Intelligent Systems in Account Finance & Management

Co-evolved Genetic Programs for Stock Market Trading

Jason F. Nicholls
Department of Computer Science, University of Pretoria, Pretoria, South Africa
Andries P. Engelbrecht
Department of Industrial Engineering and Computer Science Division, Stellenbosch University, Stellenbosch, South Africa

Abstract

The profitability of trading rules evolved by three different optimised genetic programs, namely a single population genetic program (GP), a co-operative co-evolved GP, and a competitive co-evolved GP is compared. Profitability is determined by trading thirteen listed shares on the Johannesburg Stock Exchange (JSE) over a period of April 2003 to June 2008. An empirical study presented here shows that GPs can generate profitable trading rules across a variety of industries and market conditions. The results show that the co-operative co-evolved GP generates trading rules perform significantly worse than a single population GP and a competitively co-evolved GP. The results also show that a competitive co-evolved GP and the single population GP produce similar trading rules. The profits returned by the evolved trading rules are compared to the profit returned by the buy-and-hold trading strategy. The evolved trading rules significantly outperform the buy-and-hold strategy when the market trends downwards. No significant difference is identified among the buy-and-hold strategy, the competitive co-evolved GP, and single population GP when the market trends upwards.

Keywords: technical analysis, stock market trading, Johannesburg Stock Exchange, genetic programming, co-evolution, competitive co-evolution, co-operative co-evolution

jason.nicholls@tuks.co.za

1 Introduction

A genetic program (GP) is a domain-independent meta-heuristic algorithm that genetically breeds a population of computer programs to solve a problem (Koza & Poli, 2005). Meta-heuristic algorithms search the solution space by avoiding parts of the search space that produce poor solutions. The result is an approximate solution to the optimisation problem under consideration (Bäck, 1996; Bäck & Schwefel, 1996; Engelbrecht, 2007; Fogel, 2006a).

Originally developed by Koza (Koza, 1992a) in the late 1980s to evolve computer programs, a GP can evolve trading rules using basic mathematical operations. Allen and Karjalainen (Allen & Karjalainen, 1995, 1999) used a GP to evolve trading rules for the S&P 500 index using daily prices from 1928 to 1995. Their approach used a single population of individuals to evolve a trading rule over a fixed number of generations. The single population GP of Allen and Karjalainen was reimplemented by Neely *et al.* (Neely, Weller, & Dittmar, 1997), Telbany (El-Telbany, 2004), Mahfoud and Mani (Mahfoud & Mani, 1996; Mani, Quah, Mahfoud, & Barr, 1995), Li and Tsang (E. Tsang, 2009; E. P. K. Tsang, Li, & Butler, 1998), and Potvina *et al.* (Potvina, Sorianoa, & Vall, 2004) with varying degrees of success.

In nature, populations rarely evolve in isolation. Instead, populations evolve in co-operation with other populations or in competition with other populations (Rosin & Belew, 1997). This paper proposes a co-operative and competitive co-evolutionary approach to GP for evolving trading rules. The performance of trading rules evolved through co-operative and competitive co-evolution is compared to trading rules derived by the single population GP of by Allen and Karjalainen (Allen & Karjalainen, 1995, 1999). It is shown that co-operative co-evolved GP generated trading rules perform significantly worse than trading rules generated by a single population GP and a competitively co-evolved GP. The results also show that a competitive co-evolved GP and the single population GP produce similar trading rules. The profits returned by the evolved trading rules are compared to the profit returned by the buy-and-hold trading strategy. The results show that the evolved trading rules significantly outperform the buy-and-hold strategy when the market, including fees trends downwards. No significant difference is found among the buy-and-hold strategy, the competitive co-evolved GP, and single population GP when the market (including fees) trends upwards.

Section 2 provides a short background on the use of technical analysis in stock market trading and the application of evolutionary algorithms (EAs) in stock market trading. Section 3 discusses Allen and Karjalainen's implementation of a GP for stock market forecasting and how it was adapted for this study. Section 4 describes both co-evolved strategies in detail. Section 5 presents the empirical process and stock market data used for the purpose of this study. Section 6 presents the results of the study. This paper is concluded by Section 7 which presents a summary of the findings and proposals for future work.

2 Background

Stock market analysis is generally *ex-ante*, focusing on the impacts of long-term cash flows, earnings and revenue. This type of *ex-ante* analysis is known as fundamental analysis (Peterson, 2007). Fundamental analysis focuses on analysing company fundamentals such as revenue, assets, production rates, demand, and interest rates. Fundamental analysis often relates back to the share price to determine the future price. Analysis of the share price is known as technical analysis (Peterson, 2007). A belief that the share price reflects all known fundamental information is referred to as the efficient market hypothesis (EMH) (Fama, 1965).

Charles Dow (Cowles, 1933; Edwards, Magee, & Bassetti, 2007; King, 1934) argued that the market is not completely efficient, but that fundamental information rather takes time to propagate through the market. The propagation of fundamental information is therefore reflected in the movement of the share price. Dow believed that if the share price is on the rise, the probability of a continued rise is greater than a fall and vice versa, resulting in a perceived direction overtime known as a trend (Cowles, 1933; James, 1968; King, 1934). Dow proposed the first scientific stock movement theory known as Dow theory (Bishop, 1961; Edwards et al., 2007). Dow theory states that a market is either in an upward trend, downward trend, or continuing in the same direction (Bishop, 1961; Edwards et al., 2007; Rhea, 1993).

Technical analysis techniques use the historic share price to find which of the three Dow trends the market is in and if the trend may change. Knowledge of the trend, and if the trend is changing, gives guidance as to when a trader should buy or sell a share. A simple moving average (SMA) is a technical analysis function used to determine the current market trend. Summing the share price for n consecutive days before a specific day t and dividing the result by n returns the average price of the share over time for the day t.

A comparison of different moving averages can reinforce the trend certainty. For example, if the 10-day SMA follows the same direction as the 50-day SMA, then it can be assumed that the trend is set. If the two moving averages do not follow the same direction, it could signify a trend reversal. Brock *et al.* (Brock, Lakonishok, & LeBaron, 1992) examined the returns generated by various SMAs against the Dow Jones Index from 1897 to 1986. Brock *et al.* showed that when costs are excluded, SMAs can generate profitable buy and sell rules. Many variations of the SMA function exist. Two variations are:

- the weighted moving average (WMA), defined as:

$$WMA(n) = \frac{\omega P_t + (\omega - 1)P_{t-1} + \cdots + P_{t-n+1}}{\omega + (\omega - 1) + \cdots + 1} \tag{1}$$

  where P_t is the opening, closing, high, or low share price on day t, ω is a weight. In this example, ω is equal to n.

- the exponential moving average (EMA), defined as:

$$EMA(n) = \frac{\alpha^n P_t + \alpha^{n-1} P_{t-1} + \cdots + \alpha P_{t-n+1}}{\alpha^n + \alpha^{n-1} + \cdots + \alpha} \tag{2}$$

  where $\alpha = 1 - \frac{2}{n+1}$ and α is the weight.

A common technique used to determine the trend using two different moving averages is to subtract the longer (i.e. slower) moving average from the shorter (i.e. faster) one. A positive result indicates an upward trend, a negative result indicates a downward trend, and a zero result indicates a moment of uncertainty. In the 1960s, Appel (Achelis, 2013; Bäck, Fogel, & Michalewicz, 1997; Tilkin, 2001) subtracted a 26-day EMA from a 12-day EMA, and called this the moving average convergence divergence (MACD) (Achelis, 2013; Tilkin, 2001).

The SMA, EMA, WMA, and MACD are just four of the many technical analysis functions that exist (Edwards et al., 2007). Each function requires a set of parameters that may be unique to a specific stock or market. Technical analysis functions are combined to form a trading rule. An example of a trading rule is to buy shares when the 20-day SMA is greater than its 50-day WMA; otherwise sell the shares held.

Selecting which technical analysis functions to use, and determining the correct parameters to form a trading rule is an optimisation problem. Bauer (Bauer, 1994) was one of the earliest researchers to use a class of meta-heuristics known as genetic algorithms (GAs) (John H. Holland, 1992) to optimise the combination of technical analysis functions and parameters to maximise return on a set of stock trades.

A GA is based on simulated evolution (Bäck et al., 1997; Fogel, 2006b). Simulated evolution breeds a population of candidate solutions called individuals to solve a specific problem (Koza & Poli, 2005). Specifically, GAs iteratively transform a population of individuals into new generations of individuals by applying the analogue of naturally occurring genetic operations. Each individual has a set of phenotypes encoded as genes within a chromosome.

Friedman and Fraser (Bäck et al., 1997; Fogel, 2006b; Friedman, 1956) are recognised as the first to experiment with GAs. However, it was Holland (Bäck et al., 1997; John H. Holland, 1992, 2000; John Henry Holland, 1995) that formalised the first version of a GA. Holland's emulation of evolution used a fixed length binary string representation of a chromosome.

Rather than optimising which technical analysis functions to use with which parameters for a given stock, a technical analysis function can be evolved. To evolve the technical analysis functions from basic mathematical operators requires a variable length chromosome structure that defines a relationship between genes within the chromosome. The chromosome structure must cater for binary, logical and numerical functions. Koza (Koza, 1992b) experimented with an alternative version of a GA called a genetic program (GP). A GP replaces the fixed length vector structure used in a GA with a random length, non-linear, hierarchical abstract data structure, known

as a *tree*. It is possible to encode logical formulae, arithmetic formulae, or computer programs using a tree structure.

Allen and Karjalainen (Allen & Karjalainen, 1995, 1999) were one of the earliest implementers of GPs for stock market trading. Using data from the S&P 500 index and the New York Stock Exchange (NYSE), Allen and Karjalainen (Allen & Karjalainen, 1995, 1999) showed that evolved trading rules do not earn excess returns over a simple buy-and-hold strategy after transaction costs. The trading rules enter the market when returns are positive and daily volatility is low and stay out of the market when the returns are negative and volatility is high. The trading rules are sensitive to trading costs, but showed robustness to the impact of the 1987 stock market crash (Allen & Karjalainen, 1995, 1999). Lowering the trading costs increases the returns of the GP (Allen & Karjalainen, 1995, 1999). Allen and Karjalainen noted that, while their approach produced trading rules that generated profitable returns, their GP is relatively simple and the parameters of the GP were not optimised (Allen & Karjalainen, 1995, 1999).

Variations of Allen and Karjalainen's implementation of a GP were implemented by Neely *et al.* (Neely et al., 1997), Telbany (El-Telbany, 2004), Mahfoud and Mani (Mahfoud & Mani, 1996; Mani et al., 1995), Li and Tsang (E. Tsang, 2009; E. P. K. Tsang et al., 1998), and Potvina *et al.* (Potvina et al., 2004) with varying degrees of success.

3 Genetic Programming for Trading

This section discusses the single population GP. Section 3.1 describes a trading rule and how it is encoded within a chromosome. Section 3.2 presents the fitness function implemented in this study. Section 3.3 presents the singple population GP algorithm.

3.1 Trading Rule

Each individual within a GP represents a trading rule. Trading rules are encoded within the individual's chromosome as a tree. Each tree consists of *terminal nodes* (i.e. nodes with no successors) and *non-terminal nodes*. Terminal nodes correspond to input parameters. Each node provides arguments (or parameters) to the function defined in the node's predecessor. The entire tree is interpreted as a function, which is evaluated recursively starting from the root node of the tree. Each node within the tree is referred to as a gene.

This study uses the functions and input parameters defined by Allen and Karjalainen (Allen & Karjalainen, 1995, 1999). Non-terminal nodes are defined as either a real number function or a boolean function, respectively returning a real number or a boolean value.

Allen and Karjalainen (Allen & Karjalainen, 1995, 1999) used different real number functions to include "average", "minimum", "maximum", "arithmetic op-

erators", "lag" and "norm". "Minimum" returns the minimum of two real-valued parameters. "Maximum" returns the maximum value of two real-valued parameters. The "arithmetic operators" perform their functions $(+, -, \div, \times)$ on two real-valued parameters. "Lag" returns the price of the share n days prior, where n is a natural number. "Average" returns the SMA for the n days prior. "Norm" returns the absolute value between the difference of two real-valued parameters.

The boolean functions includes "if-then-else","and","or","not", "<" and ">". "If-then-else" requires three boolean parameters, "and" requires two boolean parameters, "or" requires two boolean parameters, and the logical comparison operators ("<" and ">") require two real-valued parameters. Parameters are defined as the children of the non-terminal nodes.

Terminal nodes are defined as either the boolean constants `true` or `false`, "close price", "high price", "low price", "volume", a real number constant, or a natural number constant. Real number and natural number constants are drawn from a uniform distribution between a minimum and maximum configured value during initialisation of the GP. "Close price" is defined as the price on that day's trade. "High price" is defined as the highest price on that day's trade. "Low price" is defined as the lowest price on that day's trade. "Volume" is defined as the volume traded on that day.

The evaluated root node returns a boolean value of `true` or `false`, which respectively refers to buy or sell. The trader holds position if it has shares on hand and the root node is evaluated to buy. The trader holds position if is has no shares on hand and the root node is evaluated to sell. The evaluated action is described in Table 1.

For an evolved rule to be grammatically correct, the root node has to return a boolean value. Each function within the function set has a set arity or number of parameters. A function with an arity n has n branches associated with it. Each parameter within the function has a required type. Each associated branch returns a value of the required parameter type. If a child node returns a value different from the parameter expected by the parent node, the tree is considered grammatically incorrect.

Three parameter types are defined to ensure that a function is passed the correct parameter type. These types include: natural numbers, numeric, and boolean. A numeric parameter can either be a natural number or a real number. The boolean constants are either `true` or `false`, while the real number constants are drawn from a uniform distribution between configured minimum and maximum values, and natural numbers are drawn from an uniform distribution between configured minimum and maximum natural numbers. Allen and Karjalainen (Allen & Karjalainen, 1995, 1999) did not specify the configured minimum and maximum natural numbers, except that they are parameters to the lag, and average functions. For this study, natural numbers are drawn from a uniform distribution between 1 and 500. The reason for choosing 1 and 500 is that the dataset samples used do not exceed 500 days.

Allen and Karjalainen (Allen & Karjalainen, 1995, 1999) specified the limits of

the real numbers to be between 0.0 and 2.0, without offering any explanation why. A positive real number limit between 0.0 and 2.0 in conjunction with positive natural number limits does not allow the evolution of negative numbers. Some technical analysis functions such as the oscillating indexes return values within limits such as -100.0 and 100.0 or -1.0 and 1.0. Changing the real number limits from 0.0 and 2.0 to -1.0 and 1.0 allows the GP to evolve sub-trees that return negative numbers such as -100.0 (-1.0×100).

The mathematical operator "$\div$" can invalidate a trading rule, as division by zero is undefined. The second parameter of the "$\div$" operator is therefore restricted to a non-zero numeric constant.

Figure 1 presents two different trading rules encoded as trees. The left tree corresponds to a 50-day moving average, returning a buy action if the closing price is greater than the 50-day SMA, otherwise a sell action is returned. The tree on the right depicts a break out rule. If the day's trading price is greater than the maximum price over the last 30 days, then a buy action is returned, otherwise a sell action is returned.

3.2 Fitness measure

A fitness function quantifies an individual's performance Engelbrecht, 2007. Performance is generally determined by how well the candidate solution completes the problem. Because the fitness measure is viewed independently of any other solution within the population, it is referred to as the *absolute* fitness Angeline and Pollack, 1993. The aim of a trading rule is to make the most profit at the end of trading. Allen and Karjalainen (Allen & Karjalainen, 1995, 1999) used a fitness measure based on the compounded excess returns over the buy-and-hold strategy during a trading period. The excess return is given by:

$$\Delta r = r - r_{bh} \tag{3}$$

where the continuous compounded return, r, of the trading rule is computed as

$$r = \sum_{t=1}^{T} r_i I_b(t) + \sum_{t=1}^{T} r_f I_s(t) + n \log\left(\frac{1-c}{1+c'}\right) \tag{4}$$

$$\tag{5}$$

with

$$r_i = \log P_t - \log P_{t-1}$$

The return for the buy-and-hold strategy, r_{bh}, is calculated as

$$r_{bh} = \sum_{t=1}^{T} r_t + n \log\left(\frac{1-c}{1+c'}\right) \tag{6}$$

In the above, c denotes the one-way transaction cost; r_f is the risk free cost when the trader is not trading; $I_b(t)$ is equal to one if a rule signals a buy, otherwise $I_b(t)$ is equal to zero; $I_s(t)$ is equal to one if a rule signals sell, respectively, otherwise $I_s(t)$ is equal to zero; n denotes the number of trades and r_{bh} represents the returns of a buy-and-hold, while r represents the returns of the trader.

The continuous compounded return function rewards an individual when the share value is dropping and the individual is out of the market. The continuous compounded return function penalises the individual when the market is rising and the individual is out of the market.

Each individual within the population is evaluated on a given dataset by the fitness function. The better the evaluation, the higher the probability that the individual will remain for successive generations. At the end of evolution the individual that performed the best on a given dataset is selected as the solution. The individual represents a forecasting model for the observed dataset. If the model corresponds too closely or exactly to the dataset, the model may fail to reliably predict future observations (i.e. generalisation will be poor); the model is said to over-fit. Allen and Karjalainen (Allen & Karjalainen, 1995, 1999) reserved a selection period immediately following the training period for validation of the evolved trading rules. After each generation, they apply the fittest rule within the generation, based on the excess return in the training period, to the selection period. If the excess return in the selection period improves upon the best previously saved rule, the new rule is saved. At the end of evolution the saved rule is returned as the solution.

3.3 Algorithm for single Population Genetic Program

The pseudo code listing presented in Algorithm 1 outlines the single population evolutionary process as defined by Allen and Karjalainen (Allen & Karjalainen, 1995, 1999). An initial population of μ individuals is created stochastically. The fitness of each individual is calculated by applying the trading rule to the training dataset ($Sample_{in}$) and calculating the fitness measure using the fitness function. The best individual (B_g) within the initial population is selected and saved as the global best individual (B). The initial population is referred to as generation 0.

The algorithm uses a steady state population model to allow stronger individuals to survive many generations. A probability factor determines if mutation is performed.

If mutation does not occur, rank selection biased towards the best individuals is used to sample two individuals from the population to undergo reproduction. Rank selection uses a relative fitness value derived from the absolute fitness (Bäck et al., 1997). The absolute fitnesses of the individuals are calculated using the fitness function. Individuals of the population are sorted by their fitness value from best to worst. The worst individual is assigned a rank value of 1, the second worst a rank value of 2, and so on until the best individual has rank n, where n is the total number

of individuals. A probability of selection is assigned to each individual in the current generation using:

$$p(x_i) = \frac{r(x_i)}{\sum_{j=1}^{n} r(x_j)} \tag{7}$$

where x_i is the ith individual within the population, p is the probability of selection for individual x_i, and r is the rank of individual x_i.

The selected parents undergo reproduction using a crossover operator to form offspring. Offspring contain genetic material from both parents. To perform crossover between two trees, a node is selected at random from the first tree. A sub-set of nodes are selected from the second tree, such that the nodes selected are valid grammatical replacements for the node selected in the first tree. A random node is then selected from this sub-set. The sub-trees of the selected nodes are then swapped. An example of crossover using two trees is illustrated in Figure 2. Node (4) from the left parent is selected at random. Node $(-)$ from the right parent is selected at random. The nodes and their sub-trees are then swapped to form two offspring trees.

The new offspring are added to the population. Rank selection samples two individuals from the population with a bias towards the worst individuals. The two selected individuals are then removed from the population.

Allen and Karjalainen introduced mutations by randomly generating a new tree in place of the second parent used in crossover (Allen & Karjalainen, 1995, 1999). This technique is known as headless-chicken mutation (Jones et al., 1995). Headless-chicken mutation, while very simple in concept, is computationally more expensive than altering a single node. For example, suppose that a terminal node is mutated, headless-chicken requires that an entire individual is initialised and then crossed over with the selected parent at the selected terminal node. In this case only one node is altered. It is computationally more efficient to alter the terminal node through application of local mutation operators. For this reason two mutation operators are implemented, namely grow mutation and prune mutation as illustrated in Figure 3. Prune mutation replaces a randomly selected node with a syntactically correct terminal node. Grow mutation replaces a randomly selected node with a new stochastically generated syntactically correct sub-tree.

Mutation samples an individual from the population using rank selection biased towards the worst individual within the population. The sampled individual is mutated to form an offspring. The offspring is evaluated and added to the population. An individual is then removed from the population using rank selection biased towards the worst individual. The offspring count λ is incremented by 1.

The processes of crossover and mutation are repeated until the total number of generated offspring λ is equal to or greater than the configured population size μ. Because individuals are removed during crossover and mutation, the size of the population remains constant.

After each generation, the best rule in the population is applied to a validation

dataset ($Sample_{sel}$). If the rule leads to a higher excess return than the best rule so far, the new rule is saved. The evolution terminates when a maximum number of generations is reached. The best trading rule is then applied to the out-of-sample dataset ($Sample_{out}$) immediately following the validation dataset.

4 Co-evolved Genetic Programming for trading

Allen and Karjalainen (Allen & Karjalainen, 1995, 1999) implemented a single population GP. In nature, populations rarely evolve in isolation. Instead, populations co-evolve in co-operation with other populations or in competition with other populations.

Co-evolved populations evolve in isolation ensuring that each population can explore the solution space independently (Fogel, 2006a), reducing the probability of premature convergence. Co-operative co-evolution (Fogel, 2006a; Hillis, 1990) results in populations evolving parts of the solution independently. The performance of individuals are based on the performance of all the populations. Competitive co-evolution (Hillis, 1990) results in populations competing against each other. The performance of individuals in one population are indirectly proportional to the performance of individuals in other populations.

The single population GP discussed in Section 3.3 is extended in this section to incorporate co-evolution. The co-operative co-evolved GP is presented in Section 4.1, while Section 4.2 presents the competitive co-evolved GP.

4.1 Co-operative Co-evolved Genetic Program

Co-operative co-evolution requires two or more populations that evolve together. Each population benefits from the other populations through information sharing and reward sharing. Each population undergoes evolution independently. The single population evolutionary process defined in Section 3.3 is divided into two parts to support co-operative co-evolution. The first part, presented in Algorithm 2, initialises multiple populations, iterates through the generations, and returns the global best individuals from each population. The second part, illustrated in Algorithm 3, evolves each population independently.

The key differences between the co-operative co-evolution algorithm presented here and the single population GP are that more than one population is evolved, a reward sharing mechanism is used to determine the best individuals, and multiple individuals are returned. A dual population approach is implemented: one population evolves a buy rule and the other a sell rule. Two individuals are required to form a trading rule: one individual from the buy population, and one individual from the sell population.

Evaluation of co-operatively co-evolved individuals is more complex than the single population GP. Each co-operatively co-evolved individual is part of a solution and must share its fitness value with an individual from another population. Holland

(Casas, 2015; John, 1985; Seshadri, 2003) devised the "bucket brigade" credit scoring system. The "bucket brigade" is modelled after the water passing chains of fire fighters. Each type of bucket passes through a set of fire fighter chains. The best bucket, is the one that loses the least amount of water as it passes along the various chains. The best fire fighter chain is the one that loses the least amount of water across all the buckets the fire fighters pass along. A "bucket brigade" credit system can be used to evaluate co-operatively evolved individuals. The fitness value of each individual is the sum of all the evaluations the individual takes part in. Each individual from the buy population is evaluated with each individual from the sell population, an approach followed by De Jong and Potter (Casas, 2015; Seshadri, 2003) and Axelrod (Axelrod, 2006; Seshadri, 2003). The buy and sell rule combination is evaluated using the compounded excess return fitness function defined in Section 3.2. The total reward of both the individuals used in the evaluation are incremented by the fitness value.

The process of buying and selling shares is illustrated in Algorithm 5. *EvaluateCombination* receives three parameters. The first parameter is an individual that represents a buying rule, the second is an individual that represents a selling rule. A buy and sell individual is defined by the same grammar and syntax as the single population GP. The third parameter represents the dataset used for evaluation.

A trader has no shares on the first day of trading. The trader executes the trading rule represented by an individual from the buy population. If the buy rule returns `true`, the trader must buy shares, otherwise no shares are bought. If the trader has no shares on day two, the trader executes the buy trading rule again. If the trader has shares, the trader executes the sell trading rule. If the sell trading rule returns `true`, the trader must sell the shares. If the sell trading rule returns `false`, the trader does nothing. This process continues for each trading day. On the last day of trade any shares in the possession of the trader are sold. Each trading action is used by the fitness function to calculate a fitness value.

The co-operative co-evolution GP implemented maintains two global best individuals, one from each population. Because the reward value of an individual is the sum of the fitness values returned by all the evaluations an individual takes part in, the global best individual is included in the reward sharing process.

The complete reward sharing process is illustrated in Algorithm 4. The function *RewardSharing* receives five parameters. The first two parameters represent the individuals from the buy population as the array *cBuy* and the global best individual from the buy population. The second two parameters represent the sell population, and the final parameter contains the dataset. The reward process evaluates every combination of individuals from the buy population and the sell population using the function presented in Algorithm 5. The fitness result is added to the individual's accumulated reward. The reward process is repeated by evaluating the combination of each individual in the buy population and the global best individual from the sell

population, and again combining each individual within the sell population and the global best individual from the buy population. The accumulated reward of each individual is returned.

4.2 Competitive Co-evolved Genetic Program

Competitive co-evolution requires two or more populations that evolve in competition, each population reciprocally driving one another to increasing levels of performance and complexity (Casas, 2015; Rosin & Belew, 1997). The performance of one population is inversely proportional to the performance of another population, implying that the fitness of one individual is dependent on the fitness of another individual. Angeline and Pollack (Angeline & Pollack, 1993) defined a relative fitness function as any fitness calculation that is dependent on the quality of other individuals (Angeline & Pollack, 1993).

The single population GP is modified to support a relative fitness function, and more than one population. The modified evolutionary process is presented in Algorithm 6. Instead of a single population, the competitively co-evolved algorithm begins by initialising P populations of equal size. Individuals within a population are initialised using the same approach as the single population GP. The size of each population is determined by the configured total number of individuals μ divided by the number of populations P.

The evaluation of a competitively co-evolved individual is determined by a relative fitness value as opposed to the absolute fitness value used in a single population GP. The relative fitness calculation is presented in Algorithm 8 as two functions: `CalcAllRelativeFitness` and `CalcRelativeFitness`. The function `CalcAllRelativeFitness` requires two parameters. The first parameter C_g contains all the individuals from every population. The second parameter denotes the current population p. `CalcAllRelativeFitness` iterates through each individual in population p calling function `CalcRelativeFitness`.

`CalcRelativeFitness` compares an individual from population p to all individuals in all other populations. In order to do these comparisons the absolute fitness of each individual is calculated to determine which individual has the largest absolute fitness value. The individual from population p receives a point each time its absolute fitness value is greater than the absolute fitness value of the individual it is compared too (Casas, 2015; Hillis, 1990; Rosin & Belew, 1997). `CalcRelativeFitness` returns the total awarded points as the relative fitness value of the individual.

The competitive co-evolved GP maintains a global best individual for each population. At the end of each generation the relative fitness value of the global best individuals are re-calculated by comparing the global best individual to all the individuals in all the other populations. The best individual from the current population is the individual that has the largest relative fitness value across all the individuals within the current population. The best individual from the current population is then compared to the global best individual from the current population. The itera-

tion best individual from the population becomes the new global best individual, if its $Sample_{in}$ and $Sample_{sel}$ relative fitness values are greater than the relative fitness values of the global best individual of that population. At the end of evolution, the competitive co-evolved GP returns the overall global best individual as the solution.

5 Empirical Procedure

Individuals within a single population GP are quantified using an absolute fitness function, while individuals within a co-evolved GP are quantified using a relative fitness function. To ensure that trading rules evolved through single population evolution or co-evolution are compared fairly, the profit returned at the end of trading is used to quantify the performance of a trading rule. Profit was calculated as

$$Profit = \left[\sum_{t=1}^{T}(P_t - sc) \times I_s(t)\right] - \left[\sum_{t=1}^{T}(P_t + bc) \times I_b(t)\right]$$

where $I_b(t)$ is equal to one if a rule signals a buy, otherwise $I_b(t)$ is equal to zero; $I_s(t)$ is equal to one if a rule signals sell, otherwise $I_s(t)$ is equal to zero; sc represents the selling cost and bc the buying cost; T is the number of trade days; P is the closing price on day t.

The profit returned by an evolved trading rule is referred to as an observation. A sample is a set of observations returned by a simulation run. A simulation run is a fixed configuration of a GP. Each simulation run was repeated 30 times, from different initial conditions, recording the results as individual observations. A simulation run was repeated for all share datasets independently.

Each configuration of a simulation run was intended to test the effect of the configuration on the observation. However, each observation within a simulation run requires an initial set of randomly generated individuals. Naturally, if two GPs are initialised with two different populations of individuals, the observations could be different. Furthermore, each decision such as whether to implement mutation or crossover depends on a random probability that affects the observation. To minimise the effect of randomness, a fixed set of 30 random seeds were used; one seed for each of the 30 GP executions within a simulation run. The same 30 random seeds were used across all simulation runs.

The random seed ensures that the GP returns the same observation no matter how may times it is run. The random seed ensures that the GP is initialised with the same initial population. This means that if two GPs are run with the same random seed but a different configuration, then the change in observsation is due to the configuration change and not the initial population.

To compare the samples of two or more different simulation runs the *null-hypothesis* was assumed (Bluman, 1995). The null-hypothesis assumes that the samples of various simulation runs are the same, and that no further investigation is

required unless the samples are significantly different. A statistical test is performed on the results to determine if two or more samples rejected the null-hypothesis.

The most common statistical tests used to evaluate the validity of the null-hypothesis are the t-tests and the Wilcoxon signed rank tests (Bluman, 1995; Demšar, 2006; García, Fernández, Luengo, & Herrera, 2008). These tests are not adequate when comparing more than two samples due to the multiplicity effect (Graczyk, Lasota, Telec, & Trawiński, 2010; Salzberg, 1997). EA empirical analysis have shown that the results do not fulfil the requirements for parametric tests such as t-tests (Graczyk et al., 2010; Sheskin, 2003). Therefore, a common non-parametric statistical test known as a Krushkal-Wallis test is used to test the validity of the null-hypothesis across more than two samples.

The Krushkal-Wallis test is a non-parametric test based on a one-way Analysis of Variance (ANOVA) between the sample ranks. The Krushkal-Wallis test indicates if at least one sample is stochastically different from the other samples. The Krushkal-Wallis test does not show which samples are different (Graczyk et al., 2010). If the Krushkal-Wallis test rejects the null-hypothesis, a post-hoc test is required to determine which sample rejects the null-hypothesis (Demšar, 2006; Graczyk et al., 2010).

Post-hoc analysis performs a pairwise comparison of sample results to determine which sample results are different from one another. Traditionally, Wilcoxon signed rank tests are used for post-hoc analysis (Bluman, 1995; Demšar, 2006; García et al., 2008; Graczyk et al., 2010).

The statistical tests return a *p-value*. The p-value is a number between 0 and 1 denoting the probability that the null-hypothesis is valid. A p-value of less than 0.05 indicates a strong possibility that the null-hypothesis is incorrect and that the samples are not the same.

Demšar (Demšar, 2006) noted that ANOVA is based on assumptions which are most probably violated when testing the results of EAs. ANOVA assumes that the samples are drawn from normal distributions. Demšar pointed out that there is no guarantee for normal distribution across a set of algorithms or various configurations of algorithms. The second more important assumption of ANOVA is that the random variables such as control parameters and the initialised population have equal variance, which, as Demšar noted, can not be taken for granted (Demšar, 2006).

Demšar (Demšar, 2006) proposed that the Friedman test be used for the pairwise comparisons, and that the Iman and Davenport extension (Iman & Davenport, 1980) to the Friedman statistical test be used for multiple results comparison in place of a one-way ANOVA.

If the null-hypothesis is rejected, Demšar proposed the Nemenyi test as a post-hoc pairwise test (Demšar, 2006). The Nemenyi test produces a critical difference value which is easily displayed graphically as a critical difference plot. An example of a critical difference plot is presented in Figure 4. The top line in the figure represents the average ranks. The lowest or best ranks are on the right. The critical difference is

displayed above the graph. Algorithms similar to the sample with the highest mean are within the critical difference marking. Similar algorithms are plotted to the left and the right of the sample with the highest mean. Groups that are not significantly different are connected by a line.

Allen and Karjalainen (Allen & Karjalainen, 1995, 1999) used a GP to evolve trading rules for the S&P 500 index using daily prices from 1928 to 1995. This paper examines the profitability of evolved trading rules on the JSE. The datasets used in this study were sourced from Sharenet. Sharenet is a market statistics company based in Cape Town, South Africa. The data contained eleven shares from a variety of market sectors; including mining, resources, consumer goods and services, industrials, banking, and insurance. The stocks are listed in Table 2 and cover the period April 2003 to June 2008. The start date is referred to as day 0. The stock market dataset spans 1 350 days. Trading days exclude weekends and public holidays. Simulation runs reference the share code, which is the code used by the JSE.

Each simulation run was repeated for each share within the share datasets. In addition to company shares, the data includes the ALSI40 market index. The ALSI40 index constitutes the 40 largest companies by market capitalisation across all sectors listed on the JSE. In addition to the datasets listed in Table 2, fictitious datasets known as *inverted shares* were created by reversing the share price of Nedbank, Remgro and Standard Bank. To reverse the dataset, a new list of inverted prices were generated where the inverted price on day 0 corresponds to the original price data on day 1 350. The inverted price on day 1 corresponds to the original price on day 1 249. The inverted price on day 2 corresponds to the original price on day 1 248, and so on.

Each share dataset was divided into three continuous samples. $Sample_{in}$ spans two years of trading days from day 0 to day 500. $Sample_{sel}$ spans one year of trading days from day 500 to 750. $Sample_{out}$ spans two years of trading days from day 750 to 1250.

Trading costs were calculated using the 2008, Standard Bank online trading platform fees which include:

- a headline brokerage fee of 0.5%, with a minimum of R80,
- security transfer tax of 0.25%,
- strate tax of 0.005787%, with a minimum strate of R11.58 and a maximum of R57.87,
- VAT of 14% (the study was done before the recent VAT increase to 15%), and
- a financial service board fee of 0.0002%.

Buy and sell transactions incur the same fees, except for the security transfer tax which is payable on purchases and not sales.

GPs require values to be assigned to control parameters. Four control parameters were identified namely mutation probability, number of individuals within a

population, maximum number of generations, and the maximum number of nodes within the chromosome.

To select the most appropriate values for each control parameter, a response surface methodology (De Lima, Pappa, de Almeida, Gonçalves, & Meira, 2010) known as $2^k r$ factorial design (Shahsavar, Najafi, & Niaki, 2011) was used. Each control parameter k was assigned an upper and lower limit. A simulation run was performed for each combination of control parameters. The observations were recorded and compared using the statistical tests defined.

The $2^4 30$ factorial design results showed that a mutation probability of 35%, a population size of 1 600 individuals, a maximum number of 125 generations, a maximum of 100 nodes, and a maximum tree depth of 10 were preferred control parameter values for this study (for a detailed discussion of the results see (Nicholls, 2018)).

6 Empirical Analysis

The co-evolved adaptations of the standard GP defined by Allen and Karjalainen (Allen & Karjalainen, 1995) were evolved without fees using the datasets provided. The mean profit returned is presented in Table 3. The mean profit is calculated as the mean observation profit across all observations within the sample.

The results of the evolved traders were compared using the empirical process defined in Section 5. An Iman and Davenport test comparing the results returned by the three GP's trained on different share datasets returned p-values greater than 0.05 for ALSI, GFI, IMP and LON. The p-values for these share results were 0.160536, 0.707859, 0.991976, and 0.707859 respectively. The remainder of the datasets resulted in a p-value less than 0.00001. This shows that a significant statistical difference exists between the results returned by the three different GPs.

Post-hoc analysis was performed on the significantly different trading rule results to determine which GP resulted in significantly different profit results. The mean profit results in conjunction with the statistical test results show that the co-operative co-evolved GP performed significantly worse than the other two approaches for 10 of the 11 $Sample_{out}$ share datasets. The box plots presented in Figure 5 depict how poorly the co-operative co-evolved trading rules performed. Except for NED, the results returned by the evolved trading rules using the standard single population GP and the competitively co-evolved GP have smaller boxes denoting a smaller deviation in the results, and a higher mean than the co-operative co-evolved GP across all the significantly different profit results.

The box plots in Figure 5 in conjunction with the mean results in Table 3 show that the competitively co-evolved trading rules performed better than the trading rules evolved using the single population GP when trading ALSI40, BIL, NED, REM, SBK, SAB, and SOL. The trading rules evolved using the single population GP performed better than the competitively co-evolved trading rules when trading INVNED, INVREM, INVSBK, and RCH. However, except for the INVREM results,

the Friedman test failed to find a significant difference between the results returned by the trading rules evolved using competitive co-evolution and the results returned by the trading rules evolved using single population evolution.

The results show that the trading rules evolved using competitive co-evolution and single population evolution performed equally well across all shares, except for NED and INVREM. When trading NED, a cooperative co-evolved GP was preferred. When trading INVREM, a single population GP was preferred.

The simulations were repeated using transaction fees. The mean profit returned by the evolved trading rules are presented in Table 4. The results show that the competitive co-evolved trading rules returned the highest mean profit in 11 of the 15 shares and the co-operative co-evolved trading rules produced the lowest mean profit across 14 of the 15 shares.

The mean standard deviation of profit (σ) results presented in Table 4 show that the competitively co-evolved trading rules were the most consistent resulting in the lowest deviation across 11 of the 15 shares. The mean standard deviation of profit (σ) is defined as

$$\sigma = \sqrt{\frac{1}{N-1}\sum_{i=1}^{N}(x_i - \overline{x})^2} \tag{8}$$

where $x_1, x_2, \ldots, x_N$ are the observed values of the sample items (i.e. profit with or without cost), $\overline{x}$ is the mean value of these observations, and N is the number of observations in the sample. The mean standard deviation of profit is used to determine how close each individual was to the profit mean. The lower the mean standard deviation, the similar the trading rules.

The Iman and Davenport (Iman & Davenport, 1980) statistical test returned a p-value < 0.002 for all trading result comparisons. Further post-hoc analysis showed that the profit results returned by the co-operatively co-evolved trading rules were significantly different from the results of the other two algorithms. The critical difference plots presented in Figure 6, in conjunction with the mean profit results of the best individuals, confirm that the trading rules evolved using the co-operative co-evolved GP performed significantly worse than the other trading rules.

The p-values obtained from the post-hoc analysis of the INVNED and SOL profit results show that the results obtained from the trading rules evolved using a competitive co-evolved GP are significantly different from that of the single population GP results. The results show that the trading rules evolved using the competitive co-evolved GP performed significantly better than the trading rules evolved using the standard GP when trading SOL. However, trading rules evolved using the standard GP performed significantly better than the trading rules evolved using the competitive co-evolved GP when trading INVNED. The trading rules evolved using a competitive co-evolved GP performed better than the evolved trading rules using the single population GP for 10 of the 15 shares.

The profit returned by the trading rules evolved using the single population GP

and competitive co-evolved GP were compared to a buy-and-hold trading strategy to test the trading rules' performance against the market's performance. The results are presented in Figure 7

The GFI, INVNED, INVREM, and INVSBK profit results returned by the buy-and-hold strategy were significantly less than the results returned by the co-evolved and evolved trading rules. The NED profit results returned by the buy-and-hold strategy were significantly more than the results returned by the co-evolved and evolved trading rules. The evolved trading rules returned a positive trading balance at the end of trading INVNED, INVREM, and INVSBK, compared to the buy-and-hold strategy that returned a positive trading balance at the end of INVREM.

The results show that the trading rules evolved using either a competitive co-evolved GP or a standard GP can return a profit significantly greater than the buy-and-hold strategy if the shares are trending down, or are highly volatile. However, when the share price is trending upwards; the trading rules evolved using either a competitive co-evolved GP or a standard GP performed as well as the buy-and-hold strategy.

7 Conclusions and future work

This paper proposed two co-evolution variations of the single population GP for stock market trading. The first variation was that of a co-operative co-evolved GP. The second variation was a competitive co-evolved GP. The three different GPs were run against 15 different share datasets. Each dataset was sub-divided into three different samples. An in-sample dataset used for evolving a generation. A selection or verification sample dataset used to validate the best individual, and an out-of-sample dataset to test the performance of the evolved rules on unseen data.

The co-operative co-evolved trading rules performed significantly worse than the single population GP and the competitive co-evolved GP across all the shares, with and without fees than the other trading rules. Trading rules evolved by the competitively co-evolved GPs performed better, but not significantly better than the trading rules evolved by the standard single population GP, for 10 of the 15 shares. The results also showed that the competitively co-evolved GP evolved trading rules that performed significantly better than the trading rules evolved by the standard GP when trading SOL. However, trading rules evolved using the standard GP performed significantly better than trading rules evolved by the competitively co-evolved GP when trading INVNED.

No significant difference was found between the results returned by the trading rules and the results returned by the buy-and-hold strategy for 10 of the 15 shares. The shares that showed significant difference were shares trending downwards. In these cases the evolved traders performed significantly better than the buy-and-hold strategy. The trading rules significantly out-performed the buy-and-hold for four of the 15 different shares.

This study presented evidence showing that trading rules evolved using either a competitive co-evolved GP or a standard GP can return a profit significantly greater than the buy-and-hold strategy if the shares are trending down, or are highly volatile. However, when the share price is trending upwards, trading rules evolved using either a competitive co-evolved GP or a standard GP performed as well as the buy-and-hold strategy.

The study found that the competitive co-evolved GP produced more reliable trading rules and performed generally better than trading rules evolved using a single population GP.

Several areas are identified for future research to improve the competitive co-evolved GP in stock market trading rule generation. The competitive co-evolved GP used two populations and a simple bipartite relative fitness as defined by Hillis (Casas, 2015; Hillis, 1990; Rosin & Belew, 1997). Future work could include the work of Rosin and Belew (Casas, 2015; Rosin & Belew, 1997) by studying the effect that competitive fitness sharing, shared sampling, and the hall-of-fame has on the performance of the evolved trading rules.

This study shows that the market conditions used during evolution affect the performance of a trading rule when exposed to the out-of-sample data. To evolve the trading rules, this study divided datasets into three distinct continuous samples of fixed sizes. A suggestion is to automatically classify continuous samples within the dataset, determining the trend of the sample. Then to combine continuous samples of varying trends into an in-sample dataset and selection dataset. The intent is to expose the evolutionary process to share trends that evolve the best out-of-sample trading rules.

The mutation operator used in this study implemented both a grow mutation operator and a prune mutation operator with equal probability. Future work could study the effect the probability has on the performance of evolved trading rules. Furthermore, the probability could change overtime. For example, while the population has a small average tree size, the mutation operator may favour growing the chromosome. When the population has a large average tree size the mutation operator may favour pruning the chromosomes.

Future work could explore the impact that a dynamic mutation probability has on the profitability of evolved trading rules. The dynamic mutation probability could start high allowing the GP to explore the search space, and overtime the mutation probability could decrease, resulting in less exploration and more optimisation.

Table 1
Trading actions based on the result of a trading rule and shares on hand as defined by Allen and Karjalainen (Allen & Karjalainen, 1995, 1999).

Result	**Has shares**	**No shares**
`true`	Hold	Buy
`false`	Sell	Hold

Table 2
The share data used in the simulations.

Code	Name	JSE Sector
AGL	Anglo American Plc	Metals & Minerals
ALSI	JSE All Share Index	Traded Fund
BIL	BHP Billiton Plc	Metals & Minerals
GFI	Gold Fields Ltd	Gold Mining
IMP	Impala Platinum Holdings Ltd	Platinum
LON	Lonmin Plc	Platinum
NED	Nedbank Group Ltd	Banks
RCH	Richemont Securities AG	Clothing & Footware
REM	Remgro Ltd	Diversified Industrials
SAB	Sabmiller Plc	Beverages & Brewers
SBK	Standard Bank Group Ltd	Banks
SOL	Sasol Ltd	Oil & Integrated

Table 3
Mean $Sample_{out}$ profit per share without fees.

Share Code	Co-Operative	Competitive	Standard
AGL	*17518.23*	**25381.37**	25139.13
ALSI	*4783.57*	**9316.80**	9065.63
BIL	*5870.03*	**14118.47**	13075.57
GFI	-419.67	**131.97**	*-1280.03*
IMP	*13243.33*	**16291.73**	15982.73
INV-NED	*929.77*	12892.07	**13091.80**
INV-REM	*-1251.40*	6323.30	**7977.53**
INV-SBK	*130.83*	5037.13	**6693.43**
LON	*13807.13*	**17522.17**	15756.33
NED	**-219.60**	-2279.83	*-2848.47*
RCH	*641.73*	2184.63	**2187.70**
REM	*5108.03*	**9835.50**	9762.33
SAB	*1868.63*	**5453.03**	4969.83
SBK	*249.20*	**1692.23**	1555.63
SOL	*5010.50*	**16213.47**	13053.43

Note: **bold** represents the highest value, and *italic* the lowest value.

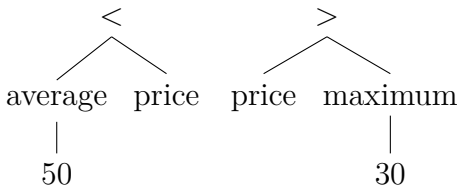
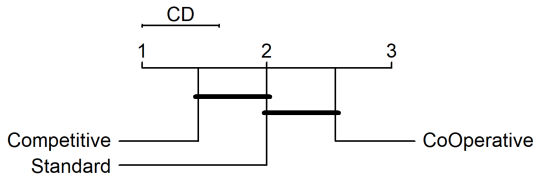


Figure 1. Two examples of trading rules encoded as trees.

Table 4
Mean $Sample_{out}$ profit, and standard deviation per share with fees.

	Standard		**Competitive**		**Co-Operative**	
Share Code	**Mean Profit**	σ	**Mean Profit**	σ	**Mean Profit**	σ
AGL	233796.07	±8624.00	**238416.06**	**±0.00**	*116075.26*	*±116075.26*
ALSI	91128.61	±2851.79	**92640.05**	**±258.73**	*33711.21*	*±43035.70*
BIL	123908.93	±7832.92	**128804.51**	**±0.00**	*44087.70*	*±56477.87*
GFI	**-12889.49**	±13785.28	-13837.08	*±14357.16*	*-27415.77*	**±11547.11**
IMP	169605.59	±5426.77	**172412.54**	**±0.00**	*122435.69*	*±58605.91*
INV-NED	**54435.84**	±6730.08	46679.36	*±9018.08*	*250.53*	**±515.66**
INV-REM	6769.87	*±10045.23*	**10404.96**	**±4781.11**	*-2952.77*	±5511.83
INV-SBK	**20998.47**	**±2718.97**	14849.33	±8208.28	*-5591.64*	*±9319.41*
LON	196280.74	±13085.38	**197359.67**	**±10999.45**	*98341.61*	*±90639.88*
NED	-48931.84	±28239.59	*-54749.04*	*±32513.65*	**-1393.64**	**±4543.67**
RCH	**15436.26**	**±0.00**	**15436.26**	**±0.00**	*4286.16*	*±6690.06*
REM	67225.70	±960.79	**67722.66**	**±0.00**	*33692.59*	*±31862.22*
SAB	49231.25	6338.79	**53030.66**	**±0.00**	*17676.89*	*±23569.18*
SBK	6152.10	±131.01	**6219.87**	**±0.00**	*1800.57*	*3240.82*
SOL	98461.25	±55094.09	**142051.53**	**±0.00**	*46388.96*	*±61851.94*



(o) $Sample_{out}$ SOL

Figure 6. Critical difference plots for results with fees (Cont.).

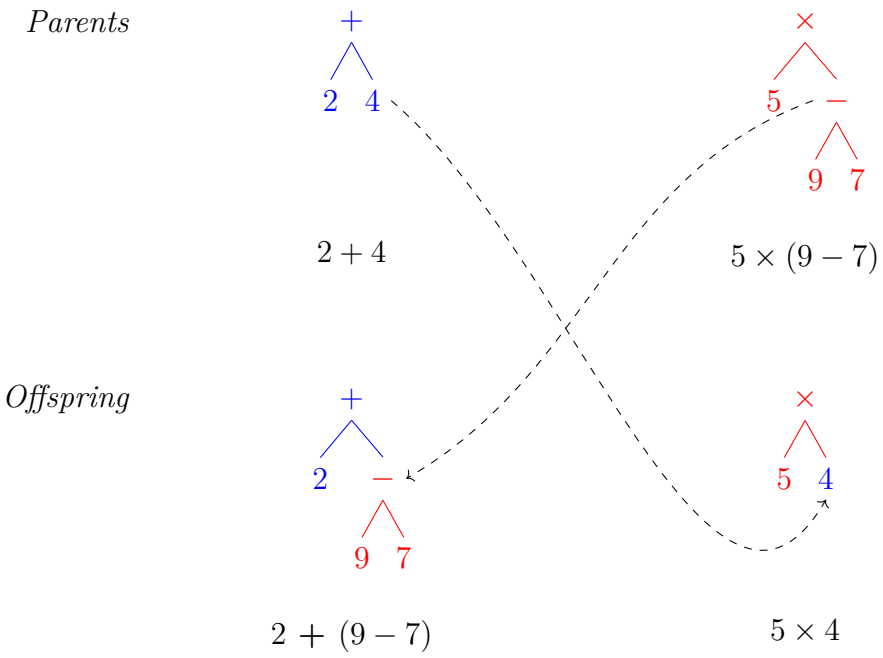
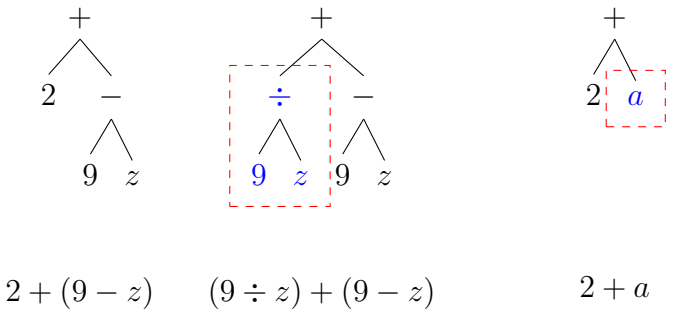


Figure 2. An example of GP crossover using two trees. Crossover is dependent on the grammar of the tree.



(a) Original *(b) Grow mutation* *(c) Prune mutation*

Figure 3. Examples of GP grow and prune mutation strategies.

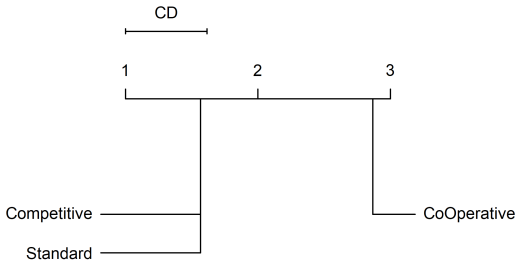
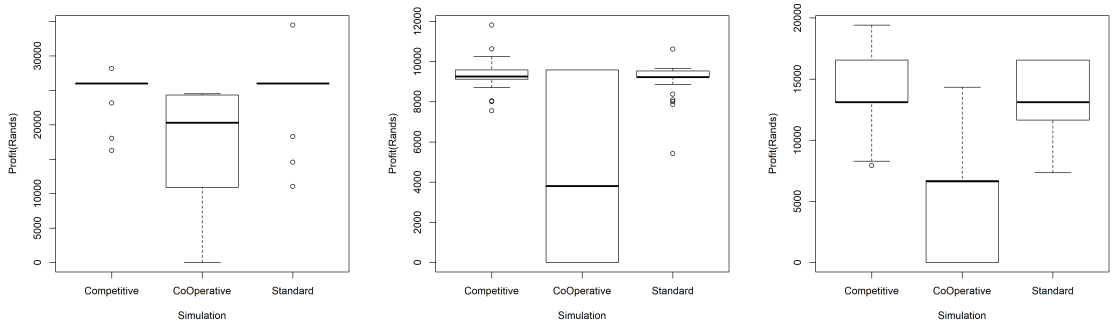
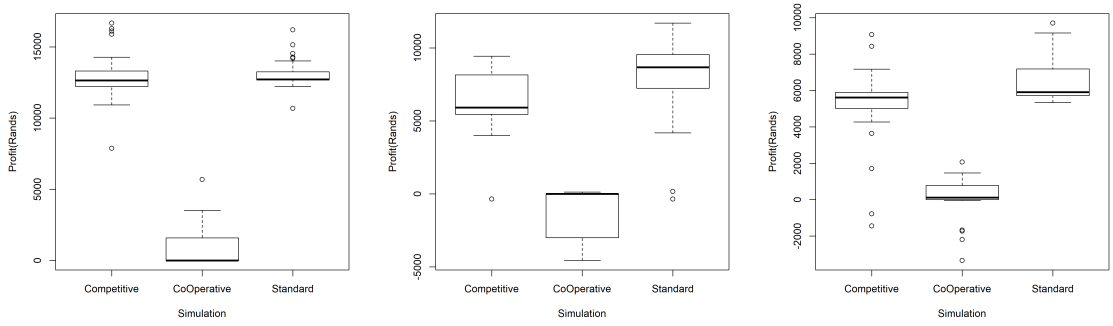


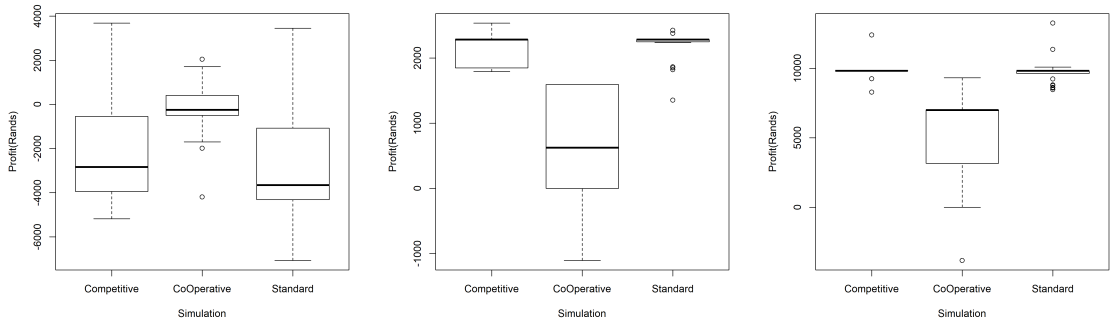
Figure 4. Critical difference plot illustration.



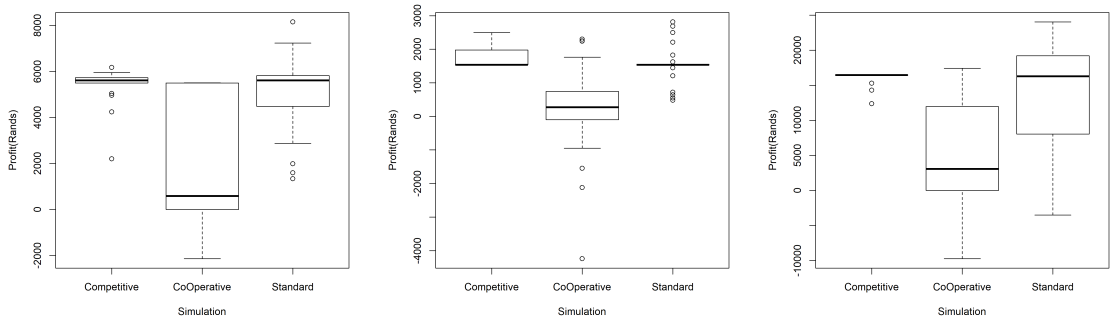
(a) $Sample_{out}$ AGL (b) $Sample_{out}$ ALSI (c) $Sample_{out}$ BIL



(d) $Sample_{out}$ INV-NED (e) $Sample_{out}$ INV-REM (f) $Sample_{out}$ INV-SBK

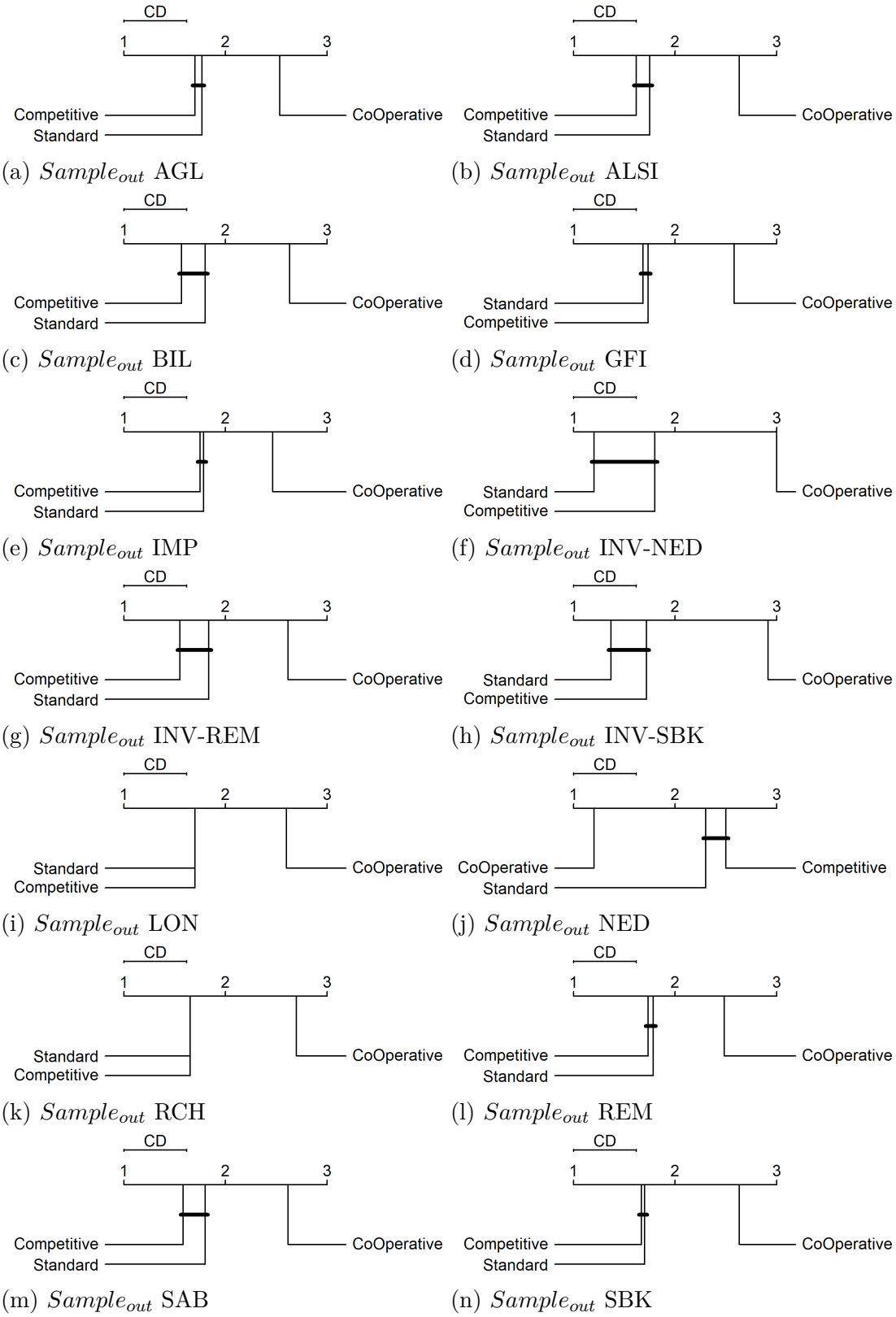


(g) $Sample_{out}$ NED (h) $Sample_{out}$ RCH (i) $Sample_{out}$ REM



(j) $Sample_{out}$ SAB (k) $Sample_{out}$ SBK (l) $Sample_{out}$ SOL

Figure 5. Box-plots of $Sample_{out}$ profit results without fees.



(a) $Sample_{out}$ AGL (b) $Sample_{out}$ ALSI

(c) $Sample_{out}$ BIL (d) $Sample_{out}$ GFI

(e) $Sample_{out}$ IMP (f) $Sample_{out}$ INV-NED

(g) $Sample_{out}$ INV-REM (h) $Sample_{out}$ INV-SBK

(i) $Sample_{out}$ LON (j) $Sample_{out}$ NED

(k) $Sample_{out}$ RCH (l) $Sample_{out}$ REM

(m) $Sample_{out}$ SAB (n) $Sample_{out}$ SBK

Figure 6. Critical difference plots for results with fees.

Algorithm 1 Implemented GP algorithm as presented by Allen and Karjalainen (Allen & Karjalainen, 1995).

$g = 0$; ▷ Generation counter
Set the maximum number of generations G;
Set the total number of individuals μ;
Initialise the initial population C_g;
Draw three continuous samples from the dataset
Initialise the in sample dataset $Sample_{in}$
Initialise the validation sample dataset $Sample_{sel}$
Evaluate fitness $f_{C_{g,n},Sample_{in}}$ of each individual $C_{g,n}$ in population C_g using $Sample_{in}$;
Set $B = C_{g,n}$ where $C_{g,n}$ has the largest $Sample_{in}$ fitness;
Evaluate fitness $f_{B,Sample_{sel}}$ of B using $Sample_{sel}$;
while $g < G$ **do**
  $g = g + 1$;
  Set number of offspring $\lambda = 0$;
  while $\lambda < \mu$ **do**
    Set random number $r \sim U(0, 1)$;
    if $r < P_m$ **then**
      ▷ Sampling is performed using a selection strategy
      Sample parent P_1 from C_{g-1} with bias to worst fitness;
      Mutate P_1 to form O_1;
      Evaluate the f of O_1 using $Sample_{in}$;
      Add O_1 to C_g;
      Sample D_1 from C_g with bias to worst $f_{C_g,Sample_{in}}$;
      Remove D_1 from C_g;
      $\lambda = \lambda + 1$;
    else
      ▷ Sampling is performed using a selection strategy
      Sample parents P_1 and P_2 from C_{g-1} with bias to best $f_{C_g,Sample_{in}}$;
      Offspring O_1 = Re-combination of P_1 and P_2;
      Offspring O_2 = Re-combination of P_2 and P_1;
      Evaluate the f of O_1 and O_2 using $Sample_{in}$;
      Add O_1 and O_2 to C_g;
      Sample D_1 and D_2 from C_g with bias to worst $f_{C_g,Sample_{in}}$;
      Remove D_1 and D_2 from C_g;
      $\lambda = \lambda + 2$;
    end if
  end while
  Set $B_g = C_{g,n}$ where $C_{g,n}$ has the largest $f_{C_g,Sample_{in}}$
  if $f_{B_g,Sample_{in}} > f_{B,Sample_{in}}$ **then**
    Evaluate the f of B_g using $Sample_{sel}$;
    if $f_{B_g,Sample_{sel}} > f_{B,Sample_{sel}}$ **then**
      $B = B_g$;
    end if
  end if
end while
return B as the solution;

Mutation

Crossover

Algorithm 2 Co-operative co-evolved GP algorithm.

$g = 0$; ▷ Generation counter
Set the maximum number of generations G;
Set the total number of individuals μ;
Initialise the in sample dataset $Sample_{in}$
Initialise the validation sample dataset $Sample_{sel}$
Initialise buy population $cBuy_g$; ▷ Randomly generate a population of trading rules
Initialise sell population $cSell_g$; ▷ Randomly generate a population of trading rules
Calculate the shared reward of each individual:
$rBuy_g$, $rSell_g$ = RewardSharing($cBuy_g$, $null$, $cSell_g$, $null$, $Sample_{in}$)
Set the best buy individual $bBuy = cBuy_{g,n}, n \in \{rBuy_{g,0}, \ldots, rBuy_{g,\frac{\mu}{2}}\}$, where $rBuy_{g,n}$ is the maximum shared reward;
Set the best sell individual $bSell = cSell_{g,n}, n \in \{rSell_{g,0}, \ldots, rSell_{g,\frac{\mu}{2}}\}$, where $rSell_{g,n}$ is the maximum shared reward;
while $g < G$ **do**
  $g = g + 1$;
  Evolve($cBuy_g$, $bBuy$, $cSell_g$, $bSell$, $Sample_{in}$, `'buy'`);
  Evolve($cBuy_g$, $bBuy$, $cSell_g$, $bSell$, $Sample_{in}$, `'sell'`);
  Calculate the shared reward of each individual using $Sample_{in}$:
    $rbBuy$, $rbSell$, $rBuy_g$, $rSell_g$ = RewardSharing($cBuy_g$, $bBuy$, $cSell_g$, $bSell$, $Sample_{in}$)
  Re-calculate the temporary shared reward of each individual using $Sample_{sel}$:
    $rbBuyT$, $rbSellT$, $rBuyT_g$, $rSellT_g$ = RewardSharing($cBuy_g$, $bBuy$, $cSell_g$, $bSell$, $Sample_{sel}$)
  Set the best buy individual index $x = rBuy_{g,n}, n \in \{rBuy_{g,0}, \ldots, rBuy_{\frac{\mu}{g,2}}\}$, where $rBuy_{g,n}$ is the maximum shared reward;
  if $rbBuy > rBuy_{g,x}$ **and** $rbBuyT > rBuyT_x$ **then**
    $bBuy = cBuy_{g,x}$; ▷ New global best buy rule
  end if
  Set the best sell individual index $y = rSell_{g,n}, n \in \{rSell_{g,0}, \ldots, rSell_{g,\frac{\mu}{2}}\}$, where $rSell_{g,n}$ is the maximum shared reward;
  if $rbSell > rSell_{g,y}$ **and** $rbSellT > rSellT_{g,y}$ **then**
    $bSell = cSell_{g,y}$; ▷ New global best sell rule
  end if
end while
return $bBuy, bSell$ as the solution; ▷ Returns both parts of the trading rule

Algorithm 3 Co-operative co-evolved GP evolutionary process.

procedure EVOLVE($cBuy_g$, $bBuy$, $cSell_g$, $bSell$, $Sample_{in}$, `rule`)
  Set number of offspring $\lambda = 0$;
  while $\lambda < (\mu \div 2)$ **do** ▷ $\mu \div 2$ because there are 2 populations
    Calculate the shared reward of each individual using $Sample_{in}$:
      $rbBuy$, $rbSell$, $rBuy_g$, $rSell_g$ = REWARDSHARING($cBuy_g$, $bBuy$, $cSell_g$, $bSell$, $Sample_{in}$)
    Set evolution population C_g and reward share R_g:
    if `rule` = '`buy`' **then**
      $C_g = cBuy_g$;
      $R_g = rBuy_g$;
    else
      $C_g = rSell_g$;
      $R_g = rSell_g$;
    end if
    Set random number $r \sim U(0, 1)$;
    if $r < P_m$ **then** ▷ P_m is the mutation probability
      ▷ Sampling is performed using rank selection
      Sample parent P_1 from C_{g-1} with bias to the worst R_{g-1};
      Mutate P_2 to form O_1;
      Sample D_1 from C_g with bias to the worst shared reward R_g;
      Remove D_1 from C_g;
      Add O_1 to C_g;
      $\lambda = \lambda + 1$;
    else } Mutation
      ▷ Sampling is performed using rank selection
      Sample parents P_1 and P_2 from C_{g-1} with bias to the worst R_{g-1};
      Offspring O_1 = Re-combination of P_1 and P_2;
      Offspring O_2 = Re-combination of P_2 and P_1;
      Add O_1 and O_2 to C_g;
      Sample D_1 and D_2 from C_g with bias to the worst R_g;
      Remove D_1 and D_2 from C_g;
      $\lambda = \lambda + 2$;
    end if } Crossover
  end while
end procedure

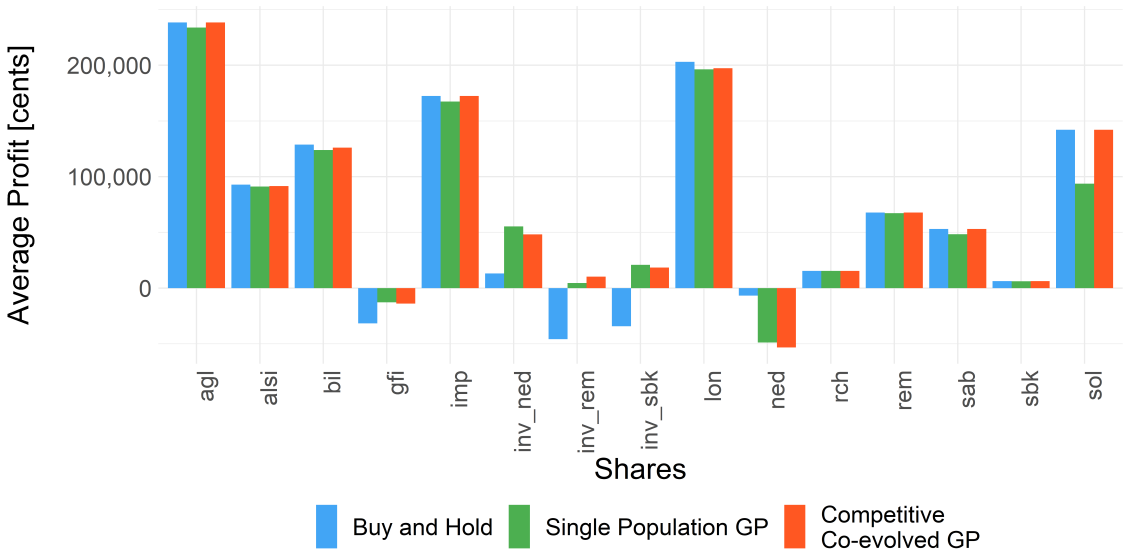


Figure 7. Comparison of profit returned by a buy and hold strategy to the profit returned by evolved traders.

Algorithm 4 Co-operative co-evolution reward sharing algorithm.

function RewardSharing($cBuy$, $bBuy$, $cSell$, $bSell$, $Sample$)
  Initialise the total reward $rBuy$ of each buy individual in the population $cBuy$ to 0;
  Initialise the total reward $rSell$ of each sell individual in the population $cSell$ to 0;
  for $x = 0; x < (\mu \div 2); x = x + 1$; **do** ▷ 2, Because there are 2 populations
    for $y = 0; y < (\mu \div 2); y = y + 1$; **do**
      Calculate the fitness f = EvaluateCombination($cBuy_x$, $cSell_y$, $Sample$);
      Update the total reward of the individuals used in the evaluation:
        $rBuy_x = rBuy_x + f$;
        $rSell_y = rSell_y + f$;
    end for
  end for
  if $bBuy \neq null$ **and** $bSell \neq null$ **then**
    Initialise total reward $rbBuy$ of the best buy individual $bBuy$ to 0;
    Initialise total reward $rbSell$ of the best sell individual $bSell$ to 0;
    for $y = 0; y < (\mu \div 2); y = y + 1$; **do**
      Calculate the fitness f = EvaluateCombination($bBuy$, $cSell_y$, $Sample$);
      Update the total reward of the individuals used in the evaluation:
        $rbBuy = rbBuy + f$;
        $rSell_y = rSell_y + f$;
    end for
    for $x = 0; x < (\mu \div 2); x = x + 1$; **do**
      Calculate the fitness f = EvaluateCombination($bSell$, $cBuy_x$, $Sample$);
      Update the total reward of the individuals used in the evaluation:
        $rbSell = rbSell + f$;
        $rBuy_x = rBuy_x + f$;
    end for
  end if
  return $rbBuy$, $rbSell$, $rBuy$, $rSell$; ▷ Return the shared rewards
end function

Algorithm 5 Co-operative co-evolved trading rule evaluation.

```
function EvaluateCombination(BuyRule, SellRule, Sample)
    Set boolean s = false;                    ▷ The trader has no shares on the first day
    Set fitness f = 0;
    for i = 0; i < Length(Sample); i = i + 1; do
        if s == false then
            Set buy action a = BuyRule(Sample_i);      ▷ Determine the action using the BuyRule
            if a == true then
                Buy shares;
                s = true;
            end if
            Update fitness value f using fitness function and action;
        else
            Set sell action a = SellRule(Sample_i);    ▷ Determine the action using the SellRule
            if a == true then
                Sell shares;
                s = false;
            end if
            Update fitness value f using fitness function and action;
        end if
    end for
    if s == true then
        Sell shares;
        Update fitness value f using fitness function and action;
    end if
    return f;          ▷ Return the fitness value of the BuyRule, SellRule combination
end function
```

Algorithm 6 Competitive co-evolved GP algorithm.

$g = 0$; ▷ Generation counter
Set the maximum number of generations G;
Set the maximum number of populations P;
Set the total number of individuals μ;
Initialise the in sample dataset $Sample_{in}$
Initialise the validation sample dataset $Sample_{sel}$
for $p = 0; p < P; p = p + 1$; **do**
    Initialise initial generation's population $C_{p,g}$; ▷ Randomly generate a population of trading rules
end for
for $p = 0; p < P; p = p + 1$; **do**
    Calculate relative fitness rf for each individual in $C_{p,g}$
        $rf_{p,g}$ = CalcAllRelativeFitness(C_g, p);
    Set the population's global best individual for this population $pB_p = C_{p,g,n}$, $n \in C_{p,g}$, where
        pB_p produces the maximum relative fitness rf_p;
end for
while $g < G$ **do**
    $g = g + 1$;
    for $p = 0; p < P; p = p + 1$; **do**
        Evolve(C, p, g);
        Calculate relative fitness rf for each individual in $C_{p,g}$
            $rf_{p,g}$ = CalcAllRelativeFitness(C_g, p);
        Set the best individual index x, for this population and generation:
            $x = rf_{p,g,n}$, $n \in \{rf_{p,g,0}, \ldots, rf_{p,g,\frac{\mu}{P}}\}$, where
            $rf_{p,g,n}$ is the maximum relative fitness value;
        Re-calculate the population's global best individual's relative fitness using $Sample_{in}$:
            rfB = CalcRelativeFitness(pB_p, $Sample_{in}$, C_g, p);
        if $rfB > rf_{p,g,x}$ **then**
            Calculate temporary relative fitness using the validation sample $Sample_{sel}$:
                rfI = CalcRelativeFitness($C_{p,g,x}$, $Sample_{sel}$, C_g, p);
                rfB = CalcRelativeFitness(pB_p, $Sample_{sel}$, C_g, p);
            if $rfI > rfB$ **then**
                Set $pB_p = C_{p,g,x}$; ▷ New global population best
            end if
        end if
    end for
end while
return B_x, $x \in \{pB_0, \ldots, pB_P\}$, where ▷ Return the best individual across all the populations
    B_x produces the maximum relative fitness for both $Sample_{in}$ and $Sample_{sel}$;

Algorithm 7 Competitive co-evolved GP evolutionary process.

▷ Evolve receives all populations as C

procedure EVOLVE(C, p, g)
  Set number of offspring $\lambda = 0$;
  while $\lambda < (\mu \div P)$ **do** ▷ $\mu \div P$ because there are P populations
    Set random number $r \sim U(0, 1)$;
    if $r < P_m$ **then** ▷ P_m is the mutation probability
      ▷ Sampling is performed using rank selection
      Sample parent P_1 from $C_{p,g-1}$ with bias to worst $rf_{p,g-1}$;
      Mutate P_2 to form O_1;
      Sample D_1 from $C_{p,g}$ with bias to worst $rf_{p,g}$;
      Remove D_1 from $C_{p,g}$;
      Add O_1 to $C_{p,g}$ as $C_{p,g,n+1}$, where n is the size of $C_{p,g}$;
      Set relative fitness:
        $rf_{p,g,n+1}$ = CALCRELATIVEFITNESS($C_{p,g,n+1}$, $Sample_{in}$, C_g, p);
      $\lambda = \lambda + 1$;
    else
      ▷ Sampling is performed using rank selection
      Sample parents P_1 and P_2 from $C_{p,g-1}$, with bias to best $rf_{p,g-1}$;
      Offspring O_1 = Re-combination of P_1 and P_2;
      Offspring O_2 = Re-combination of P_2 and P_1;
      Sample D_1 and D_2 from $C_{p,g}$ with bias to worst $rf_{p,g}$;
      Remove D_1 and D_2 from $C_{p,g}$;
      Add O_1 and O_2 to $C_{p,g}$ as $C_{p,g,n+1}$ and $C_{p,g,n+2}$, where n is the size of $C_{p,g}$;
      Set relative fitness:
        $rf_{p,g,n+1}$ = CALCRELATIVEFITNESS($C_{p,g,n+1}$, $Sample_{in}$, C_g, p);
      Set relative fitness:
        $rf_{p,g,n+2}$ = CALCRELATIVEFITNESS($C_{p,g,n+2}$, $Sample_{in}$, C_g, p);
      $\lambda = \lambda + 2$;
    end if
  end while
end procedure

Mutation

Crossover

Algorithm 8 Competitive co-evolved GP relative fitness algorithms.

▷ C denotes all the populations, and generations
function CalcAllRelativeFitness(C_g, p)
  for $n = 0; n < (\mu \div P); n = n + 1$; **do** ▷ For each individual within the population
    Set the relative fitness rf of the individual n in population p, and generation g:
      $rf_{p,n}$ = CalcRelativeFitness($C_{p,g,n}$, $Sample_{in}$, p);
  end for
  return $rf_{p,g}$ ▷ Return the fitness values for all individuals within the population and generation
end function
function CalcRelativeFitness(*Individual*, *Sample*, C_g, p)
  Set relative fitness $rf = 0$;
  Calculate the *Individual*'s fitness value f using a fitness function and the *Sample* dataset;
  for $i = 0; i < P; i = i + 1$; **do**
    if $i \neq p$ **then** ▷ Compare the *Individual* against individuals from other populations
      for $n = 0; n < (\mu \div P); n = n + 1$; **do** ▷ For each individual within the population
        Calculate the temporary *Sample* fitness value t of $C_{p,g,n}$,
          using a fitness function and the *Sample* dataset;
        Individual gets a point each time its fitness f
          is greater than an individual in another population
        if $f > t$ **then**
          $rf = rf + 1$;
        end if
      end for
    end if
  end for
  return fR; ▷ Return the individual's relative fitness for the sample provided
end function

References

Achelis, S. B. (2013). *Technical analysis from A to Z* (Second). McGraw-Hill Education. Retrieved from https://www.amazon.com/Technical-Analysis-2nd-Steven-Achelis/dp/0071826297

Allen, F., & Karjalainen, R. (1995). Using genetic algorithms to find technical trading rules. *Rodney L. White Center for Financial Research*, 20–95. Retrieved from https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6996

Allen, F., & Karjalainen, R. (1999). Using genetic algorithms to find technical trading rules1. *Journal of financial Economics*, *51*(2), 245–271. doi:https://doi.org/10.1016/S0304-405X(98)00052-X

Angeline, P. J., & Pollack, J. B. (1993). Competitive environments evolve better solutions for complex tasks, 264–270.

Axelrod, R. M. (2006). *The evolution of co-operation* (Revised). Basic books, member of the Perseus Books Group. Retrieved from https://www.amazon.com/Evolution-Cooperation-Revised-Robert-Axelrod/dp/0465005640

Bäck, T. (1996). *Evolutionary algorithms in theory and practice: Evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press.

Bäck, T., Fogel, D. B., & Michalewicz, Z. (1997). *Handbook of evolutionary computation* (First). IOP Publishing Ltd.

Bäck, T., & Schwefel, H.-.-.-P. (1996). Evolutionary computation: An overview, 20–29. doi:10.1109/ICEC.1996.542329

Bauer, R. J. (1994). *Genetic algorithms and investment strategies*. John Wiley & Sons.

Bishop, G. W. J. (1961). Charles H. Dow and the Dow Theory. *The economic history review*, *13*(3), 520–521.

Bluman, A. G. (1995). *Elementary statistics*. Brown Melbourne.

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

Casas, N. (2015). A review of landmark articles in the field of co-evolutionary computing. *arXiv*, 1–5. Retrieved from https://arxiv.org/abs/1506.05082

Cowles, A. (1933). Can stock market forecasters forecast. *Econometrica*, *1*(3), 309–324.

De Lima, E. B., Pappa, G. L., de Almeida, J. M., Gonçalves, M. A., & Meira, W. (2010). Congress on evolutionary computation, tuning genetic programming parameters with factorial designs, 1–8. doi:10.1109/CEC.2010.5586084

Demšar, J. (2006). Statistical comparisons of classifiers over multiple data sets. *Journal of machine learning research*, *7*, 1–30.

Edwards, R. D., Magee, J., & Bassetti, W. H. C. (2007). *Technical analysis of stock trends* (Ninth). Taylor & Francis. Retrieved from http://books.google.co.za/books?id=wklriRw9a1oC

Engelbrecht, A. P. (2007). *Computational intelligence: An introduction*. Wiley. Retrieved from https://books.google.co.za/books?id=IZosIcgJMjUC

Fama, E. F. (1965). The behaviour of stock market prices. *The journal of business*, *38*(1), 34–105.

Fogel, D. B. (2006a). *Evolutionary computation: Toward a new philosophy of machine intelligence*. IEEE Press Series on Computational Intelligence. Wiley. Retrieved from https://books.google.co.za/books?id=1SQuadczM9oC

Fogel, D. B. (2006b). Foundations of evolutionary computing. *Proceedings of SPIE, the international society for optical engineering*, 1–13. doi:10.1117/12.669679

Friedman, G. J. (1956). *Selective feedback computers for engineering synthesis and nervous system analogy*. Master's thesis, UCLA. Retrieved from https://books.google.co.za/books?id=Qy-2NwAACAAJ

García, S., Fernández, A., Luengo, J., & Herrera, F. (2008). A study of statistical techniques and performance measures for genetics-based machine learning: Accuracy and interpretability. *Computational Intelligence and Complexity*, *13*(10), 959. doi:https://doi.org/10.1007/s00500-008-0392-y

Graczyk, M., Lasota, T., Telec, Z., & Trawiński, B. (2010). Non-parametric statistical analysis of machine learning algorithms for regression problems. Lecture Notes in Computer Science, *6276*, 111–120. Retrieved from https://link.springer.com/chapter/10.1007/978-3-642-15387-7_15

Hillis, W. D. (1990). Co-evolving parasites improve simulated evolution as an optimisation procedure. *Physica D: Nonlinear Phenomena*, *42*(1-3), 228–234.

Holland, J. H. [John H.]. (1992). *Adaptation in natural and artificial systems: An introductory analysis with applications to biology, control, and artificial intelligence*. MIT press.

Holland, J. H. [John H.]. (2000). *Emergence: From chaos to order*. OUP Oxford.

Holland, J. H. [John Henry]. (1995). *Hidden order: How adaptation builds complexity*. Basic Books.

Iman, R. L., & Davenport, J. M. (1980). Approximations of the critical region of the fbietkan statistic. *Communications in statistics: Theory and methods*, *9*(6), 571–595.

James, F. E. (1968). Monthly moving averages an effective investment tool. *Journal of financial and quantitative analysis*, *3*(3), 315–326.

John, H. (1985). Holland. Properties of the bucket brigade, 1–7.

Jones, T. et al. (1995). Crossover, macromutation, and population-based search, 73–80.

King, W. I. (1934). Technical methods of forecasting stock prices. *Journal of the American statistical association*, *29*(187), 323–325.

Koza, J. R. (1992a). *Genetic programming II, automatic discovery of reusable subprograms*. MIT Press, Cambridge, MA.

Koza, J. R. (1992b). *Genetic programming: On the programming of computers by means of natural selection*. MIT press.

Koza, J. R., & Poli, R. (2005). Genetic programming. In *Search methodologies* (pp. 127–164). Springer.

Mahfoud, S., & Mani, G. (1996). Financial forecasting using genetic algorithms. *Applied artificial intelligence*, *10*(6), 543–566.

Mani, G., Quah, K.-.-.-K., Mahfoud, S., & Barr, D. (1995). An analysis of neural-network forecasts from a large-scale, real-world stock selection system, 72–78. doi:10.1109/CIFER.1995.495254

Neely, C., Weller, P., & Dittmar, R. (1997). Is technical analysis in the foreign exchange market profitable? A genetic programming approach. *Journal of financial and Quantitative Analysis*, *32*(4), 405–426.

Nicholls, J. F. (2018). *Co-evolved genetic program for stock market trading* (Master's thesis, University of Pretoria).

Peterson, N. (2007). *Wall street lingo: Thousands of investment terms explained simply*. Atlantic Publishing Group. Retrieved from https://books.google.co.za/books?id=3Y9YAAAAYAAJ

Potvina, J., Sorianoa, P., & Vall, M. (2004). Generating trading rules on the stock markets with genetic programming. *Computers & Operations Research*, *31*(7), 1033–1047. doi:10.1016/S0305-0548(03)00063-7

Rhea, R. (1993). *The Dow Theory: An explanation of its development and an attempt to define its usefulness as an aid in speculation*. Fraser Publishing Company.

Rosin, C. D., & Belew, R. K. (1997). New methods for competitive co-evolution. *Evolutionary computation*, *5*(1), 1–29.

Salzberg, S. L. (1997). On comparing classifiers: Pitfalls to avoid and a recommended approach. *Data mining and knowledge discovery*, *1*(3), 317–328.

Seshadri, M. (2003). *Comprehensibility, overfitting and co-evolution in genetic programming for technical trading rules* (Doctoral dissertation, Worcester Polytechnic Institute).

Shahsavar, M., Najafi, A. A., & Niaki, S. T. A. (2011). Statistical design of genetic algorithms for combinatorial optimization problems. *Mathematical Problems in Engineering*, *2011*(872415), 17. doi:10.1155/2011/872415

Sheskin, D. J. (2003). *Handbook of parametric and non-parametric statistical procedures*. crc Press.

El-Telbany, M. E. (2004). The Egyptian stock market return prediction: A genetic programming approach, 161–164. doi:10.1109/ICEEC.2004.1374410

Tilkin, G. (2001). MACD divergences. *Active trader*, 2–4. Retrieved from http://www.activetradermag.com/

Tsang, E. (2009). Forecasting where computational intelligence meets the stock market. *Frontiers of computer science in China*, *3*(1), 53–63. doi:10.1007/s11704-009-0012-8

Tsang, E. P. K., Li, J., & Butler, J. M. (1998). EDDIE beats the bookies. *Softw., Pract. Exper. 28*(10), 1033–1043.