Universiteit Leiden
Opleiding Informatica

Evolving Technical Trading Strategies

using Genetic Programming

Name: Thijs Vermeulen

Date: 27/08/2014

1st supervisor: Prof. Dr. Thomas Bäck
2nd supervisor: Dr. Michael Emmerich

MASTER'S THESIS

Leiden Institute of Advanced Computer Science (LIACS)
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

In this thesis, we apply Genetic Programming to evolve technical trading strategies for the stock market, based on a collection of 242 constituent stocks of the S&P 500 index. Previous research into evolving technical trading strategies has focused on market timing, where a strategy trades on a single security aiming to profit from price fluctuations. We apply this same approach, and compare it to a stock picking system, where the trading strategy maintains a portfolio of investments in multiple stocks.

We find that for market timing, the trading strategies are not able to consistently beat the buy-and-hold benchmark, even when transaction costs are ignored. We do find that the evolved strategies have at least some predictive value, as they are able to outperform a random trading strategy.

For stock picking, the results are more promising, and when transaction costs are ignored the evolved strategies outperform buy-and-hold by a large margin. These strategies appear to be largely based on a short-term contrarian investment style. Adding a reasonable level of transaction costs forces the strategies to make more long-term investments, and unfortunately the excess returns completely disappear.

We can conclude that stock prices do not seem to be entirely unpredictable, but the Efficient Market Hypothesis is not challenged by this research.

Contents

1 Introduction

Forecasting of the financial markets has been the subject of research for many years, and different forecasting methods have been developed by both academics and practitioners. The motivation for attempting to forecast the markets is clear, as consistent success potentially leads to enormous economic profits. However, the *Efficient Market Hypothesis* (EMH) [8][9] states that the financial markets are essentially unpredictable, and that as a result, consistently profiting from price forecasting is in fact impossible. Although generally accepted, the validity of the EMH is often questioned, and investors continue their practice of forecasting on a large scale.

One widely used method of forecasting is *technical analysis*, which involves searching for recurring patterns in the prices of securities. *Genetic Programming* (GP), an evolutionary computation technique developed by Koza [10], can be used to automate the process of technical analysis by evolving trading strategies. This thesis describes an investigation into the potential profitability of this approach. As such, the research can be seen as a test of the EMH.

The rest of this introduction describes technical analysis in more detail, followed by a closer look at the EMH and a brief introduction to mechanical trading and genetic programming. Chapter 2 describes the methodology used in this research. Chapter 3 reports the results of the experiments, followed by the conclusion in chapter 4. For a literature review on related work the reader is referred to "A literature review on evolving trading rules using genetic programming" by the author of this thesis [18].

1.1 Technical Analysis

Methods of analyzing the stock markets with the goal of forecasting fall into two categories, *technical analysis* and *fundamental analysis*. The difference between the two lies in the different sets of information that are studied. Technical analysis is based solely on the price history of a stock and the history of its trading volume, where fundamental analysis is the study of any other information that could be of interest. This can include for example the financial details of a company, the current economic situation, or information about the management of a company or a company's competitors. The two methods of analysis are quite distinct, although it is also possible to combine the two in order to use all available information and make the best possible investment decisions.

This research focuses purely on technical analysis. Technical analysis is typically based on visual inspection of charts that show the recent price

history of securities. The idea is to recognize patterns or situations occurring in the charts that present a trading opportunity, where an increase or decrease in price can be expected in the near future.

There is a wide variety of different known patterns that are believed to have such predictive qualities. Although these patterns vary strongly in type and complexity, very often the motivation for a prediction can be characterized as trend following. A trend is a continuously rising price (*uptrend*), or a continuously falling price (*downtrend*). Trend following generally means to buy a stock when it is perceived to be in an uptrend, or sell a stock when it is perceived to be in a downtrend. These actions are based on the assumption that the trend is likely to continue for some time.

Completely opposed to trend following is the concept of *mean reversion*, also known as *counter-trend* or *contrarian* investing. Following this concept, in the case of recent increases in price, the assumption is made that the stock is now likely to be priced above its actual value. It is then more reasonable to expect a decrease in price (return to the mean), rather than to assume that the security is in an uptrend that will continue into the future. These two opposing interpretations of a similar situation illustrate one of the main challenges for the technical analyst, which comes down to recognizing whether a trend is likely to continue, or more likely to reverse.

Security trading strategies and security traders can be separated based on the duration of a typical investment. *Intraday trading*, or *day trading*, involves trades that are opened and closed within the same day. The investment durations range from several hours to only minutes. *Interday trading* involves trades that last until at least the next day, and often several days or longer. Generally, for shorter term trades, a shorter time frame of past data is analyzed, but with a higher sampling rate.

1.1.1 Technical Indicators

Historically, technical analysis was constrained to manual inspection of the price charts. With advances in computer power more computational and statistical methods have been developed to aid the process. One widely used tool is a collection of *technical indicators*. Technical indicators are mathematical functions of the price history and/or the trading volume history. These functions are commonly overlaid on the price charts, and are aimed at giving the analyst a better insight into price developments. There are many different widely known technical indicators that have been developed and used over the years. The chart in Figure 1 shows two examples of commonly used technical indicators, and we illustrate their use in the rest of this

subsection.

Figure 1: Price history for the S&P 500 index (ticker symbol ˆGSPC), including the indicators Simple Moving Average (SMA) and Relative Strength Index (RSI) (source: Yahoo! Finance [1])

The *Simple Moving Average* (SMA) indicator is simply defined as the average price over the most recent history. As is the case for all technical indicators, it can be calculated using different time windows. The two moving averages in the chart are calculated for windows of 10 days and 50 days. The main use of the moving average is to identify the direction of the general price trend, by smoothing out shorter term price variations, or noise. The size of the window has a strong effect on the shape of the SMA, as can be seen in the chart. The SMA with window size 10 follows the price more closely than the SMA with window size 50, identifying shorter term price trends.

Besides the simple identification of the direction of the trend, a common method in which the SMA is used is to analyze two SMA indicators of different time windows together as a pair, and look for crossover points between

the two. For example, a crossover point where the shorter term SMA rises above the longer term SMA, can indicate that the price has recently been rising more than it has been on a larger time-frame. This could be interpreted as a sign of an uptrend currently in progress. In the example in Figure 1, this happens for example at the start of July, where the SMA(10) crosses through the SMA(50), after some strong price increases. In hindsight this did not seem to be the start of a long-lasting uptrend, as the price did not rise further in the following period. The crossover point at the beginning of August, where the SMA(10) crosses the SMA(50) in the opposite direction, was in fact followed by a continuation of the trend.

The *Relative Strength Index* (RSI) is a more derived indicator. Its calculation is based on the ratio between recent *updays* and recent *downdays*, which are defined as days during which the security has risen in price or fallen in price, respectively. The RSI indicator is an example of what is called an oscillator, as instead of following the price, its value moves within a fixed range, between 0 and 100. Subsection 2.2 describes the exact calculation of RSI. Oscillators such as RSI can not be overlain on the price chart, but their values can be directly interpreted. The value of RSI is generally high (e.g. > 70) when the price has recently been increasing, and low (e.g. < 30) when the price has recently been falling. The version shown in the chart is calculated for a time window of 14 days, which is a commonly used time window for RSI.

A typical interpretation of RSI is that a high value indicates an overbought situation, where the price is higher than the real value of the security, and can be expected to fall. The opposite can be said for low values of RSI. RSI reached such high values twice during the period shown in the example, at the start of May and the start of July, and in both cases the prices decreased significantly in the ensuing periods. Following the same principle, the very low value of RSI during the first half of August would be interpreted as a buy signal, expecting that the price will shoot back up. We can not tell from the chart if it would have been the right decision, as the downtrend could have very well continued into the future.

1.2 The Efficient Market Hypothesis

The *Efficient Market Hypothesis* (EMH) is first described by Fama [8]. It states that financial markets are efficient, where an efficient market is defined as a market in which the price of a security represents its true underlying value at all times. The theory is based on the assumptions that all information that influences the markets is widely and directly available to investors, and that the body of investors as a whole uses the information to invest in

stocks rationally, and responds to new information quickly. Under these assumptions, security prices are always an accurate reflection of the available information. This renders them unpredictable, since future price movements can only be caused by newly available information.

Jensen [9] gives a more refined definition of essentially the same theory:

Definition 1.1. A market is efficient with respect to information set θ_t, if it is impossible to make economic profits by trading on the basis of information set θ_t, where economic profits are the risk adjusted returns net of all costs.

This is a less strict definition. For it to hold, prices do not need to be an entirely accurate reflection of the available information, but good approximations where mispricing is allowed within the range of the trading costs. Also, excess profits are in fact possible, but only as a premium for taking on more investment risk. Following from this definition, both realistic trading costs and investment risk need to be taken into account before the EMH can be challenged.

The EMH has been described in three forms, *weak*, *semi-strong*, and *strong*. These three forms refer to different degrees of efficiency, or more specifically, different sets of information that are reflected in the price.

Weak-form EMH states that at least all past price data is reflected in the current price. As a result, technical analysis can not be used to predict prices.

Semi-strong-form EMH states that all public knowledge is reflected in the price. This results in the impossibility of earning excess returns using technical or fundamental analysis, as both are based on analysis of public knowledge. The semi-strong-form EMH is what is usually meant by unspecified mentions of the EMH.

Strong-form EMH states that also all privately available information is reflected in the price. Prices that reflect all private information are only possible with frequent occurrence of insider trading, so strong-form efficiency is only possible if there are no laws against insider trading or the existing laws are frequently broken. It is generally assumed that the strong-form EMH does not hold.

The research presented in this thesis can be seen as a test of the weak-form EMH (and thus all forms) since an attempt is made to achieve excess returns using only technical analysis.

1.3 Mechanical Trading and Trading Rules

Analyzing the price charts and trading can be done in a discretionary fashion, where an analyst relies on experience or intuition to make arbitrary trading decisions. Mechanical (or systematic) trading is a more methodical approach, which involves defining a trading strategy that specifies exactly when to buy and sell, and applying this strategy consistently. This has important advantages over discretionary trading, as a fully specified trading strategy can be back-tested against historical data, and human error of judgment can be avoided while trading. Also, computational methods can be used to optimize or even create these trading strategies.

In order to perform mechanical trading, some method of defining a trading strategy is needed. One method is to use trading rules in the form of boolean expressions, typically represented by a tree structure. Trading rules in this form are suitable for the Genetic Programming technique. In the case of technical analysis, the trading rules typically combine signals from various technical indicators, the current price, and constants. The trading rule can be applied to a stock's price at any point in time, generating a boolean value for that point in time, which can be used as a trading signal.

To define a strategy using these trading rules, the boolean signal that a rule produces for each day needs to be mapped to trading decisions. The simplest example is a trading strategy that trades on a single stock, and goes long in the market (owning the stock) on the days the rule evaluates to true, and out of the market (not owning the stock) on the days the rule evaluates to false.

1.4 Genetic Programming

Genetic Programming (GP) is a specific form of evolutionary algorithm, developed by Koza [10]. Evolutionary algorithms mimic biological evolutionary processes in order to find solutions to optimization and search problems. Specifically, the evolutionary processes of natural selection, gene mutation, and sexual reproduction are mimicked.

GP is specialized in that it searches for solutions in the form of computer programs that perform well at a certain predefined task. The GP algorithm starts by generating a number of completely random programs. A selection of these programs is then made based on how well they perform at the task (analogous to natural selection). These selected programs undergo random individual alterations (gene mutation) and they are recombined (sexual reproduction) in order to create a new generation of programs. This process of selection, mutation and recombination is repeated several times, with the

goal of finding increasingly well-performing programs.

GP is originally designed to evolve programs in the form of tree structures. For this reason, functional programming languages (e.g. LISP [17]) are most suitable for GP, as these languages allow programs to be expressed by tree structures most naturally. Later, different GP variations have been developed which evolve programs in a different structure. An example of this is *Linear Genetic Programming* (LGP) [4], which evolves computer programs in the form of sequences of imperative instructions (typically assembly code).

Since technical trading rules are essentially mathematical expressions, they can be represented naturally by tree structures. Because of this, the original tree-based GP approach is a suitable technique for evolving them. Subsection 2.4 describes the GP implementation used in this research in detail.

2 Methodology

The research in this thesis is based on interday trading in the stock markets, for a collection of 242 different stocks. Our approach is partially derived from the work by Allen and Karjalainen [2] and Becker and Seshadri [3], but we make some adjustments. This chapter describes the adjusted approach in detail. We start by describing the used dataset, the set of technical indicators we use as input signals, and the structure of the trading rules. This is followed by a specification of the GP implementation, and the setup of the experiments.

2.1 Data

The dataset we use for the experiments consists of daily stock price data of a selection of companies included in the S&P 500 index. The S&P 500 is a stock market index based on 500 of the largest companies listed on the American *NYSE* and *NASDAQ* exchanges. We use data gathered from the *Yahoo! Finance* website [1], which offers daily price data available for download that includes a closing price corrected for stock splits and dividend payouts. We base both the calculation of the technical indicators and the trading simulations on these corrected closing prices. Data is used from the period 1993–2013, where the years 1993–1999 are used solely for training purposes, and multiple out-of-sample trading tests are performed on the period 2000–2013.

Stocks are included in the dataset when they meet the following criteria:

- The stock was one of the S&P 500 constituents on the date of 1 January 2000
- The stock's available price history dates back to at least 1 January 1993
- The stock's price history was available for download from Yahoo! Finance [1] at the start of 2014, which is the case if it is still enlisted (still exists) at that point.

These selection criteria result in a total of 242 stocks, all with an available price history that fully includes the years 1993–2013. For each year, there are slightly more than 250 dates for which the price is reported, as the exchanges are closed during weekends and holidays.

2.2 Technical Indicators

Previous research on evolving technical trading rules has varied in the set of possible input signals available to the trading rules. Early work (Allen and Karjalainen [2], Neely [14]) used relatively simple inputs such as a *simple moving average*, *lag* (price level at a given number of days ago), and *min/max* (the minimum or maximum price over the past days). Later, Becker and Seshadri [3], Potvin [16], and Lohpetch and Corne [11][12][13] extended these sets, by adding more derived technical indicators such as *Relative Strength Index* and *Rate of Change* (both defined later in this subsection). Preliminary testing of our GP implementation showed that the choice of allowed input signals has a significant effect on the performance of the evolved rules. We found that performance improved when we use only derived technical indicators and exclude the simpler inputs such as lag and min/max. Also, we tested a variety of technical indicators in several combinations, and concluded that the combination of four particular technical indicators performed relatively well, which were *Exponential Moving Average*, *Relative Strength Index*, *Rate of Change* and *Volatility*. These indicators are defined in detail later in this subsection. Although the preliminary testing of different combinations of inputs was not exhaustive, it was the motivation to base the experiments described in this thesis on only these four technical indicators.

Calculation of the indicators is based on *normalized* prices. *Normalization* is done by dividing each day's price by its 250 day Simple Moving Average, which is a common approach used by Allen and Karjalainen [2] and many others. Normalizing the price data allows for better generalization of the

trading rules, as indicator values are no longer dependent on the absolute price level. Because of this, rules can be expected to function properly for a longer time span as the absolute price level changes, and it is more likely that a single rule will work well for multiple stocks.

We allow each indicator to be used with any time window taken from the set {1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 17, 20, 25, 30, 35, 45, 55, 70, 90, 110, 150, 200}. The reason for this limited set of time windows is that the indicators are calculated only once at the start of the algorithm, and the results are stored in memory. This was not feasible when we allow every length between 1 and 200 for example. When the algorithm selects a time window for an indicator, one of the time windows in the set is randomly selected, with equal probability.

The normalization step requires a start up phase of 250 business days (one year) preceding the period for which we calculate the indicators, since the 250 day SMA must be known. Calculation of the indicators also requires a start up phase due to the nature of their calculation, with a length of at least the indicator's maximum time window. In order to allow for the normalization and to allow indicator period lengths of up to 200 days, a total of almost two years of stock history preceding the trading simulation is used. The rest of this subsection describes the calculation of the four indicators in detail. In the formulas, n is the size of the time window of the indicator in days, and p_t is the stock's normalized price at day t.

Exponential Moving Average

The Exponential Moving Average (EMA) is similar to the SMA which is described in the introduction, but differs in the weighting of days, where more recent days are weighted heavier. This results in a stronger responsiveness to recent price changes. Calculation is as follows:

$$\mathrm{EMA}_{n,t} = \begin{cases} p_0 & \text{if } t = 0 \\ (\frac{2}{n+1}) \times p_t + (1 - \frac{2}{n+1}) \times \mathrm{EMA}_{n,t-1} & \text{if } t > 0 \end{cases}$$

Relative Strength Index

As stated in the introduction, the Relative Strength Index (RSI) is an oscillator based on the ratio of gains during updays and losses during downdays. It is calculated as follows:

$$\mathrm{Gain}_t = \begin{cases} p_t - p_{t-1} & \text{if } p_t > p_{t-1} \\ 0 & \text{if } p_t \leq p_{t-1} \end{cases}$$

$$\text{Loss}_t = \begin{cases} 0 & \text{if } p_t \geq p_{t-1} \\ p_{t-1} - p_t & \text{if } p_t < p_{t-1} \end{cases}$$

Of these Gain and Loss series the exponential moving averages are taken, and RSI is calculated as:

$$\text{RSI}_{n,t} = 100 - \frac{100}{1 + \frac{\text{EMA}_{n,t}(\text{Gain})}{\text{EMA}_{n,t}(\text{Loss})}}$$

Rate Of Change

The Rate Of Change (ROC) indicator is an oscillator which measures the price change over the previous period, as a percentage. We divide it by n so that it represents the average daily change, in order to make two ROC indicators based on different time windows more comparable. It is calculated as:

$$\text{ROC}_{n,t} = \frac{p_t - p_{t-n}}{p_{t-n}} \times \frac{100}{n}$$

Volatility

The fourth indicator used in the algorithm is simply the standard deviation of the price over the past n days, which is a commonly used measure of price volatility. In the rest of this thesis, this indicator is referred to as SD.

2.3 Trading rules

We have seen in the previous subsection which technical indicators the trading rules can use as input signals. This subsection describes the structure of the trading rules, or in other words, how the different input signals are combined in the rules to create a single trading signal.

The work by Allen and Karjalainen [2] is based on a trading rule structure that allows for a free combination of all input signals, using an operator set which includes both boolean and arithmetic operators. Later, Becker and Seshadri [3] adjusted this structure by removing the arithmetics from the set of allowed operators, in order to evolve more comprehensible rules.

We make a further adjustment in this direction, by allowing a technical indicator to only be compared to constants or to the same technical indicator (based on a different time window). This is done because the different technical indicators that we use as input signals do not compare very well with

each other. For example, EMA(10) $\geq$ EMA(50) makes sense, as it represents a variation of the trend-following moving average crossover rule described in the introduction. On the other hand, a rule such as ROC(10) $\geq$ EMA(50) makes less sense, and we can not clearly tell whether it is trend-following or contrarian in nature.

This restriction also allows us to compare a technical indicator to a constant initialized specifically for that type of indicator. Initialization values for these constants are taken from a normal distribution that best matches the possible values of that indicator. The means and standard deviations of these distributions have been found by prior analysis of the indicator values for the full simulation period, over all stocks and all available indicator time windows.

The internal operators available to the trading rules are AND, OR, $\leq$, and $\geq$. The set of terminals consists of the technical indicators, the constants as described above, and the boolean TRUE and FALSE values. The grammar below gives an exact specification of the allowed trading rule structure, including the initialization parameters for the constants. Here, *window* is the size of the time window, randomly picked from the list of available time windows presented in subsection 2.2.

⟨*root*⟩ ::= ⟨*boolean*⟩

⟨*boolean*⟩ ::= ‘`(`’ ⟨*boolean*⟩ ‘`)`’ ⟨*logic_op*⟩ ‘`(`’ ⟨*boolean*⟩ ‘`)`’
| ⟨*roc_value*⟩ ⟨*rel_op*⟩ ⟨*roc_value*⟩
| ⟨*rsi_value*⟩ ⟨*rel_op*⟩ ⟨*rsi_value*⟩
| ⟨*ema_value*⟩ ⟨*rel_op*⟩ ⟨*ema_value*⟩
| ⟨*sd_value*⟩ ⟨*rel_op*⟩ ⟨*sd_value*⟩
| ‘`TRUE`’
| ‘`FALSE`’

⟨*logic_op*⟩ ::= ‘`AND`’ | ‘`OR`’

⟨*rel_op*⟩ ::= ‘$\leq$’ | ‘$\geq$’

⟨*roc_value*⟩ ::= ‘`roc(`’⟨*window*⟩‘`)`’ | ⟨*roc_constant* ($\mu = 0, \sigma = 1$)⟩

⟨*rsi_value*⟩ ::= ‘`rsi(`’⟨*window*⟩‘`)`’ | ⟨*rsi_constant* ($\mu = 50, \sigma = 20$)⟩

⟨*ema_value*⟩ ::= ‘`ema(`’⟨*window*⟩‘`)`’ | ⟨*ema_constant* ($\mu = 1.05, \sigma = 0.2$)⟩

⟨*sd_value*⟩ ::= ‘`sd(`’⟨*window*⟩‘`)`’ | ⟨*sd_constant* ($\mu = 0.05, \sigma = 0.05$)⟩

2.4 Genetic Programming

This subsection describes the details of the GP approach. The GP implementation that we use is developed specifically for this research. The algorithm uses a single population of 500 trading rules, which is evolved for 50 generations. The result of the algorithm is the one best individual found during these 50 generations. Algorithm 1 gives a high level overview. The rest of this subsection describes the fitness function, the specifics of the initialization of the rules and the genetic operators of selection, crossover and mutation.

Algorithm 1 GP

```
populationSize ← 500
maxGeneration ← 50
population ← initializePopulation(populationSize)
for generation = 1 to maxGeneration do
    evaluations ← calculateFitnessEvaluations(population)
    newPopulation ← ∅
    newPopulation ← getBestIndividual(population, evaluations)
    while sizeOf(newPopulation) < populationSize do
        parent ← selectParent(population, evaluations)
        operator ← selectOperator()
        if isCrossover(operator) then
            parent2 ← selectParent(population, evaluations)
            child ← performCrossover(parent, parent2, operator)
            newPopulation ← child
        else
            child ← performMutation(parent, operator)
            newPopulation ← child
    population ← newPopulation
evaluations ← calculateFitnessEvaluations(population)
return getBestIndividual(population, evaluations)
```

2.4.1 Fitness Function

The main component of the fitness function is determined by the profits generated during a trading simulation, where the trading rule in question is used to determine the trading decisions. The trading simulation is based on one continuous period, and at the start of the simulation, a virtual capital of 1 is available for investment. Returns from investments are added back to this capital, and can be reinvested. The goal is simply to maximize the

resulting capital at the end of the simulation period. We run experiments for two different types of trading simulations, which are described in detail in subsection 2.5.

The other component of the fitness function is a penalty for the size of the trading rule. This is added in order to deal with *bloat*, which is a common problem in GP applications. Bloat is the propensity of the algorithm to evolve increasingly larger trees. Evolving very large trees causes problems such as an increased computation time, stronger allowance for overfitting, and less interpretable results. We follow the approach taken by Becker and Seshadri [3], who only let the algorithm evolve relatively small trees (compared to other work such as Allen and Karjalainen [2]), with the goal of generating highly comprehensible trading rules. We set a maximum tree size of 20 nodes, and penalize any trading rule larger than this size. This is done using the following fitness function, where r is the resulting capital at the end of the trading simulation, and l is the number of nodes in the tree:

$$f = \begin{cases} r & \text{if } l \leq 20 \\ r \times \frac{120-l}{100} & \text{if } l > 20 \end{cases}$$

Using this formula, the fitness value is effectively penalized by 1% of the resulting capital for every node over the maximum size of 20. Initial test runs showed that this gives a good trade-off between properly enforcing the maximum size and not being too strict, as we want to allow rules to be slightly larger than the maximum size, if this is accompanied with increases in performance.

2.4.2 Initialization

The initial population of trees is created using the *ramped half-and-half* initialization method, which is a commonly used method suggested by Koza [10]. The goal of this initialization method is to fill the population with trees of different shapes and sizes. *Half-and-half* refers to the fact that half the trees are initialized using the *full* method and the other half using the *grow* method. Both methods construct the trees by filling them randomly from top to bottom, using a depth limit to control the tree size. The difference between the full and the grow methods is that the full method creates trees in which every branch reaches the depth limit exactly, where the grow method also allows shorter branches that terminate before reaching the depth limit, typically resulting in smaller trees with a more varied shape.

During top-to-bottom construction of the tree, for each node that is to be created, a subset of all the available node types is determined which contains the types that are allowed at the current point in the tree, in regard to our trading rule grammar. From this set of available types, one node type is randomly selected, with equal probability, and a node of this type is inserted in the tree. The set of available types depends on the current depth in the tree, the depth limit, whether the full or the grow method is being used, the parent node type, and possibly the sibling type. For example, using the full method with a depth limit of 4, the subset contains only logical operators (AND,OR) at depths 1 and 2, since inserting any other type of node would make it impossible to reach the full depth. At depth 3, all operator types are allowed, and at depth 4, we allow only terminals. The sibling type is considered when the parent node is a comparator ($\leq$,$\geq$), in order to enforce comparing only compatible technical indicators and constants to each other (comparing RSI to RSI, EMA to EMA, etc.).

The initialization method is called "ramped" because the depth limit varies randomly per tree, which causes trees of different sizes to be created. In our case, this depth limit is chosen for each tree from the range 2 to 4, with equal probability.

2.4.3 Selection

Selecting a parent from the population to be used for reproduction is done using *tournament selection*, which is a common approach used in evolutionary algorithms. In order to select one parent, a fixed number (the tournament size) of individuals are picked randomly from the population. Out of these individuals, the one with the highest fitness value is selected as the parent, to be used for reproduction. The tournament size determines the *selection pressure*, which is the degree to which individuals with a higher fitness value are favored. When this selection pressure is too low, the *convergence rate* is low, or in other words, it will take unnecessarily long to find a good solution. When the selection pressure is too high, the algorithm is more likely to converge prematurely to a local optimum. We use a tournament size of 5. Iinitial test runs using a few different values showed that this value performs relatively well.

Also, the algorithm is *elitist*, which means the one best individual in the population always survives to the next generation, without being altered by crossover or mutation.

2.4.4 Crossover and Mutation

The algorithm uses one type of crossover, and several different mutation operators, which are described in detail at the end of this subsection. The *subtree crossover*, *reproduction* and *subtree mutation* are the traditional operators described by Koza [10]. We have added a number of other mutations, with the goal of enhancing the search process. We have defined and added a "tune"-mutation, in order to enable tuning of the constants and parameters in the terminal nodes, by making small random changes to these values.

Application of the *hoist*, *shrink*, and *remove* operators all result in a decrease in tree size, and they are added to control the sizes of the trees during evolution. This is important for our application of GP, since we aim to generate only small comprehensible rules with a maximum size of 20 nodes. Using only subtree crossover and subtree mutation tends to grow trees quickly, as subtree crossover on average maintains equal tree sizes, and subtree mutation tends to increase the sizes of trees, as it often replaces small subtrees or terminals with larger subtrees. Test runs of the algorithm showed that this tends to cause a large amount of trees in each generation with a size over the maximum of 20. Adding the hoist, shrink and remove operators proved to generate more balanced populations in terms of tree size, so the algorithm does not have to rely solely on the fitness function to keep the tree size small.

The algorithm applies the different operators with different probabilities. The probabilities of each operator being used is given in parentheses in the list of operator definitions at the end of this subsection. These probabilities are chosen mainly in order to achieve a good ratio between mutations that typically increase tree sizes and mutations that decrease tree sizes. Trial runs of the algorithm showed that the chosen set of probabilities works relatively well. Using equal probabilities for all mutations results in bad performance, caused by the hoist, shrink and remove operators being applied to often which causes the generation of very small trees.

We have experimented with other combinations of probabilities but this (surprisingly) did not seem to have a strong effect on the performance of the algorithm. However, we have not tested different combinations of operators and probabilities extensively, and it could be interesting to further investigate the effects of these algorithm parameters in future work.

The implementation of the operators is complicated somewhat by our restricted grammar, as the operators must take care that the resulting trees do not violate it. In particular, we specify the following five return types that the operators need to take into account:

- Boolean (AND, OR, $\geq$, $\leq$, TRUE, FALSE)
- RSI (RSI(n) or an RSI-related constant)
- ROC (ROC(n) or a ROC-related constant)
- EMA (EMA(n) or an EMA-related constant)
- SD (SD(n) or an SD-related constant)

Using this specification, grammar violations can be avoided by implementing the operators so that they never replace a node by another node that has a different return type.

Selecting a node for the operator is always done randomly, where all nodes in the tree that are eligible have equal probability of being selected. If there are no eligible nodes in the tree to select from, the operator can not be applied. In these cases, the operation is simply omitted and the parent tree is copied to the next generation without any alteration. Note that omitting the operator and simply reusing the parent does not lead to a waste of resources as the fitness value for this tree does not need to be recalculated.

The details of the operators are given below, with the probability of the operator being used given in parentheses.

Subtree crossover (0.25) Selects any node in the first parent, and then selects a node in the second parent that has the same return type. The node in the first parent is replaced by the node from the second parent. The offspring of the node is also transferred in the process.

Reproduction (0.25) Simply copies the parent tree to become part of the next generation, without alteration. Note that, similar to omitting failed operators as described above, this does not lead to a waste of computational resources.

Subtree mutation (0.1) Selects a node in the tree, and replaces it by a new randomly created node that has the same return type. In case a non-terminal node is created, its children are initialized using the grow method that is also used in the initialization phase of the algorithm. Here, a depth limit of either 2 or 3 is used, chosen with equal probability.

Tune mutation (0.1) Selects any terminal node from the tree and changes either its value, in case of a constant, or the size of the time window, in case of an indicator. It makes only minor changes to the node. Boolean terminal nodes are simply flipped from true to false or vice-versa. Numeric constants are multiplied with a random factor, taken from a normal distribution with a mean of 1 and a standard deviation of 0.05. Indicator period lengths are either increased or decreased (with equal probability) by the smallest amount possible while still using a value from the list given in subsection 2.2.

Point mutation (0.1) Selects a node in the tree, and replaces it by a new randomly created node that has the same return type and the same type and amount of children. Only the node itself is replaced, the original children stay intact.

Hoist mutation (0.05) Selects any non-root node that has a boolean return type, and replaces the root of the tree with this node. The offspring of the selected node stays in place, so that the subtree that has the selected node as its root effectively becomes the new tree.

Shrink mutation (0.05) Selects any operator node in the tree and replaces it with a new boolean terminal node.

Remove mutation (0.05) Selects any non-root node that has a boolean return type, and moves the node (including its offspring) up one level, replacing its parent. Its original sibling (the other child of the replaced parent) is discarded.

Insert mutation (0.05) Selects a node from the tree with a boolean return type, and inserts a new logical operator node in its place. The selected node becomes one of the children of the new operator node, and a new boolean terminal node is created which becomes the other child of the new operator node.

2.5 Experimental Setup

2.5.1 Trading Simulation

For the trading simulation, we run experiments using two rather different setups. The first setup is the one used in previous research into evolving technical trading strategies, which involves trading on a single market index

or stock. Here, the stock is repeatedly bought and sold, aiming to profit from price fluctuations.

The second setup involves trading on multiple stocks simultaneously, where a trading strategy maintains an investment portfolio by picking the currently most attractive stocks. Research into evolving stock picking strategies has been done by Caplan and Becker [5], but this was based on fundamental analysis instead of technical analysis. Details of the implementation of both single stock trading and the stock picking system are given below.

Single stock trading

For single stock trading, the trading rule generates a buy (true) or sell (false) signal for every day in the simulation period. In case of a buy signal, 100% of the available capital is invested in the stock for that day. In case of a sell signal, the capital is kept completely out of the market for that day, earning a zero return.

We run this trading simulation on all the stocks in our dataset. Existing work has always evolved trading rules specifically for a single security. This is done under the assumption that occurring price patterns are to some degree specific to the security, and that a rule found for one security would not generalize well to other securities. We first take this same approach, by doing a separate run of the GP algorithm for every stock in the dataset. This results in a separate rule for each stock.

We also experiment with another approach, where we do a single run of the algorithm for all the stocks combined, producing one "collective" rule which should trade well on any of the stocks. The objective function for this collective rule is based on the average performance of the rule over all 242 stocks, where a separate trading simulation is run for each of the stocks.

Stock picking

For our stock picking approach, a trading rule is used to trade on all 242 available stocks simultaneously. Instead of only determining when to buy and sell a single stock, the trading rule picks the currently most attractive stocks, and these are invested in.

Trading is based on a single trading rule. The rule is applied to all the stocks, generating a buy or sell signal every day for every stock. This can result in any number of stocks with a buy signal for each day. Because of this, the implementation of this stock selection approach is less straightforward than single stock trading.

The simplest implementation would be to equally spread out the full investment capital each day, over all stocks that currently generate a buy

signal. Then, on days where none of the stocks generate a buy signal, the money is kept out of the market.

The problem with this approach is that the amount of capital invested per stock will vary strongly from day to day, as there will be a varying number of stocks to spread the capital over. This requires a lot of rebalancing of the investment capital, and in the presence of transaction costs, this is not very desirable.

We extend the approach to avoid this problem. The system is based on investments that span over multiple days. Once an investment in a stock is made, it is maintained for as long as the trading rule gives a buy signal for the stock, and the investment is not resized/rebalanced in the mean time. The trading strategy maintains a collection of simultaneous investments, but never two separate simultaneous investments in the same stock. The system works as follows:

For each day:

1. exit any existing investments in stocks for which the trading rule now gives a sell signal, and add the returns to the available investment capital.

2. if the available investment capital is now larger than a certain threshold, make new investments in all stocks for which holds that:

    - it is not already being invested in
    - it is given a buy signal by the trading rule

The new investments in step 2 are made by spreading the full available capital over these stocks equally. As the threshold in step 2 we use 10% of the total current capital (investments+free capital). This threshold prevents making tiny investments, in cases where only a small portion of the total capital is available for new investments, and there are many stocks for which a buy signal is generated. If the treshold is not reached, no investments are made and the available capital is kept on hand until a larger portion of the capital becomes free.

2.5.2 Data split

We use the *walk-forward* testing routine originally proposed by Robert Pardo [15] to divide the data into training and testing periods. This involves using a number of consecutive testing periods, and retraining the model for every test period on a period of training data that directly precedes it.

This walk-forward approach is well-suited for evaluating self-learning trading systems because using consecutive testing periods simulates a prolonged real-life application of the trading system. As such, the simulation gives a good indication of the profitability of the system over the total testing period. Also, the tested model is always trained on the most recent available data, which is likely to be the most relevant data.

We use the years 2000-2013 for testing, and split this period into 14 test periods of one year each. For each test year, we use the five preceding years as the training set. This means the years 1995-2012 are used for training purposes, where the years 1995-1999 are used to train for the year 2000, the years 1996-2000 are used to train for the year 2001, and so on. We use a training window size of five years because it is a fairly common value in the literature (for example Allen and Karjalainen [2] and Chen et al.[6]), and preliminary runs of our GP implementation indicated that it produces relatively high out-of-sample performance compared to both shorter and longer training windows.

2.5.3 Buy-and-hold benchmark

We compare the returns of the evolved trading strategies to the returns of the *buy-and-hold* strategy, which is a popular benchmark for evaluating trading systems. The buy-and-hold strategy is passive, simply investing in a security initially and holding on to it for a long time. In our case, we define buy-and-hold as investing all available capital in all available stocks on the first day of the simulation, using an equal amount for each stock, and then selling all stock on the last day of simulation.

Buy-and-hold is a useful benchmark, as consistently beating it by an active trading strategy is only possible if the strategy exploits some market inefficiency. In an efficient market, any trading strategy, given a long enough time-frame, would at best perform as well as the buy-and-hold strategy.

2.5.4 Transaction costs

We find that the level of transaction costs has a very strong effect on the profitability of the system, which is why we experiment with a few different levels of transaction costs for comparison. We use one-way transaction costs proportional to the size of the investment, of 0%, 0.1%, 0.25%, and 0.5%. One-way costs means that the cost is made both when buying and when selling a stock. The levels are taken from the work by Allen and Karjalainen [2], who assume 0.25% one-way transaction costs to be realistic, but also test their system with 0.1% and 0.5% cost levels.

Table 1: Annual returns (in percentages) for single stock market timing, over the 14 test periods, ignoring transaction costs. Returns (r) are averaged over all stocks, and averaged over 10 GP runs. The standard deviation (σ) of the return over the 10 GP runs is given. In-sample (In) and out-of-sample (Out) returns are given. The bottom line gives the annualized out-of-sample returns over the full test-period.

Period		Buy & Hold		Individual rules				Collective rule			
In	Out	In	Out	In		Out		In		Out	
		r	r	r	σ	r	σ	r	σ	r	σ
95-99	2000	22.2	15.8	69.5	0.4	16.1	1.5	32.7	0.8	21.4	2.3
96-00	2001	16.0	10.2	74.4	0.5	8.6	1.2	31.3	1.8	12.1	2.6
97-01	2002	12.2	-11.8	76.3	0.7	-2.9	0.8	32.6	3.0	-10.5	1.5
98-02	2003	2.0	32.1	70.6	0.3	15.4	1.1	20.3	2.6	20.4	4.7
99-03	2004	5.4	19.2	68.2	0.1	10.5	1.2	23.6	1.2	17.9	0.4
00-04	2005	8.3	10.0	64.4	0.3	7.9	0.2	19.3	0.7	10.0	0.7
01-05	2006	8.7	14.4	53.5	0.3	9.6	0.4	17.7	0.2	14.7	0.3
02-06	2007	10.3	6.6	49.9	0.3	7.9	0.5	17.1	0.4	13.7	0.7
03-07	2008	14.1	-33.3	46.6	0.2	-10.7	1.2	21.5	1.2	-29.7	4.5
04-08	2009	-1.5	38.4	50.7	0.4	21.9	2.6	16.2	0.6	27.0	4.3
05-09	2010	1.3	19.8	64.7	0.6	14.0	0.6	19.5	0.7	15.2	0.5
06-10	2011	3.1	-1.3	68.1	0.7	5.1	0.9	21.6	1.3	2.7	2.2
07-11	2012	0.1	15.2	68.6	0.4	7.9	0.6	21.4	1.2	7.8	3.1
08-12	2013	2.5	34.6	69.3	0.2	16.8	0.6	21.5	0.4	22.8	2.0
	00-13		10.45			8.84	0.34			9.25	0.43

3 Results

This section describes the results of the experiments. We start with the results from single stock trading, for both the rules evolved for individual stocks and the collective rules evolved to trade on all stocks. This is followed by the results from stock picking, and we finish with a structure analysis of the evolved rules.

3.1 Single Stock Trading

In order to gain reliable results, all experiments presented in this section are based on 10 repetitions of the GP algorithm, of which the average results are given, together with the standard deviations over the 10 runs. Table 1 shows the results for single stock trading, based on zero transaction costs. Results from evolving rules individually per stock and evolving collective rules to trade on all stocks are shown.

The in-sample returns from trading the evolved rules clearly show that

the GP algorithm is capable, in terms of finding very profitable rules in hindsight, when all price data over the simulation period is known. Buy-and-hold is beaten by a large margin every year, for both the individual and the collective rules. Also, the standard deviations over the 10 GP runs are quite low, showing that GP consistently finds very similarly performing rules. This could suggest that GP finds rules that are near optimal, given the available input signals, trading rule grammar, and data at hand.

Out-of-sample returns are much lower however. Annualized returns for the full test-period are 8.84% and 9.25% for the individually and collectively evolved rules, respectively. We can conclude from this that, even with zero transaction costs, both approaches are unable to beat the buy-and-hold benchmark over the full test-period, which earns 10.45% annually. Taking a closer look at the out-of-sample returns per year, we do see that there are a few years for which at least one of the approaches beat buy-and-hold, but during most years buy-and-hold does better. Although the rules consistently perform very well in-sample, they apparently do not generalize well to out-of-sample data.

We observe an interesting difference between the individual and the collective rules. In-sample, the individually evolved rules generate much higher returns, but out-of-sample, the collective rule is slightly superior. This outcome is returned to in the next subsection, as more notable differences are discovered.

As buy-and-hold can not even be beaten without transaction costs, it does not seem useful to test this single stock trading approach with transaction costs. This can only be expected to decrease performance further, and we omit this from the research.

3.2 Single Stock Trading and Lottery Trading

Chen et al. [7] propose a different benchmark for evaluating GP performance, which they call *Lottery Trading*. The aim is not to test whether or not the system is profitable, but it is a bare minimum benchmark used to investigate if there is anything at all that GP can learn from the training data which generalizes to out-of-sample data. Lottery Trading involves a completely random trading strategy, which goes long in a security for the same amount of days as the trading strategy it is compared with. When the evolved strategies outperform this random trading in out-of-sample tests, the conclusion can be made that they have at least some predictive value. We apply this idea as follows. We set a certain maximum to the amount of days that the evolved trading strategies can be long in the stock each year. During simulation, as soon as the trading strategy has reached this maximum, it is forced to stay

out of the market for the rest of the year. As the benchmark, we use the return that would be expected from the random strategy that is long in the market for as many days as the trading rule is allowed to be.

This expected return for a single year can be calculated as: $(1+r)^{\frac{\text{max days}}{\text{total days}}} - 1$, where r is the return of the stock for that year, *max days* denotes the maximum number of days in the market, and *total days* is the number of trading days in the year (which is between 250 and 255).

We rerun the experiments from the previous subsection for a few different amounts of maximum days in the market per year, applying the maximum both during training and testing. The results for a maximum of 50 days are given in Table 2. Summarized out-of-sample results for other amounts are given in Table 3. Full results for these amounts are omitted here, for brevity.

We can see that both the individual rules and the collective rule significantly outperform the random trading benchmark, for all amounts of maximum days. The full results for the maximum of 50 days show that the benchmark is not beaten every single year, but it is during most years. Although we have seen that the trading rules can not beat buy-and-hold, these results show that they have some predictive value out-of-sample.

Furthermore, the predictive value could be viewed as substantial, as for 5, 20 or 50 maximum days per year, the collective rules generate roughly double the returns of the benchmark. Another interesting observation is that this factor by which the benchmark is beaten increases for smaller amounts of maximum long days.

Interestingly, a more pronounced difference in performance between the individual rules and the collective rules is found here (for example, 3.50% versus 4.69% for a maximum of 50 long days per year).

On the one hand, perhaps we would expect individual rules to perform better than collective rules, because a certain stock's price might not behave quite like another stock's price. In other words, a pattern found for one stock, might not exist for another stock.

On the other hand, a rule trained specifically for a single stock is trained on slightly over 1250 data points (5 years), where the collective rule is trained on slightly over 302500 data points (1250 days $\times$242 stocks). This means a more varied training sample and much less allowance for overfitting. We can conclude that, at least for these circumstances, the advantage of the collective rule being trained on more data points outweighs the more specific training of the rules trained for a single stock.

Table 2: Annual returns (in percentages) for single stock market timing, over the 14 test periods, ignoring transaction costs. The strategies trade long in the stock at most 50 days per year. Returns (r) are averaged over all stocks, and averaged over 10 GP runs. The standard deviation (σ) of the return over the 10 GP runs is given. In-sample (In) and out-of-sample (Out) returns are given. The benchmark is the expected return of a random trading strategy that is long 50 days per year. The bottom line gives the annualized out-of-sample returns over the full test-period.

Period		Benchmark		Individual rules				Collective rule			
In	Out	In	Out	In		Out		In		Out	
		r	r	r	σ	r	σ	r	σ	r	σ
95-99	2000	4.45	3.16	39.25	0.27	5.91	0.53	12.56	0.13	7.40	1.56
96-00	2001	3.20	2.04	45.08	0.09	4.23	1.05	13.46	0.71	6.93	1.86
97-01	2002	2.44	-2.35	47.85	0.14	2.85	0.63	14.22	0.46	-0.17	1.11
98-02	2003	0.39	6.41	43.60	0.27	4.23	0.56	10.55	0.48	5.71	1.41
99-03	2004	1.07	3.83	41.76	0.14	2.62	0.23	10.62	1.24	3.23	1.01
00-04	2005	1.67	2.00	37.75	0.23	2.24	0.27	10.00	0.20	3.99	0.19
01-05	2006	1.73	2.88	30.08	0.11	2.34	0.25	7.57	0.07	-0.51	0.19
02-06	2007	2.06	1.32	31.13	0.22	3.82	0.50	7.67	0.48	5.71	2.14
03-07	2008	2.81	-6.67	27.60	0.19	0.37	0.85	7.95	0.16	7.09	2.58
04-08	2009	-0.31	7.68	32.75	0.43	5.94	1.18	8.31	0.60	14.05	6.58
05-09	2010	0.26	3.96	42.96	0.14	4.36	0.64	12.34	0.81	4.24	0.81
06-10	2011	0.61	-0.26	44.61	0.31	2.60	0.66	13.14	1.00	1.67	2.70
07-11	2012	0.03	3.05	45.96	0.34	2.48	0.55	13.76	0.28	2.17	2.63
08-12	2013	0.50	6.93	45.50	0.50	5.20	0.53	11.91	0.69	5.38	2.39
	00-13		2.36			3.50	0.07			4.69	0.59

Table 3: Overview of out-of-sample annualized returns of single stock market timing, for different amounts of maximum days long in the market per year, ignoring transaction costs. r_i and r_c denote the average return (in percentages) for the individually evolved rules and the collective rules, respectively. σ_i and σ_c denote the standard deviations of the returns over the 10 GP runs.

Max. days	Benchmark	r_i	σ_i	r_c	σ_c
125	5.66	7.08	0.21	7.91	0.37
50	2.36	3.50	0.07	4.69	0.59
20	0.96	1.44	0.14	2.12	0.47
5	0.24	0.30	0.08	0.65	0.13

3.3 Stock Picking

Despite the apparent predictive value of the trading rules, single stock trading did not lead to returns in excess of buy-and-hold, even when ignoring transaction costs. We found that when the rule has to select only a small number of days per year to be in the market, it can select days that have a substantially higher than average return. This, together with the fact that the trading rules seem to generalize quite well to other stocks, was the motivation to test the system in a stock picking context. If the rules can repeatedly recognize different stocks that are currently in such a situation where the returns will be higher than average, there would be a good chance of beating buy-and-hold.

The experimental results for different levels of transaction costs are given in table 4. Clearly, when we ignore transaction costs, the predictive ability of the system can in fact be put to good use, earning an excessive annual profit of 31%. This contrasts strongly with the results of single stock market timing.

Unfortunately, when transaction costs are included, these excess returns quickly diminish. At low costs of 0.1%, the system still generates some excess returns out-of-sample, but at cost levels that we assume to be realistic (0.25%), the rules underperform buy-and-hold with an annual return of only 4.3%.

Let us look at some statistics that shed more light on the heavy impact of added transaction costs. As we can see in table 5, the inclusion of transaction costs has a strong effect on the average duration of investments made by the system. This makes sense, as an average investment duration of 2.3 days is not maintainable at for example 0.25% transaction costs. This means reinvesting the full capital in the neighborhood of 100 times per year, meaning 200 transactions and thus a yearly cost of 50% of the full capital. Longer term investments would lead to more manageable cost levels. However, the results suggest that the longer investment durations lead to a much weaker forecasting ability.

Table 4: Annual returns (in percentages) from stock picking, over the 14 test periods, for different levels of one-way transaction costs (TC). Given are the averaged returns (r) over 10 GP runs, and the standard deviation (σ) of the return over the 10 GP runs. In-sample (In) and out-of-sample (Out) returns are given. The bottom line gives the annualized out-of-sample returns over the full test-period.

Period		Buy & Hold		TC = 0%				TC = 0.1%				TC = 0.25%				TC = 0.5%			
In	Out	In	Out	In		Out		In		Out		In		Out		In		Out	
		r	r	r	σ	r	σ	r	σ	r	σ	r	σ	r	σ	r	σ	r	σ
95-99	2000	22.2	15.8	371.1	45.2	127.8	35.5	196.6	27.3	72.1	43.6	116.4	18.2	29.0	24.4	99.0	10.3	11.0	10.1
96-00	2001	16.0	10.2	339.9	33.9	212.0	136.4	197.2	39.1	115.5	131.4	133.5	24.2	41.4	40.1	85.4	10.6	36.8	33.8
97-01	2002	12.2	-11.8	433.6	57.4	60.7	49.5	257.2	43.9	29.6	43.4	154.0	20.0	-9.8	22.3	111.1	17.3	-17.19	12.2
98-02	2003	1.97	32.1	322.2	40.8	66.2	31.4	232.5	62.8	30.8	28.8	131.3	37.8	13.0	17.4	99.1	14.7	16.18	16.5
99-03	2004	5.4	19.2	291.2	37.0	9.9	13.9	197.2	31.3	-1.7	20.0	137.1	46.3	-2.6	32.1	88.3	24.5	9.8	16.4
00-04	2005	8.3	10.0	180.3	22.3	29.5	21.0	138.9	28.8	0.7	19.3	91.9	23.4	-1.6	22.0	80.1	10.9	8.9	17.2
01-05	2006	8.7	14.4	155.1	18.7	22.5	30.6	128.3	23.0	10.5	19.1	102.4	26.3	-9.94	19.0	82.8	26.1	1.6	15.6
02-06	2007	10.3	6.6	112.1	14.0	6.1	22.4	85.3	19.6	-1.18	27.7	82.4	16.6	24.4	22.7	72.8	13.6	12.9	17.2
03-07	2008	14.1	-33.4	80.7	14.8	-32.1	17.8	67.7	7.2	-19.6	20.0	70.9	9.1	-31.0	15.1	62.7	9.1	-28.6	18.5
04-08	2009	-1.5	38.4	84.0	9.34	38.6	27.8	65.1	17.9	58.1	40.6	56.1	11.4	59.4	32.3	61.1	18.4	58.3	37.0
05-09	2010	1.3	19.8	109.3	27.4	29.8	15.6	88.3	22.2	15.7	15.9	78.9	15.4	8.3	7.1	65.1	17.0	5.6	9.9
06-10	2011	3.1	-1.3	108.8	32.9	-2.1	17.01	100.7	20.5	-5.6	14.0	69.6	11.6	-18.1	14.2	72.8	29.5	-6.3	13.8
07-11	2012	0.1	15.3	102.7	16.2	26.8	34.7	78.1	20.5	17.7	31.6	67.7	27.3	14.7	21.2	60.5	21.5	34.9	23.6
08-12	2013	2.51	34.6	114.7	21.8	26.4	15.9	93.0	21.2	5.63	23.9	85.5	26.6	16.8	14.5	58.4	20.9	15.4	9.5
	00-13		10.5			31.7	7.6			12.5	3.8			4.8	5.9			7.8	5.6

Table 5: Average investment duration during the stock picking experiments (for 10 runs of GP), averaged over the full out-of-sample test period

TC	duration
0%	2.3
0.1%	16.5
0.25%	21.4
0.5%	28.9

3.4 Trading Rule Structure Analysis

The trading rule grammar that we use in this research allows for a meaningful interpretation of the evolved trading rules. In particular, we can often identify that a rule, or part of a rule, will lead to either trend-following or contrarian investing. This subsection describes an interpretation of one example rule, followed by statistics of all rules evolved during the stock picking experiments.

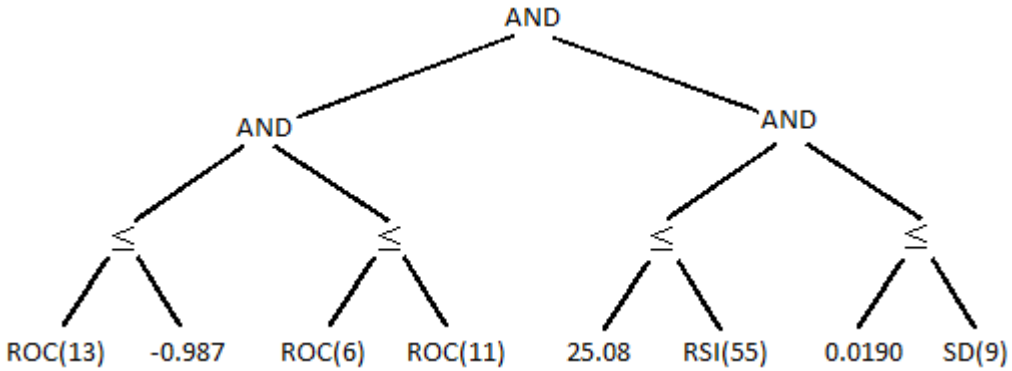

Figure 2: Rule evolved on training period 1999-2003, for the stock selection environment without transaction costs.

Figure 2 shows a rule developed for the stock picking simulation. This rule contains four comparisons involving indicators and constants, and for a stock to be invested in, all four comparisons need to evaluate to true.

The comparison ROC(13) $\leq$ -0.987 is true when the price has dropped substantially over the past 13 days, by a minimum of almost 1% per day. This part of the rule will steer the strategy towards contrarian investing, as it signals to buy stocks on the basis of sharp recent price decreases. ROC(6) $\leq$ ROC(11) is true when the daily returns over the past 6 days are less than or equal to the daily returns over the past 11 days, which means the

daily returns can not have increased recently. This supports the contrarian investing style.

The comparison $25.08 \leq \text{RSI}(55)$ gives a lower limit for RSI. As RSI is a measure of price strength (recent price increases), this will cause the strategy to avoid investing in stocks that have underperformed recently. This opposes the contrarian investing signals of the two ROC comparisons. However, the RSI is based on a longer time window, and the effect of the three ROC and RSI comparisons together is that the strategy buys stocks that have performed very poorly recently, but not so poorly over a longer time frame.

In addition, the fourth comparison $0.0190 \leq \text{SD}(9)$ leads to investing in stocks for which the volatility is relatively high.

Of course, such analysis of a single rule does not lead to reliable conclusions about apparent market conditions or patterns, as the degree to which the rule or different parts of it are based on noise in the training data is unknown. However, most evolved rules appear to be quite similar to this example rule in the sense that they are largely based on a contrarian (counter-trend) investing style. In order to quantify this observation, we analyze all the trading rules evolved for the stock picking simulation by counting certain types of comparisons that make up the rules.

The RSI, ROC, and EMA indicators are all in a way measures of recent price levels or returns, where higher values indicate higher prices and returns. Because of this, a comparison that involves one of these indicators will be true either for relatively high recent returns or for relatively low recent returns. In the same way, comparisons involving the SD indicator will be true for either a high or a low recent volatility. An overview of how we can classify the different comparisons by these terms is given in table 6.

We count the occurrences of these comparisons in all the 140 (14 test periods $\times$ 10 GP runs) evolved rules, of which the results are given in table 7. We also count the number of rules that contain at least one of a certain type of comparison. Note that the trading rule grammar does not contain boolean operators that can invert a signal, such as NOT, so the effect of included comparisons on the trading strategy is relatively straightforward. A comparison that is true in the case of relatively high recent returns is associated with trend-following, and a comparison that is true for relatively low recent returns is associated with contrarian (counter-trend) investing.

From these counts, any inactive subtrees in a rule are omitted. An example of an inactive subtree is (*true* OR $\text{RSI}(14) \leq 30$), which will always return true and therefore does not have any meaningful effect on the activity of the trading strategy.

Table 6: Classification of comparisons into: favoring relatively high recent returns (associated with trend-following), favoring relatively low recent returns (associated with contrarian investing), favoring relatively high or low recent volatility, or inactive (constant true or false evaluation).

Structure	Indicators	Window sizes	Classification
constant $\leq$ indicator(n)	rsi,roc,ema		high returns
	sd		high volatility
indicator(n) $\leq$ constant	rsi,roc,ema		low returns
	sd		low volatility
indicator(n_a) $\leq$ indicator(n_b)	rsi,roc,ema	$n_a > n_b$	high returns
		$n_a < n_b$	low returns
	sd	$n_a > n_b$	high volatility
		$n_a < n_b$	low volatility
	all	$n_a = n_b$	inactive
constant $\leq$ constant			inactive

Table 7: Numbers of trees and comparisons of a certain type (using the classification given in table 6), generated for the stock picking simulation for various levels of transaction costs. Total results are given for 10 GP runs over the 14 test periods. The tree counts represent the numbers of evolved trees that contain at least one of the mentioned type of comparison.

Type	TC=0%	TC=0.1%	TC=0.25%	TC=0.5%
total comparisons	686	647	619	580
RSI comparisons	207	177	86	55
ROC comparisons	250	195	107	84
EMA comparisons	78	137	256	302
SD comparisons	151	138	175	139
high return comparisons	99	154	149	196
low return comparisons	436	355	295	245
high volatility comparisons	95	69	84	81
low volatility comparisons	56	69	91	58
total trees	140	140	140	140
trees with high return comparison	57	88	92	103
trees with low return comparison	135	126	118	110
trees with high vlt. comparison	68	49	59	63
trees with low vlt. comparison	45	48	63	44

The resulting counts lead to a few different observations. Most notably, for zero transaction costs, the rules have a strong tendency to use comparisons that signal relatively low recent returns. This is the case for 436 out of the 535 RSI, ROC en EMA comparisons. Out of the 140 trees, 135 contain such a comparison element, while only 57 trees contain a comparison that signals high recent returns. These counts suggest that for zero transaction costs the rules are typically based on a contrarian investing style.

With increasing transaction costs, this tendency gradually decreases, and for the highest level (0.5%), there is only a minor difference.

Another observation is that for zero transaction costs, the rules use the RSI and ROC indicators the most, and make little use of the EMA indicator. For increasing transaction cost levels the EMA is used more and more, and for transaction cost levels of 0.5%, more than half the comparisons are based on EMA. It is unclear why this is the case and it might be interesting to investigate this further.

The SD indicator is included in quite a lot of rules, but not consistently used to either signal high volatility or low volatility, as these numbers are roughly equal. It is unclear from these results whether the use of the SD indicator adds to the forecasting ability of the rules, or whether its inclusion in rules is based on noise and overfitting.

4 Conclusion

We have seen that for single stock market timing, the evolved rules could not beat the buy-and-hold benchmark out-of-sample, even when ignoring transaction costs. Nevertheless, comparing the rules to randomized trading shows that they definitely have some forecasting ability. We also find that training the rules on a variety of stocks leads to a higher predictive value, where previous research has trained rules specifically for a single security.

In a stock picking context, we find that the predictive value of the system can be put to better use, leading to strong out-performance of buy-and-hold when transaction costs are ignored. The evolved strategies are shown to be based on a short-term contrarian investment style.

Unfortunately, introducing transaction costs to the system forces longer investment durations, and the excess returns disappear for a reasonable level of transaction costs.

We can conclude from the experiments that stock prices are not entirely unpredictable, but the Efficient Market Hypothesis is not challenged by this research, as economic profits are prohibited by transaction costs.

Preliminary testing of our GP implementation during development on a few stocks and market indices showed a variety of results, where in some cases out-of-sample results were easily in excess of buy-and-hold. This later appeared to be based on luck, and we found that experimenting with the algorithm on a large number of stocks and testing periods was important for producing reliable results.

In terms of testing the Efficient Market Hypothesis, using the stock picking context seems to be an interesting direction for future research. One possible extension of the approach presented in this thesis is the use of co-evolved pairs of buy and sell rules. This might help the algorithm to make better long term investments, which is vital for beating the large transaction costs in the stock markets. Using separate buy and sell rules has been done successfully by Becker and Seshadri [3], and it would be interesting to see it in this stock picking context.

References

[1] Yahoo! Finance. `http://finance.yahoo.com`. Accessed: 2014-01-05.

[2] Franklin Allen and Risto Karjalainen. Using genetic algorithms to find technical trading rules. *Journal of financial Economics*, 51(2):245–271, 1999.

[3] Lee A Becker and Mukund Seshadri. Gp-evolved technical trading rules can outperform buy and hold. In *Proceedings of the sixth international conference on computational intelligence and natural computing*, volume 26, 2003.

[4] Markus Brameier. On linear genetic programming. 2005.

[5] Michael Caplan and Ying Becker. Lessons learned using genetic programming in a stock picking context. *Genetic programming theory and practice II*, pages 87–102, 2005.

[6] Shu-Heng Chen, Tzu-Wen Kuo, and Kong-Mui Hoi. Genetic programming and financial trading: How much about what we know. *Handbook of financial engineering*, pages 99–154, 2008.

[7] Shu-Heng Chen and Nicolas Navet. Pretests for genetic-programming evolved trading programs:zero-intelligence strategies and lottery trading. In *Neural Information Processing*, pages 450–460. Springer Berlin Heidelberg, 2006.

[8] Eugene F Fama. The behavior of stock-market prices. *Journal of business*, pages 34–105, 1965.

[9] Michael C Jensen. Some anomalous evidence regarding market efficiency. *Journal of financial economics*, 6(2):95–101, 1978.

[10] John R Koza. *Genetic programming: on the programming of computers by means of natural selection*, volume 1. MIT press, 1992.

[11] Dome Lohpetch and David Corne. Discovering effective technical trading rules with genetic programming: Towards robustly outperforming buy-and-hold. In *Nature & Biologically Inspired Computing, 2009. NaBIC 2009. World Congress on*, pages 439–444. IEEE, 2009.

[12] Dome Lohpetch and David Corne. Outperforming buy-and-hold with evolved technical trading rules: Daily, weekly and monthly trading. *Applications of Evolutionary Computation*, pages 171–181, 2010.

[13] Dome Lohpetch and David Corne. Multiobjective algorithms for financial trading: Multiobjective out-trades single-objective. In *Evolutionary Computation (CEC), 2011 IEEE Congress on*, pages 192–199. IEEE, 2011.

[14] Christopher Neely, Paul Weller, and Rob Dittmar. Is technical analysis in the foreign exchange market profitable? a genetic programming approach. *Journal of Financial and Quantitative Analysis*, 32(04):405–426, 1997.

[15] Robert Pardo. *Design, testing, and optimization of trading systems*, volume 2. John Wiley & Sons, 1992.

[16] Jean-Yves Potvin, Patrick Soriano, and Maxime Vallée. Generating trading rules on the stock markets with genetic programming. *Computers & Operations Research*, 31(7):1033–1047, 2004.

[17] Guy L Steele. *Common LISP: the language*. Digital press, 1990.

[18] Thijs Vermeulen. A literature review on evolving trading rules using genetic programming.