest reliability in terms of probability of beating B&H, as it does it on 2 out of the 3 samples, and it also generates simpler solutions. The use of ADFs and the parsimony control mechanism improves the results, and the mere increase of population size does not generally results in better performance.

5.2.3. Extended population and technical indicator set

In this case, we increase both the population and the number of technical indicators. Other than that, we maintain the rest of the execution parameters.

The combination of these changes does not have a major impact in relative terms. In general, the average return of the trading strategies suffers a slight degradation for most configurations. Having said that, it is worth noting that GP based strategies profited specially from this changes in the 2007 test sample.

When we consider the proportion of rules found that either beat B&H or replicate it vs. the benchmark configurations, once again show the importance of the distribution results. The proportion of strategies that replicate or beat B&H goes up from 67.94% to 83.09% in 2007; down from 62.11% to 57.53% 2011 and up again 78.23% to 85.98% in 2015. The robustness of the strategies improves in 2 out of 3 samples, but mean results are dragged by a few weak strategies that offered very poor performance.

As in the previous three experimental setups, the average depth and the number of nodes of the individuals obtained by GP-based configurations is clearly higher than that obtained by the set of solutions that use GE as the core algorithm: 7.74, compared to 3.69 in depth and 8.88 compared to 35.66 in complexity. Finally, the complexity of the solutions achieved by both GP and GE are very similar to the one observed in the test with 3000 individuals.

Table 7: Sensitivity analysis summary. Proportion of trading rules that provide returns that are equal or better than Buy & Hold together with their complexity. Averages over 2000 runs and the three test samples.

	Baseline			Extended Function Set			Extended Population			Extended Pop. & Func. Set		
	Ret. >=	Complexity		Ret >=	Complexity		Ret. >=	Complexity		Ret. >=	Complexity	
	B&H	Depth	Nodes	B&H	Depth	Nodes	B&H	Depth	Nodes	B&H	Depth	Nodes
Ge	80.02%	3.26	7.35	75.32%	3.25	7.19	66.18%	4.14	10.91	83.42%	4.01	10.37
GePa	84.68%	2.98	6.19	76.17%	3.03	6.27	71.25%	3.59	8.55	85.70%	3.65	8.74
GeAdf	83.33%	2.86	5.56	71.53%	3.03	6.18	76.18%	3.57	8.22	84.47%	3.88	8.77
GeAdfPa	85.42%	2.76	5.08	71.18%	2.82	5.33	81.15%	3.12	6.50	86.57%	3.22	7.64
Gp	49.66%	6.57	30.87	57.13%	6.58	31.42	39.62%	6.84	36.77	64.05%	7.40	37.74
GpPa	58.95%	5.98	21.08	62.92%	6.33	26.92	48.18%	6.29	26.31	62.87%	6.50	30.06
GpAdf	55.17%	8.26	34.10	67.15%	8.08	33.97	47.63%	9.10	42.18	68.77%	8.71	39.11
GpAdfPa	58.17%	7.39	27.34	66.55%	7.43	28.08	50.80%	8.28	34.50	68.43%	8.34	35.71

Table 7 summarizes the main results of the benchmarking exercise. There, we report, for all configurations and experimental setups, a summary of the metrics considered over 2000 runs and the three test samples. Specifically, we report the proportion of trading rules that provide returns that are equal or better than B&H together with their complexity in terms of depth and number of nodes.

If we consider all experiments and configurations, the two core algorithms offered very similar average return. Having said that, GE with 0.34% beats GP by 0.9% in absolute terms. Grammat-

ical evolution also turned out to be more reliable. Configurations based on GE obtained equal or better results than B&H in 78.91% of the experiments. Genetic programming, on the other hand, matched or improved the performance of B&H in 57.88% of the times. The structure of the trading rules also differed among techniques. GE provided simpler individuals both when we consider average depth, 3.32 vs 7.38 and number of nodes, 7.43 vs 32.26.

Regarding the benefits of implementing Lexicographic Parametric Parsimony or using ADFs, the evidence is mixed in terms of returns and clear if we consider robustness. The configurations that use the complexity control mechanism offer better average returns than those that do not. Even though that applies to both GE and GP, the latter profited more from it. The impact of ADFs was limited in absolute terms. However, even though added value to GP, it turned out to have a negative impact on GE, reducing average return from 0.48% to 0.20%. If we consider the probability of obtaining results that are at least as good as B&H, both mechanisms add value. Interestingly, the average contribution that was obtained in our experiments is similar, 3.08% and 3.52%. However, even though the distribution of the gains associated to mentioned parsimony control is similar for the two core evolutionary algorithms, ADFs contribute much more to PG (robustness goes up from 55.42% for configurations without ADFs to 60.33% with them).

The analysis of the experimental results, in relation to the differential impact that the increase in population size or the terminal set might have depending on the core algorithms, leads us to the following considerations. If we focus on robustness, the extension of population was positive for GE and negative for GP. The proportion of GP-based rules that offered returns equal or higher than B&H suffered and absolute drop of 3.17% vs an improvement of 0.91% for GE. This effect differs from the observed one when we consider average returns. While this indicator remains stable for GP, GE is affected more negatively, raising the importance of distributions and the probability of obtaining specially bad strategies.

The extension of the terminal set makes apparent the same phenomenon that we found for the population size. In one instance, there is a divergence between the sign of the change in average return and robustness, and the impact of the change on the two techniques is very different. If we compare the results of the base case together with extended population one vs the one with the extended function set and the one with both extensions, we see that the addition of pivot points and new moving averages helped both GE and GP to improve the average robustness of the trading strategies. The influence on GE is limited, but it was very positive for GP, as it went up from 51.02% to 64.7%. Regarding average returns, the improvement of 0.48% in absolute terms that we see in GP comes together with a 1.16% drop for GE.

6. Summary and Conclusions

The use of evolutionary computation in the domain of trading strategy generation and optimization has a long history. One of the most popular lines of research is the use of GP and GE to evolve investment rules based on technical indicators. Among the main advantages of this approach, we can highlight the possibility of obtaining rules that are flexible, as the structure is not completely predefined in advance, and interpretable.

The literature on this topic is based on studies with very different setups. Some authors rely on GP and others on GE; some consider extensions like the addition of parsimony control mechanisms or ADFs, and others do not. Since most of the studies are also based on different samples, the combination of all of these elements makes comparison efforts very challenging. It is hard to tell

which core algorithm offers better performance, whether adding extensions makes sense, or even if their impact is similar or asymmetric depending on the choice of GP vs GE. We intend to fill this gap in the literature presenting a comprehensive benchmarking exercise.

Our experimental work compares the profitability and the complexity of the trading rules for the S&P 500 generated using GP, more specifically STGP, and GE on three consecutive four-year periods. These samples differed in the degree of difficulty to find rules with the potential to beat a naive B&H strategy. For each of the algorithms, we tested four configurations: core algorithm; core extended with lexicographic parametric parsimony control; core extended with ADFs and, finally, core with both extensions. In addition to that, we tested the sensitivity of results to larger populations and a larger number of technical indicators. For the latter, we added pivot points and moving averages to the initial function and terminal set suggested by Lohpetch and Corne [11].

The performance of the algorithms depended on the time period, but they were consistent in the sense that the difficulties to find profitable trading rules affected both core algorithms the same way. If we consider the output of the 192,000 experiments, both algorithms offered very similar average returns. Having said that, GE performed better and also turned out to be more reliable in the sense that the proportion of strategies that obtained equal or larger returns than B&H was much higher. The structure of the trading rules also was different. GE generated simpler individuals both in terms of average depth and number of nodes.

Regarding the benefits of extending the core algorithms with lexicographic parametric parsimony control, the configurations that use this complexity control mechanism offered better average returns than those that do not use it. Despite of the fact that the assertion applies to both GE and GP, the latter profited more from it. The impact of ADFs was limited in absolute terms and, even though it improved the average return of the strategies based on GP, it turned out to have a negative impact on GE. If we focus the attention on reliability, both mechanisms add value, as they increase the probability of obtaining results that are at least as good as B&H. The average contribution of the extensions that we obtained in the experiments was the same, but the distribution differed. Parsimony control provides the same benefits to GP and GE, but the former absorbed most of the positive contribution of ADFs.

The experimental results of the sensitivity tests show that the extension of population increased slightly the robustness of GE-based rules, while it resulted in a mean drop in the the proportion of GP-based rules that offered a return equal or higher than B&H. Despite of this, average return remained stable for GP, while GE was affected slightly negatively. This raises the importance of distributions and the probability of obtaining specially bad strategies. Regarding the changes in the terminal set, the addition of pivot points and new moving averages improved the average return of GP at the cost of a decline in GE. However, this change helped both core algorithms in terms robustness of the trading strategies, though not in the same amount. GP showed much better capabilities to exploit the new information.

We feel that these results will help future researchers and practitioners to make a more informed decision regarding some of the key components to be used in the development of their trading systems based on GP and GE. We also hope that other authors will follow suit and replicate the study for other assets and periods to accumulate more valuable evidence under different market conditions. That, together with new analysis on the sensitivity to other aspects like specific operators, bounds on complexity or parameterization would result in significant progress towards the completion of this complex picture.

7. Acknowledgements

The authors acknowledge financial support granted by the Spanish Ministry of Science and Technology under grant ENE2014-56126-C2-2-R.

Appendix A. Best Strategies by Setup. Baseline Configuration

Train 2004-2006. Test 2007.

- *Ge*
  Fitness: 0.12847
  Tree 0: (Or (>Mx2 M2) (Or (>Minimum UR) (<M10 LR)))
- *GePa*
  Fitness: 0.12776
  Tree 0: (Or (>Mx2 M2) (<M10 LR))
- *GeAdf*
  Fitness: 0.12847
  Tree 0: (Not (ADF1 (>M2 Mx2) (Or (>Minimum UR) (>LR M10))))
  ADF1: (<ARG1 ARG0)
- *GeAdfPa*
  Fitness: 0.11167
  Tree 0: (ADF1 (>M2 Mx2)(Or (>Opening LR) (And (>M5 M2) (>M3 M2))))
  Tree 1: (<ARG1 ARG0)
- *Gp*
  Fitness: 0.14930
  Tree 0: (Or (<Max M10) (Or (>M10 M5) (<M2 M5)))
- *GpPa*
  Fitness: 0.14930
  Tree 0: (Or (Or (<M3 M10) (<Max M10)) (Not (>M2 M5)))
- *GpAdf*
  Fitness: 0.13329
  Tree 0: (ADF1 (ADF1 (And (<Opening Closing) (<LR LR)) (<Minimum Minimum)) (And (Or (<M3 Minimum) (<UR Closing))(>Closing M3)))
  ADF1: (And (Or (Not (<M5 M3)) (<M2 M5)) (<M2 Mx2))
- *GpAdfPa*
  Fitness: 0.12776
  Tree0: (ADF1 (<M2 Mx2) (>M10 M3))
  ADF1: (Or ARG0 (<M10 LR))

Train 2008-2010. Test 2011.

- *Ge*
  Fitness: 0.04410
  Tree 0: (And (<Min2 LR) (Not (And (>M10 LR) (Not (>Mx1 M3)))))

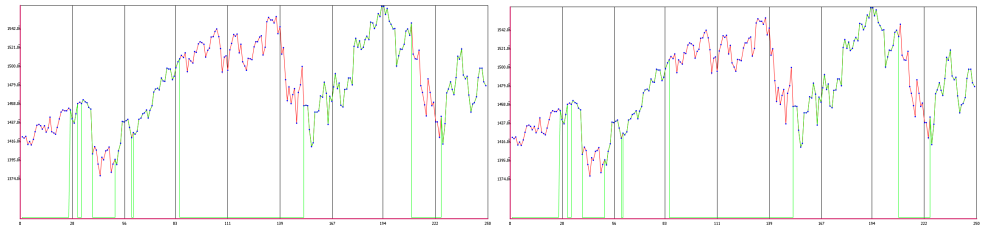
- *GePa*
  Fitness: 0.07888
  Tree 0: (And (Or (And (<M3 Minimum) (>M5 Closing)) (>LR Min2)) (>M2 Roc3))
- *GeAdf*
  Fitness: 0.04410
  Tree 0: (Or (<Mx1 M10) (Or (ADF1 (<M3 Mx1) (ADF1 (<LR Min1) (>Roc12 Min1))) (<M10 LR)))
  Tree 1: (<ARG1 ARG0)
- *GeAdfPa*
  Fitness: 0.10941
  Tree 0: (Or (Not (<LR Min2)) (And (<M2 M3) (ADF1 (>M5 UR) (<LR Minimum))))
  ADF1: (<ARG0 ARG1)
- *Gp*
  Fitness: 0.13300514044991257
  Tree 0: (Or (And (>LR Min2) (Or (>LR Min2) (And (<UR M10) (Or (>Max Min2) (>M5 UR)))))(Or (Or (And (<UR M10) (Not (>M5 UR)))(>LR Min2)) (Or (And (Not (>M5 UR)) (>Roc12 Roc3))(And (>LR Min2)(>M5 UR)))))
- *GpPa*
  Fitness: 0.13300
  Tree 0: (Or (<Min1 LR) (And (Not (<UR M5))(Or (Or (<M3 Closing) (>Min2 M5)) (Or (<Roc3 Roc12) (>M10 UR)))))
- *GpAdf*
  Fitness: 0.13300
  Tree 0: (ADF1 (And (Or (<UR M10) (<Roc3 Roc12)) (<M5 UR)) (ADF1 (<Roc3 Roc12) (ADF1 (ADF1 (<UR M10)(ADF1 (<UR M10) (>Minimum Mx1))) (>Minimum Opening))))
  ADF1: (Or (Not (<LR Min1)) (Or (<ARG0 ARG0) ARG0))
- *GpAdfPa*
  Fitness: 0.17065
  Tree 0: (Or (Or (ADF1 (<Max Min1) (<M10 Opening)) (<Min2 LR)) (Or (Not (<Min2 M5)) (<Min1 LR)))
  ADF1: (And (And (<LR Minimum) (>Mx1 UR)) (And (Or (<Closing M5) (And (>ARG1 M10) (And ARG0 ARG0))) (Or (<M5 UR) ARG0)))

Train 2012-2014. Test 2015.

- *Ge*
  Fitness: 0.02253
  Tree 0: (And (Or (>Max M3) (And (Or (>Mx1 Roc3) (<M5 Min2)) (<Minimum Closing))) (Or (>Max M3) (>Roc12 Roc3)))
- *GePa*
  Fitness: 0.01009
  Tree 0: (Or (>M5 Minimum) (<M5 M2))

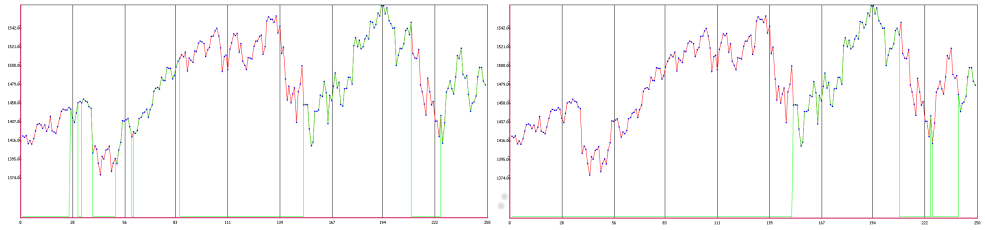
- *GeAdf*
  Fitness: 0.01009
  Tree 0: (Or (<M5 M2) (>M5 Minimum))
  ADF1: (>ARG1 ARG0)
- *GeAdfPa*
  Fitness: 0.01009
  Tree 0: (Or (<Minimum M5) (<M5 M2))
  ADF1: (Or (<ARG1 ARG1) (<ARG1 ARG0))
- *Gp*
  Fitness: 0.03046
  Tree 0: (Or (And (<Minimum M5) (>M5 M3)) (Or (Or (And (<Minimum Mx2) (Or (<UR M5) (>Min2 UR))) (Or (Not (>M5 M2)) (Or (<UR M5) (>Min2 UR)))) (<Max M5)))
- *GpPa*
  Fitness: 0.02252868979240462
  Tree 0: (Or (Not (>M3 Opening)) (And (<Minimum Closing) (Not (>Roc3 Roc12))))
- *GpAdf*
  Fitness: 0.026907
  Tree 0: (Or (And (>M5 Opening) (Not (<M5 M3))) (Or (And (>M2 Opening) (Or (And (>M5 Opening) (>Mx2 Roc12)) (<M5 M2))) (<M5 M2)))
  ADF1: (Or (>ARG1 Minimum) (<ARG0 Max))
- *GpAdfPa*
  Fitness: 0.02108
  Tree 0: (Not (And (Not (Or (>M5 UR) (Not (>Max M5)))) (<M2 M5)))
  ADF1: (And ARG1 (Or (<ARG0 Roc3) (And ARG1 ARG0)))

Appendix B. Best Strategy Behaviour by Setup. Baseline Configuration



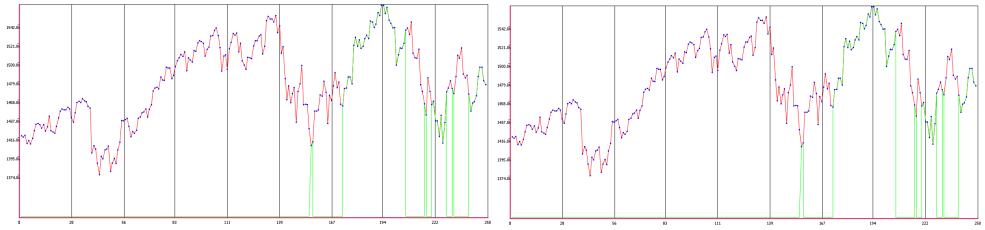
(a) Ge

(b) GePa



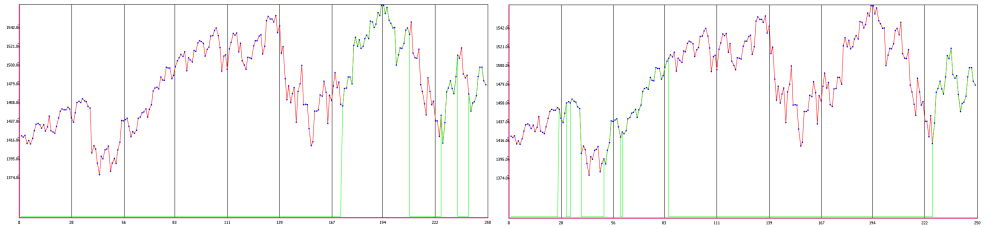
(c) GeAdf

(d) GeAdfPa



(e) Gp

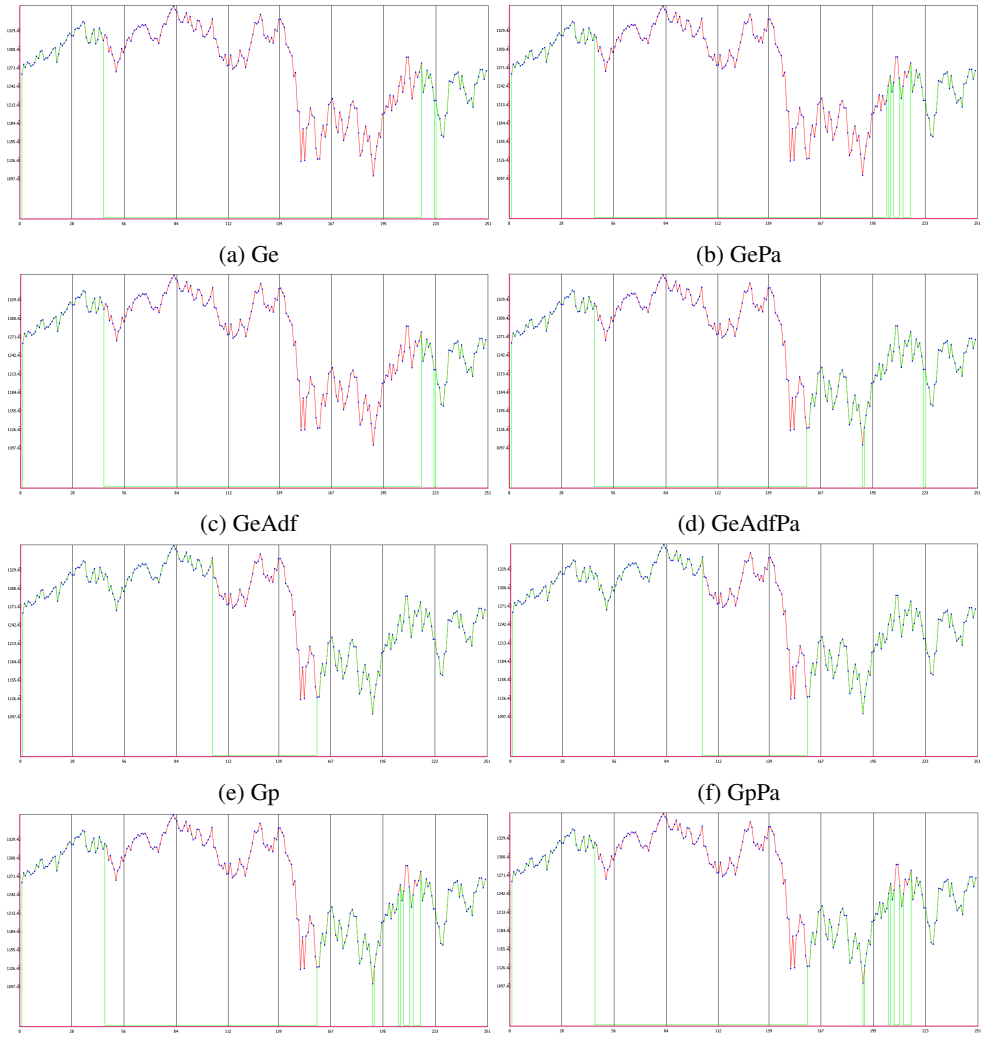
(f) GpPa



(g) GpAdf

(h) GpAdfPa

Figure B.5: Behaviour of best strategies by setup of baseline configuration on test sample. Train: 2004-2006, Test: 2007. Lighter color indicates rule recommendation.



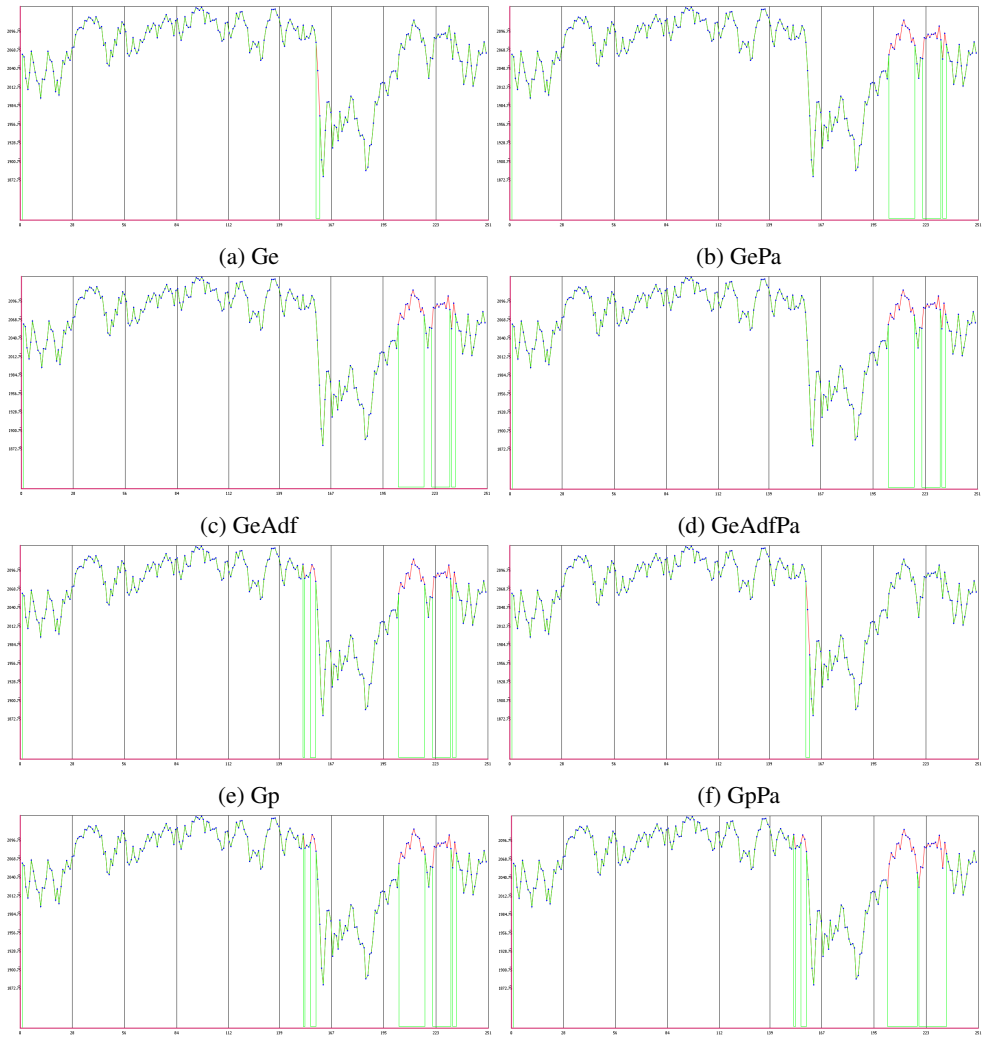
(a) Ge (b) GePa

(c) GeAdf (d) GeAdfPa

(e) Gp (f) GpPa

(g) GpAdf (h) GpAdfPa

Figure B.6: Behaviour of best strategies by setup of baseline configuration on test sample. Train: 2008-2010, Test: 2011. Lighter color indicates rule recommendation.



(a) Ge (b) GePa

(c) GeAdf (d) GeAdfPa

(e) Gp (f) GpPa

(g) GpAdf (h) GpAdfPa

Figure B.7: Behaviour of best strategies by setup of baseline configuration on test sample. Train: 2012-2014, Test: 2015. Lighter color indicates rule recommendation.

References

[1] A. P. Chaboud, B. Chiquoine, E. Hjalmarsson, C. Vega, Rise of the machines: algorithmic rrading in the foreign exchange market, The Journal of Finance 69 (5) (2014) 2045–2084.

[2] B. Biais, P. Woolley, High frequency trading, Manuscript, Toulouse University, IDEI.

[3] T. Foucault, J. Hombert, I. Rosu, News trading and speed, The Journal of Finance 71 (1) (2015) 335–382.

[4] G. Nuti, M. Mirghaemi, P. Treleaven, C. Yingsaeree, Algorithmic trading, Computer 44 (11) (2011) 61–69.

[5] B. Babcock, The Dow Jones-Irwin guide to trading systems, Dow Jones-Irwin, 1989.

[6] E. Fama, Efficient capital markets: A review of theory and empirical work, Journal of Finance 25 (1970) 383–417.

[7] A. W. Lo, H. Mamaysky, J. Wang, Foundations of technical analysis: computational algorithms, statistical inference, and empirical implementation, The Journal of Finance 55 (4) (2002) 1705–1765.

[8] A. Brabazon, M. O'Neill, Biologically inspired algorithms for financial modelling, Springer, 2006.

[9] J. R. Koza, P. J. Angeline, Genetic programming: On the programming of computers by means of natural selection, Vol. 33, MIT press, 1992.

[10] C. Ryan, J. Collins, M. O'Neil, Grammatical Evolution: Evolving Programs for an Arbitrary Language, Springer Berlin Heidelberg (1998) 83–96.

[11] D. Lohpetch, D. Corne, Outperforming buy-and-hold with evolved technical trading rules: Daily, weekly and monthly trading, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 6025 LNCS, 2010, pp. 171–181.

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

[13] C. Setzkorn, L. Dipietro, R. Purshouse, Evolving Rule-Based Trading Systems, 36th Annual Meeting of the CEA.

[14] J. D. Thomas, K. Sycara, Gp and the predictive power of internet message fraffic, in: Genetic Algorithms and Genetic Programming in Computational Finance, Springer US, Boston, MA, 2002, pp. 81–102.

[15] L. A. Becker, M. Seshadri, GP-evolved technical trading rules can outperform buy and hold, Proceedings of the Sixth International Conference on Computational Intelligence and Natural Computing, Embassy Suites Hotel and Conference Center, Cary, North Carolina USA, September 26-30.

[16] C. J. Neely, Risk-adjusted, ex ante, optimal technical trading rules in equity markets, International Review of Economics & Finance 12 (2003) 69–87.

[17] C. Fyfe, J. P. Marney, H. Tarbert, Risk adjusted returns from technical trading: a genetic programming approach, Applied Financial Economics 15 (15) (2005) 1073–1077.

[18] J. Y. Potvin, P. Soriano, V. Maxime, Generating trading rules on the stock markets with genetic programming, Computers and Operations Research 31 (7) (2004) 1033–1047.

[19] N. Navet, S. H. Chen, On predictability and profitability: Would GP induced trading rules be sensitive to the observed entropy of time series?, Studies in Computational Intelligence 100 (2008) 197–210.

[20] D. Lohpetch, D. Corne, Discovering effective technical trading rules with genetic programming: Towards robustly outperforming buy-and-hold, in: 2009 World Congress on Nature and Biologically Inspired Computing, NABIC 2009 - Proceedings, 2009, pp. 439–444.

[21] L. A. Becker, M. Seshadri, Comprehensibility and overfitting avoidance in genetic programming for technical trading rules, Tech. rep., Worcester Polytechnic Institute (2003).

[22] A. Esfahanipour, S. Mousavi, A genetic programming model to generate risk-adjusted technical trading rules in stock markets, Expert Systems with Applications 38 (7) (2011) 8438–8445.

[23] S. Jansen, Testing market imperfections via genetic programming, Ph.D. thesis, Univertitat Hohenheim (2011).

[24] J. How, M. Ling, P. Verhoeven, Does size matter? A genetic programming approach to technical trading, Quantitative Finance 10 (2) (2010) 131–140.

[25] W. Brock, J. Lakonishok, B. LeBaron, Simple technical trading rules and the stochastic properties of stock returns, The Journal of Finance 47 (5) (1992) 1731–1764.

[26] P. Gabrielsson, U. Johansson, R. Konig, Co-evolving online high-frequency trading strategies using grammatical evolution, in: IEEE/IAFE Conference on Computational Intelligence for Financial Engineering, Proceedings (CIFEr), 2014, pp. 473–480.

[27] S. Luengo, S. Winkler, D. F. Barrero, B. Castaño, Optimization of trading rules for the spanish stock market by genetic programming, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 9101, Springer, Cham, 2015, pp. 623–634.

[28] J. Gypteau, F. E. B. Otero, M. Kampouridis, Generating directional change based trading strategies with genetic programming, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 9028, 2015, pp. 267–278.

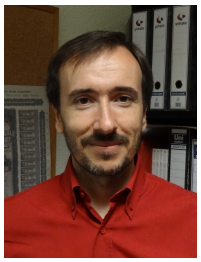
[29] L. Hongguang, J. Ping, Generating Intraday Trading Rules on Index Future Markets Using Genetic Programming, International Journal of Trade, Economics and Finance 6 (2) (2015) 112–116.

[30] A. Pimenta, C. A. L. Nametala, F. G. Guimarães, E. G. Carrano, An automated investing method for stock market

based on multiobjective genetic grogramming, Computational Economics (2017) 1–20.
[31] S. Y. Yang, S. Y. K. Mo, A. Liu, A. A. Kirilenko, Genetic programming optimization for a sentiment feedback strength based trading strategy, Neurocomputing 264 (2017) 29–41.
[32] S. Mousavi, A. Esfahanipour, M. H. F. Zarandi, A novel approach to dynamic portfolio trading system using multitree genetic programming, Knowledge-Based Systems 66 (2014) 68–81.
[33] V. Manahov, R. Hudson, H. Hoque, Return predictability and the 'wisdom of crowds': Genetic Programming trading algorithms, the Marginal Trader Hypothesis and the Hayek Hypothesis, Journal of International Financial Markets, Institutions and Money 37 (2015) 85–98.
[34] A. Agapitos, A. Brabazon, M. O'Neill, Genetic programming with memory for financial trading, in: Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), Vol. 9597, Springer, Cham, 2016, pp. 19–34.
[35] J. M. Berutich, F. López, F. Luna, D. Quintana, Robust technical trading strategies using GP for algorithmic portfolio selection, Expert Systems with Applications 46 (2016) 307–315.
[36] A. Brabazon, M. O'Neill, Evolving technical trading rules for spot foreign-exchange markets using grammatical evolution, CMS 1 (2004) 311–327.
[37] I. Dempsey, M. O'Neill, A. Brabazon, Live trading with Grammatical Evolution, GECCO 2004 Workshop Proceedings (2004) 9137–9142.
[38] I. Contreras, J. I. Hidalgo, L. Núñez-Letamendia, Combining technical analysis and Grammatical Evolution in a trading system, in: Applications of Evolutionary Computing, EvoApplications 2013: EvoCOMNET, EvoCOMPLEX, EvoENERGY, EvoFIN, EvoGAMES, EvoIASP, EvoINDUSTRY, EvoNUM, EvoPAR, EvoRISK, EvoROBOT, EvoSTOC, Vol. 7835, Springer, Berlin, Heidelberg, 2013, pp. 244–253.
[39] I. Contreras, J. I. Hidalgo, L. Núñez-Letamendia, A GA Combining Technical and Fundamental Analysis for Trading the Stock Market, in: A GA combining technical and fundamental analysis for trading the stock market, Springer, Berlin, Heidelberg, 2012, pp. 174–183.
[40] H. Schmidbauer, A. Rösch, T. Sezer, V. S. Tunalioğlu, Robust trading rule selection and forecasting accuracy, Journal of Systems Science and Complexity 27 (1) (2014) 169–180.
[41] R. Aler, D. Borrajo, P. Isasi, Using genetic programming to learn and improve control knowledge, Artificial Intelligence 141 (1-2) (2002) 29–56.
[42] W. B. Langdon, R. Poli, N. F. McPhee, J. R. Koza, Genetic programming: An introduction and tutorial, with a survey of techniques and applications, Studies in Computational Intelligence 115 (2008) 927–1028.
[43] W. Böhm, A. Geyer-Schulz, Exact uniform initialization for genetic programming, in: R. K. Belew, M. Vose (Eds.), Foundations of Genetic Algorithms IV, Morgan Kaufmann, University of San Diego, CA, USA, 1996, pp. 379–407.
[44] I. Rechenberg, Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution, Fromman-Holzbook, Stuttgart, Germany, 1973.
[45] D. J. Montana, Strongly Typed Genetic Programming, Evolutionary Computation 3 (2) (1995) 199–230.
[46] M. O'Neill, C. Ryan, Grammatical evolution, IEEE Transactions on Evolutionary Computation 5 (4) (2001) 349–358.
[47] A. Thorhauer, F. Rothlauf, Structural difficulty in grammatical evolution versus genetic programming, in: Proceedings of the 15th annual conference on Genetic and evolutionary computation - GECCO '13, 2013, p. 997.
[48] D. E. Goldberg, Genetic Algorithms in Search, Optimization, and Machine Learning, Vol. Addison-We, Addison Wesley, 1989.
[49] M. O'Neill, C. Ryan, M. Keijzer, M. Cattolico, Crossover in Grammatical Evolution: The Search Continues, in: Genetic Programming: 4th European Conference, EuroGP 2001 Lake Como, Italy, April 18–20, 2001 Proceedings, Vol. 2038, Springer Berlin Heidelberg, 2001, pp. 337–347.
[50] J. R. Koza, Discovery of a main program and reusable subroutines using genetic programming, in: Proceedings of the Fifth Workshop on Neural Networks: An International Conference on Computational Intelligence: Neural Networks, Fuzzy Systems, Evolutionary Programming, and Virtual Reality, 1993, pp. 109–118.
[51] E. Hemberg, M. O'Neill, A. Brabazon, An investigation into automatically defined function representations in grammatical evolution, in: 15th International Conference on Soft Computing, Mendel, Vol. 9, 2009, pp. 1–6.
[52] C. Ferreira, Automatically defined functions in gene expression programming, Studies in Computational Intelligence 13 (2006) 21–56.
[53] J. R. Koza, Genetic Programming 2: automatic discovery of reusable programs, Artificial Life 1 (4) (1994) 267–292.
[54] S. García, D. Quintana, I. M. Galván, P. Isasi, Multiobjective algorithms with resampling for portfolio optimization, Computing and Informatics 32 (4) (2013) 777–796.
[55] J. L. Person, A complete guide to technical trading tactics : how to profit using pivot points, candlesticks & other indicators, John Wiley & Sons, 2004.
[56] X. Tian, Optimization of intraday trading strategy based on ACD rules and pivot point system in Chinese market, Journal of Intelligent Learning Systems and Applications 04 (04) (2012) 279–284.

[57] A. Wiliński, T. Nyczaj, A. Bera, P. Błaszyński, A study on the effectiveness of investment strategy based on the concept of pivot points levels using matthews criterion, Journal of Theoretical and Applied Computer Science 7 (2013) 42–55.

Carlos Martín is a Graduate in Computer Science from UNED and M.S. in Computer Science and Technology from Universidad Carlos III de Madrid, where he is currently a Ph.D. student. His main interests in the field of Artificial Intelligence are focused on optimization techniques based on Evolutionary Computation. He is an engineer at the Security Operations Center of the Air Force JSTCIS Ciberdefense Directorate in The Spanish Ministry of Defense, Officer in charge of the Forensics, Intrusion Detection, Malware Analysis and Mitigation and Recovery sections.



David Quintana is an Interim Associate Professor with the Department of Computer Science at Universidad Carlos III de Madrid, Spain. There, he is part the bio-inspired algorithms group EVANNAI. He holds a Bachelor in Business Administration and a Ph.D. in Finance from Universidad Pontificia Comillas (ICADE), a Bachelor in Computer Science from UNED and an M.S. in Intelligent Systems from Universidad Carlos III de Madrid. His current research interests are mainly focused on applications of evolutionary computation and artificial neural networks in finance and economics. David is the current Chair of the Computational Finance and Economics Technical Committee of the IEEE Computational Intelligence Society.

Pedro Isasi Graduate and Doctor in Computer Science by the Polytechnic University of Madrid since 1994. Currently he is University professor and head of the Evolutionary Computation and Neural Networks Laboratory in the Carlos III of Madrid University. Dr. Isasi has been Chair of the Computational Finance and Economics Technical Committee (CFETC) of the IEEE Computational Intelligence Society (CIS), Head of the Computer Science Department and Vice-chancellor in the Carlos III University among others. His research is centered in the field of the artificial intelligence, focusing on problems of Classification, Optimization and Machine Learning, fundamentally in Evolutionary Systems, Metaheuristics and Artificial Neural Networks.