Chapter 1

FORECASTING MARKET INDICES USING EVOLUTIONARY AUTOMATIC PROGRAMMING

A case study

Michael O'Neill
University of Limerick
Ireland
michael.oneill@ul.ie

Anthony Brabazon
University College Dublin
Ireland
anthony.brabazon@ucd.ie

Conor Ryan
University of Limerick
Ireland
conor.ryan@ul.ie

Abstract This study examines the potential of an evolutionary automatic programming methodology, Grammatical Evolution, to uncover a series of useful technical trading rules for market indices. A number of markets are analysed; these are the UK's FTSE, Japans Nikkei, and the German DAX. The preliminary findings indicate that the methodology has much potential.

Keywords: Evolutionary automatic programming, grammatical evolution, market indices, technical trading rules, FTSE, DAX, Nikkei

1. Introduction

The objective of this study is to determine whether an evolutionary automatic programming methodology, Grammatical Evolution (GE), is capable of uncovering useful technical trading rules for a number of market indices.

We will begin by providing the background and motivation to this line of research, followed by descriptions of the problem domain and of the evolutionary automatic programming approach adopted. A demonstration of how GE can be applied to index trading will provided by conducting experiments on data from three different markets. The chapter will end with some discussion and conclusions.

1.1 Technical analysis

A market index is comprised of a weighted average measure of the price of individual shares which make up that market. The value of the index represents an aggregation of the balance of supply and demand for these shares. Some market traders, known as technical analysts, believe that prices move in trends and that price patterns repeat themselves [Murphy, 1999]. If we accept the premise that there are rules, although not necessarily static rules, underlying price behaviour, it follows that trading decisions could be enhanced through use of an appropriate rule induction methodology such as Grammatical Evolution (GE).

Although controversy exists amongst financial theorists regarding the veracity of the claim of technical analysts, recent evidence has suggested that it may indeed be possible to uncover patterns of predictability in price behaviour. Brock, Lakonishok and LeBaron [Brock, Lakonishok & LeBaron, 1992] found that simple technical trading rules had predictive power and suggested that the conclusions of earlier studies that technical trading rules did not have such power were premature. Other studies which indicated that there may be predictable patterns in share price movements include those which suggest that markets do not always impound new information instantaneously [Chan, Jegadeesh & Lakonishok, 1996], and that stock markets can overreact as a result of excessive investor optimism or pessimism [Dissanaike, 1997]. The continued existence of large technical analysis departments in international finance houses is consistent with the hypothesis that technical analysis has proven empirically useful.

1.2 Potential for application of evolutionary automatic programming

As noted by Iba and Nikolaev [Iba & Nikolaev, 2000] there are a number of reasons to suppose that the use of an evolutionary automatic programming (EAP) approach can prove fruitful in the financial prediction domain. EAP can conduct an efficient exploration of the search space and can uncover dependencies between input variables, leading to the selection of a good subset for inclusion in the final model. Additionally, use of EAP facilitates the utilisation of complex fitness functions including discontinuous, non-differentiable functions. This is of particular importance in the financial domain as the fitness criterion may be complex, usually requiring a balancing of return and risk. EAP, unlike, for example, basic neural net approaches to financial prediction, does not require the ex-ante determination of optimal model inputs and their related transformations. Another useful feature of EAP is that it produces human-readable rules that have the potential to enhance understanding of the problem domain.

1.3 Motivation for study

This study was motivated by a number of factors. Much of the existing literature concerning the application of genetic algorithms (GA) or GP to the generation of technical trading rules [Allen & Karjalainen, 1999] [Bauer, 1994] [Neely, Weller & Dittmar, 1997] [Deboeck, 1994] concentrates on the US and to a lesser extent the Japanese stock markets. Published research on this area is both incomplete and scarce. To date, only a limited number of GA / GP methodologies and a limited range of technical indicators have been considered. This study addresses these limitations by examining index data drawn from a number of markets, that is, the UK, German, and Japanese stock markets, and by adopting a novel evolutionary automatic programming approach.

The chapter is organised as follows. Section two discusses the background to the technical indicators utilised in this study. Section three describes the evolutionary algorithm adopted, Grammatical Evolution [O'Neill & Ryan, 2001] [O'Neill, 2001] [Ryan et.al., 1998]. Section four outlines the data and function sets used. The following sections provide the results of the study followed by a discussion of these results and finally a number of conclusions are derived.

2. Background

As with any modelling methodology, issues of data pre-processing need to be considered. Rather than attempting to uncover useful technical trading rules for an index using raw current and historical price information, this information is initially pre-processed. The objective of these pre-processing techniques is to uncover possible useful trends and other information in the time series of the raw index data whilst simultaneously reducing the noise inherent in the series.

2.1 Technical Indicators

The development of trading rules based on current and historic market price information has a long history [Brown, Goetzmann & Kumar, 1998]. The process entails the selection of one or more technical indicators and the development of a trading system based on these indicators. These indicators are formed from various combinations of current and historic price information. Although there are potentially an infinite number of such indicators, the financial literature suggests that certain indicators are widely used by investors [Brock, Lakonishok & LeBaron, 1992][Murphy, 1999] [Pring, 1991].

Four groupings of indicators are given prominence in prior literature:

i. Moving average indicators

ii. Momentum indicators

iii. Trading range indicators

iv. Oscillators

Given the large search space, an evolutionary automatic programming methodology has promise to determine both a good quality combination of, and relevant parameters for, trading rules drawn from individual technical indicators.

We intend to use of each of these groupings as our model is developed. Our initial study on the FTSE dataset [O'Neill et.al., 2001] included only a moving average indicator. This study also includes momentum, and trading range volatility indicators.

Moving Average Indicators.

The simplest moving average systems compare the current share price or index value with a moving average of the share price or index value over a lagged period, to determine how far the current price has moved from an underlying price trend. As they smooth out daily price fluctuations, moving averages can heighten the visibility of an underlying

trend. A variation on simple moving average systems is to use a moving average convergence divergence (MACD) oscillator. This is calculated by taking the difference of a short run and a long run moving average. In a recursive fashion, more complex combinations of moving averages of values calculated from a MACD oscillator can themselves be used to generate trading rules. For example, a nine day moving average of a MACD oscillator could be plotted against the raw value of that indicator. A trading signal may be generated when the two plotted moving averages cross. Moving average indicators are trend following devices and work best in trending markets. They can have a slow response to changes in trends in markets, missing the beginning and end of each move. They tend to be unstable in sideways moving markets, generating repeated buy and sell signals (whipsaw) leading to unprofitable trading. Trading systems using moving averages trade-off volatility (risk of loss due to whipsaw) against sensitivity. The objective is to select the lag period which is sensitive enough to generate a useful early trading signal but which is insensitive to random noise.

Momentum.

The momentum of a security is the ratio of a time-lagged price to the current price ($\frac{Price_t}{Price_{t-x}}$). The belief underlying this indicator is that strongly trending shares tend to continue to move in that direction for a period of time as more investors buy or sell the trending share. There is recent evidence that momentum trading strategies can work, particularly when investing in smaller firms. Technical analysts consider that price momentum can foretell a price turning point as momentum will tend to peak before the price peaks.

Trading Range Breakout systems.

In these systems, a signal is usually generated if a price breaks out of a defined range. A simple example of a trading rule would be to buy a share when it exceeds its previous high in the last four weeks and conversely to sell a share if it falls below its previous four week low. A more complex approach is to plot an envelope at $\pm$ 'x' standard deviations above and below a moving average. Penetration of the bands by the current day's price indicates a possible price trend reversal.

A description of the evolutionary automatic programming system used to evolve trading rules now follows.

3. Grammatical Evolution

Grammatical Evolution (GE) is an evolutionary algorithm that can evolve computer programs in any language [O'Neill & Ryan, 2001] [O'Neill, 2001] [Ryan et.al., 1998]. Rather than representing the programs as syntax trees, as in GP [Koza, 1992], a linear genome representation is used. Each individual, a variable length binary string, contains in its codons (groups of 8 bits) the information to select production rules from a Backus Naur Form (BNF) grammar. As such, GE adopts a biologically-inspired genotype-phenotype mapping process.

At present, the search element of the system is carried out by an evolutionary algorithm, although other search strategies with the ability to operate over variable-length binary strings have been used [O'Sullivan & Ryan, 2002]. In particular, future advances in the field of evolutionary algorithms can be easily incorporated into this system due to the program representation.

3.1 The Biological Approach

The GE system is inspired largely by the biological process of generating a protein from the genetic material of an organism. Proteins are fundamental in the proper development and operation of living organisms and are responsible for traits such as eye colour and height [Lewin, 2000].

The genetic material (usually DNA) contains the information required to produce specific proteins at different points along the molecule. For simplicity, consider DNA to be a string of building blocks called nucleotides, of which there are four, named A, T, G, and C, for adenine, tyrosine, guanine, and cytosine respectively. Groups of three nucleotides, called codons, are used to specify the building blocks of proteins. These protein building blocks are known as amino acids, and the sequence of these amino acids in a protein is determined by the sequence of codons on the DNA strand. The sequence of amino acids is very important as it plays a large part in determining the final three-dimensional structure of the protein, which in turn has a role to play in determining its functional properties.

In order to generate a protein from the sequence of nucleotides in the DNA, the nucleotide sequence is first transcribed into a slightly different format, that being a sequence of elements on a molecule known as RNA. Codons within the RNA molecule are then translated to determine the sequence of amino acids that are contained within the protein molecule. The application of production rules to the non-terminals of the incomplete code being mapped in GE is analogous to the role amino acids

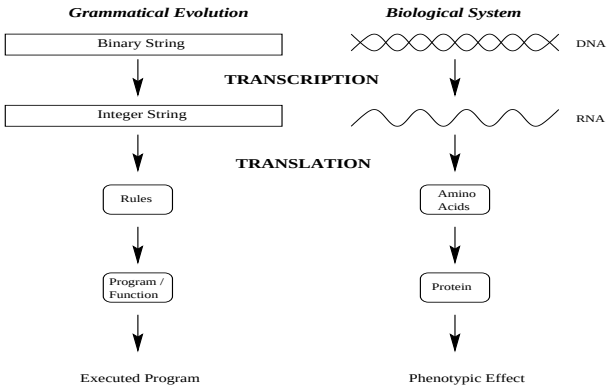

Figure 1.1. A comparison between the grammatical evolution system and a biological genetic system. The binary string of GE is analogous to the double helix of DNA, each guiding the formation of the phenotype. In the case of GE, this occurs via the application of production rules to generate the terminals of the compilable program. In the biological case by directing the formation of the phenotypic protein by determining the order and type of protein subcomponents (amino acids) that are joined together.

play when being combined together to transform the growing protein molecule into its final functional three-dimensional form.

The result of the expression of the genetic material as proteins in conjunction with environmental factors is the phenotype. In GE, the phenotype is a computer program that is generated from the genetic material (the genotype) by a process termed a genotype-phenotype mapping. This is unlike the standard method of generating a solution (a program in the case of GE) directly from an individual in an evolutionary algorithm by explicitly encoding the solution within the genetic material. Instead, a many-to-one mapping process is employed within which the robustness of the GE system lies.

Figure 1.1 compares the mapping process employed in both GE and biological organisms.

3.2 The Mapping Process

When tackling a problem with GE, a suitable BNF (Backus Naur Form) grammar definition must first be decided upon. The BNF can be either the specification of an entire language or, perhaps more usefully, a subset of a language geared towards the problem at hand.

In GE, a BNF definition is used to describe the output language to be produced by the system. BNF is a notation for expressing the grammar of a language in the form of production rules. BNF grammars

consist of `terminals`, which are items that can appear in the language, e.g. binary operators **+**, **-**, unary operators **Sin**, constants **1.0** etc. and `non-terminals`, which can be expanded into one or more terminals and non-terminals. For example from the grammar detailed below, `<expr>` can be transformed into one of four rules, i.e it becomes `<expr><op><expr>`, `(<expr><op><expr>)` (which is the same as the first, but surrounded by brackets), `<pre-op>(<expr>)`, or `<var>`. A grammar can be represented by the tuple $\{N, T, P, S\}$, where N is the set of non-terminals, T the set of terminals, P a set of production rules that maps the elements of N to T, and S is a start symbol which is a member of N. When there are a number of productions that can be applied to one element of N the choice is delimited with the '|' symbol. For example,

```
N = { <expr>, <op>, <pre_op> }
T = { Sin, +, -, /, *, X, 1.0, (, ) }
S = <expr>
```

And P can be represented as:

```
(A) <expr> ::= <expr> <op> <expr>     (0)
             | ( <expr> <op> <expr> ) (1)
             | <pre-op> ( <expr> )    (2)
             | <var>                  (3)
(B) <op> ::= +       (0)
           | -       (1)
           | /       (2)
           | *       (3)
(C) <pre-op> ::= Sin
(D) <var> ::= X      (0)
            | 1.0    (1)
```

The compilable code produced will consist of elements of the terminal set T. The grammar is used in a developmental approach whereby the evolutionary process evolves the production rules to be applied at each stage of a mapping process, starting from the start symbol, until a complete program is formed. A complete program is one that is comprised solely from elements of T.

As the BNF definition is a plug-in component of the system, it means that GE can produce code in any language thereby giving the system a unique flexibility.

For the above BNF, Table 1.1 summarizes the production rules and the number of choices associated with each.

The genotype is used to map the start symbol onto terminals by reading codons of 8 bits to generate a corresponding integer value, from which an appropriate production rule is selected by using the following mapping function:

$$Rule = Codon\ Value\ MOD\ No.\ Rule\ Choices$$

Rule no.	Choices
A	4
B	4
C	1
D	2

Table 1.1. The number of choices available from each production rule.

Consider the following rule from the given grammar i.e., given the non-terminal *op*, which describes the set of binary operators that can be used, there are four production rules to select from.

```
(B) <op> ::=    +          (0)
              | -          (1)
              | /          (2)
              | *          (3)
```

If we assume the codon being read produces the integer 6, then

$$6\ MOD\ 4\ =\ 2$$

would select rule (2) /. Each time a production rule has to be selected to transform a non-terminal, another codon is read. In this way the system traverses the genome.

During the genotype-to-phenotype mapping process it is possible for individuals to run out of codons, and in this case we wrap the individual and reuse the codons. This is quite an unusual approach in EAs, as it is entirely possible for certain codons to be used two or more times. This technique of wrapping the individual draws inspiration from the gene-overlapping phenomenon that has been observed in many organisms [Lewin, 2000].

In GE, each time the same codon is expressed it will always generate the same integer value, but, depending on the current non-terminal to which it is being applied, it may result in the selection of a different production rule. We refer to this feature as intrinsic polymorphism. What is crucial, however, is that each time a particular individual is mapped from its genotype to its phenotype, the same output is generated. This is the case because the same choices are made each time. However, it is possible that an incomplete mapping could occur, even after several wrapping events, and in this case the individual in question is given the lowest possible fitness value. The selection and replacement mechanisms then operate accordingly to increase the likelihood that this individual is removed from the population.

An incomplete mapping could arise if the integer values expressed by the genotype were applying the same production rules repeatedly. For

example, consider an individual with three codons, all of which specify rule 0 from below,

```
(A) <expr> :: =  <expr><op><expr>       (0)
                |(<expr><op><expr>)     (1)
                |<pre-op>(<expr>)       (2)
                |<var>                  (3)
```

even after wrapping the mapping process would be incomplete and would carry on indefinitely unless stopped. This occurs because the nonterminal `<expr>` is being mapped recursively by production rule 0, i.e., it becomes `<expr><op><expr>`. Therefore, the leftmost `<expr>` after each application of a production would itself be mapped to a `<expr><op><expr>`, resulting in an expression continually growing as follows: `<expr><op><expr><op><expr><op><expr>` etc.

Such an individual is dubbed invalid as it will never undergo a complete mapping to a set of terminals. For this reason we impose an upper limit on the number of wrapping events that can occur. It is clearly essential that stop sequences are found during the evolutionary search in order to complete the mapping process to a functional program. The stop sequence being a set of codons that result in the non-terminals being transformed into elements of the grammars terminal set.

Beginning from the left hand side of the genome then, codon integer values are generated and used to select rules from the BNF grammar, until one of the following situations arise:

i. A complete program is generated. This occurs when all the non-terminals in the expression being mapped are transformed into elements from the terminal set of the BNF grammar.

ii. The end of the genome is reached, in which case the *wrapping* operator is invoked. This results in the return of the genome reading frame to the left hand side of the genome once again. The reading of codons will then continue, unless an upper threshold representing the maximum number of wrapping events has occurred during this individual's mapping process.

iii. In the event that a threshold on the number of wrapping events has occurred and the individual is still incompletely mapped, the mapping process is halted, and the individual is assigned the lowest possible fitness value.

To reduce the number of invalid individuals being passed from generation to generation, a steady state replacement mechanism is employed. One consequence of the use of a steady state method is its tendency to maintain fit individuals at the expense of less fit, and in particular, invalid individuals.

4. Problem Domain & Experimental Approach

We describe an approach to evolving trading rules using GE. This study uses daily data for the UK FTSE 100, the German DAX, and the Japanese NIKKEI stock indices.

The FTSE data is drawn from the period 26/04/1984 to 4/12/1997, the training data set was comprised of 365 days from the first day plus an additional 75 days at the beginning of this time to allow for the time lag introduced with technical indicators such as the moving average. The remaining data is divided into five hold out samples totaling 2125 trading days, see Figure 1.2.

The DAX and NIKKEI data are drawn from the period 01/01/1991 to 03/12/1997, with the initial 440 days becoming the training data set as before. The remaining data is divided into two hold out samples in each case. These periods can be seen in Figure's 1.3 and 1.4

The division of the hold out period into a number of segments is undertaken to allow comparison of the out of sample results across different market conditions, in order to assess the stability and degradation characteristics of the developed model's predictions.

The rules evolved by GE are used to generate one of three signals for each day of the training or test periods. The possible signals are *Buy*, *Sell*, or *Do Nothing*. Permitting the model to output a Do Nothing signal reduces the hard threshold problem associated with production of a binary output. This issue has not been considered in a number of prior studies. A variant on the trading methodology developed in Brock et al. [Brock, Lakonishok & LeBaron, 1992] is then applied.

If a buy signal is indicated, a fixed investment of $1,000 (arbitrary) is made in the market index. This position is closed at the end of a ten day (arbitrary) period.

On the production of a sell signal, an investment of $1,000 is sold short and again this position is closed out after a ten day period. This gives rise to a maximum potential investment of $10,000 at any point in time (the potential loss on individual short sales is in theory infinite but in practice is unlikely to exceed $1,000). The profit (or loss) on each transaction is calculated taking into account a one-way trading cost of 0.2% and allowing a further 0.3% for slippage. The total return generated by the developed trading system is a combination of its trading return and its risk free rate of return generated on uncommitted funds.

The rate adopted in this calculation is simplified to be the average interest rate over the entire data set. For the FTSE the rate is 8.5%, the DAX 6.0% and for the NIKKEI it is 1.5%.

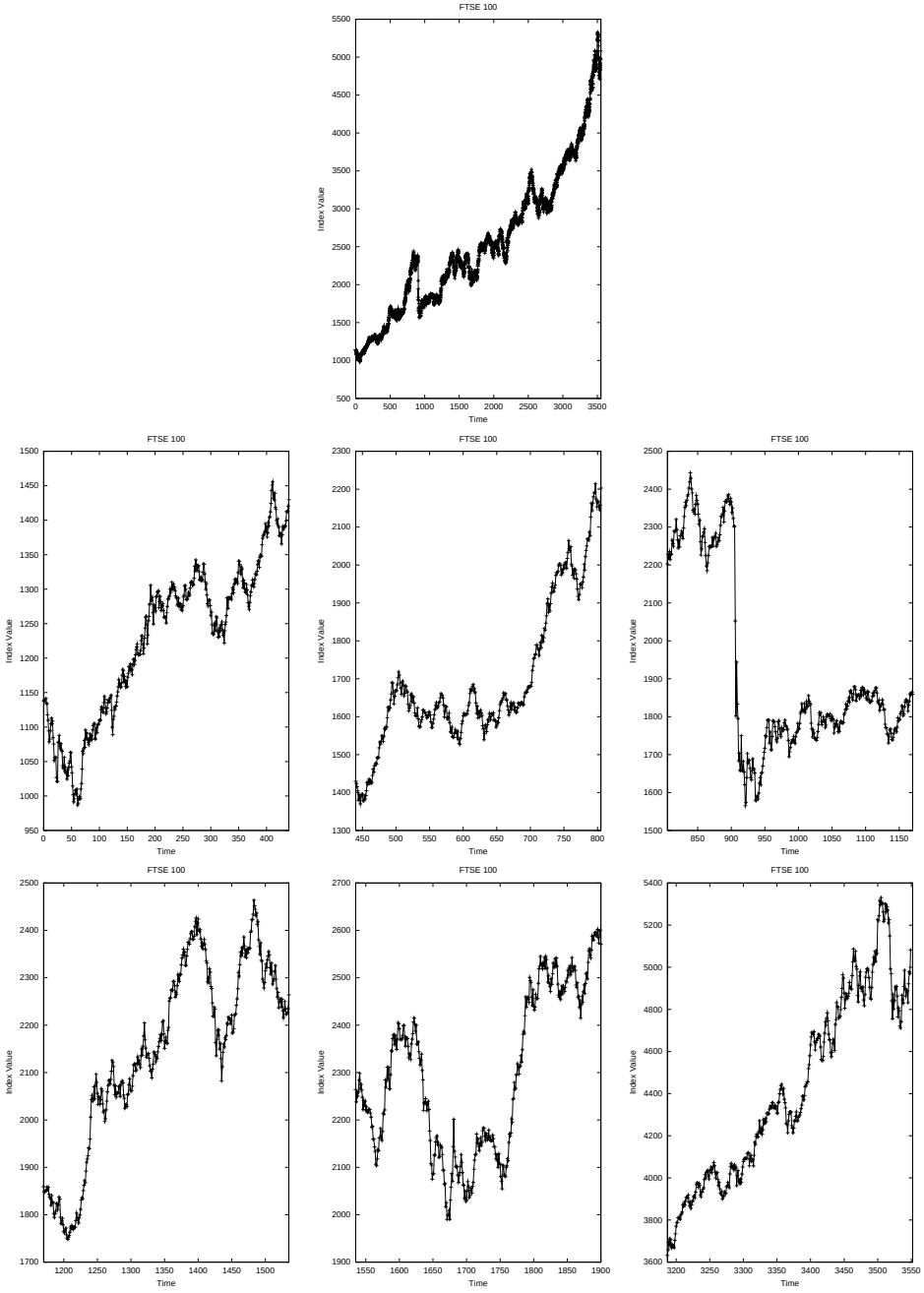

Figure 1.2. A plot of the FTSE 100 over the entire data set (top), over the training period (middle-left), over the first two test periods. Days 365 to 730 (middle-center), and days 730 to 1095 (middle-right), and the third, fourth & fifth test periods (bottom row, from left to right).

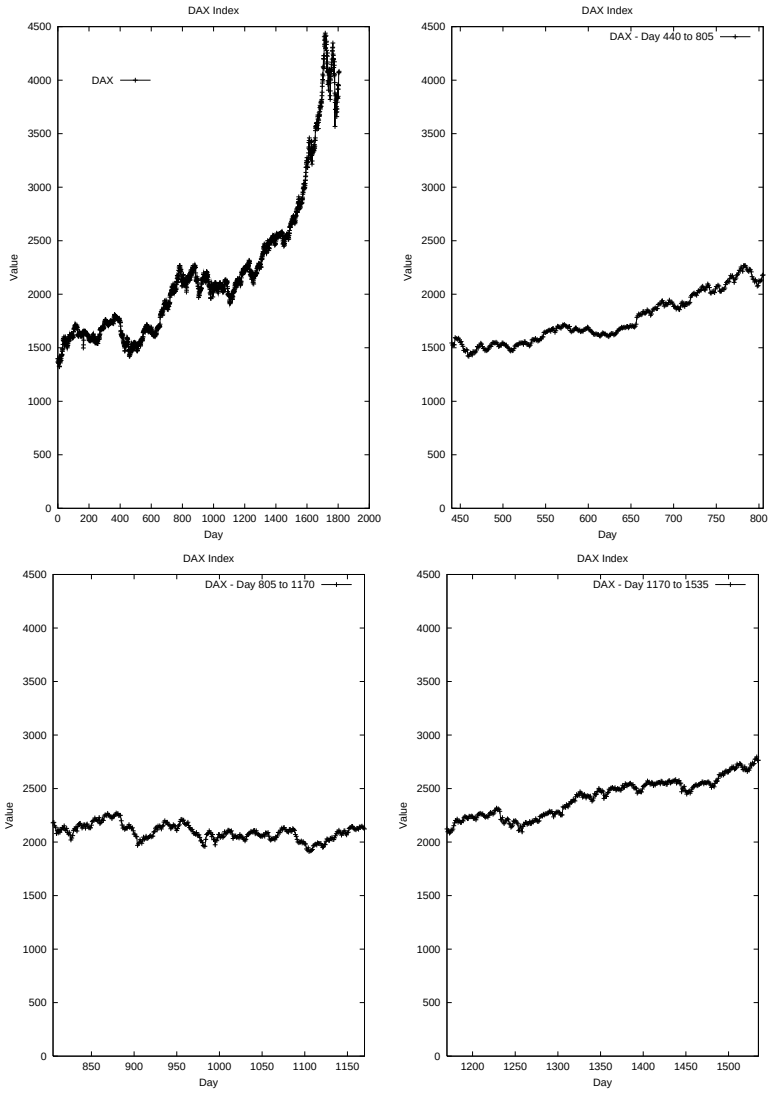

Figure 1.3. A plot of the DAX over the entire data set (top left). Taken from this data set period illustrated we can see the training period (top right), and the two test periods (bottom left and right respectively).

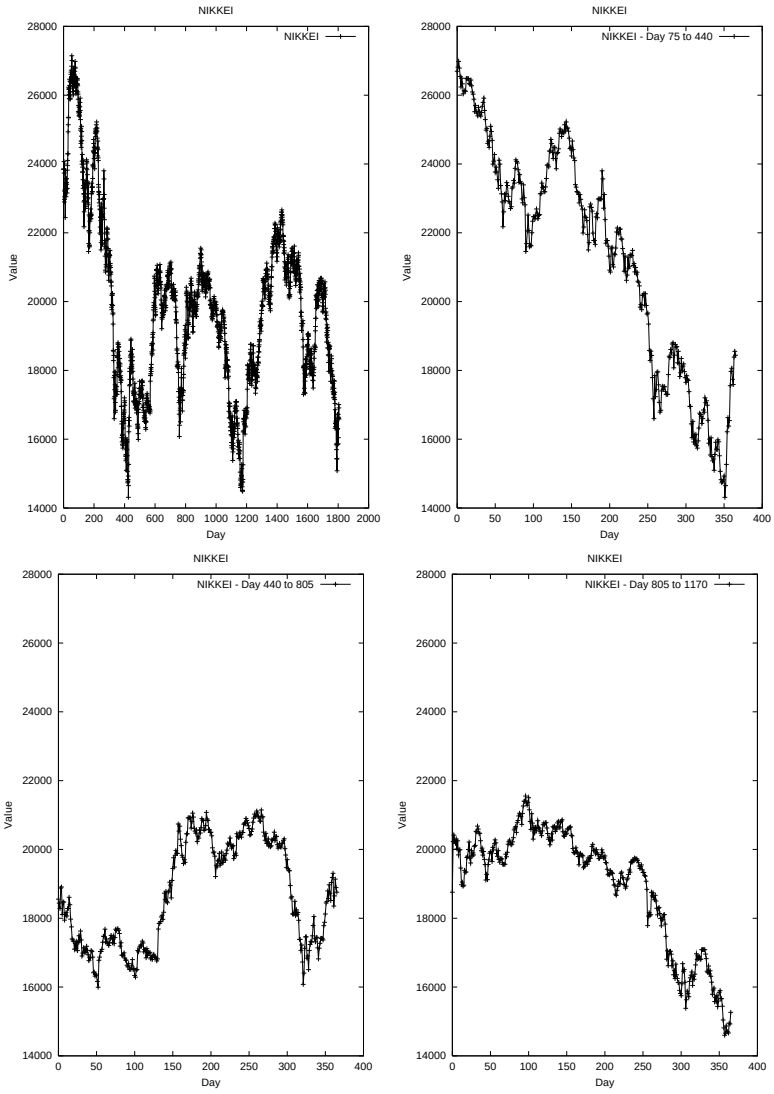

Figure 1.4. A plot of the Nikkei over the entire data set (top left), over the training period (top right), over the two test periods (bottom left and right respectively).

As well as the moving average, the momentum and trading range volatility technical indicators are adopted in these preliminary experiments, as can be seen in the grammar, given below.

```
N={<code>,<expr>,<fopbi>,<fopun>,<matbi>,<relbi>,<var>,<int>}
T={p,=,(,),f_and,f_or,f_not,+,-,*,>,<,>=,<=,scale,ma,day,1,2,3,4,5,10}
S=<code>
P={ <code> ::=  p = <expr> ;

<expr> ::= <fopbi> (<expr>, <expr>) | <fopun> (<expr>) | <expr><matbi><expr>
          | <expr><relbi><expr> | <var>

<fopbi> ::= f_and | f_or

<fopun> ::= f_not

<matbi> ::= + | - | *

<relbi> ::= > | < | >= | <=

<var> ::= <int> | day | ma(<int>,day) | momentum(<int>,day)  | trb(<int>,day)

<int> ::= 1 | 2 | 3 | 4 | 5 | 10 }
```

In addition to the technical indicators the grammar also allows the use of the binary operators `f_and`, `f_or`, the standard arithmetic operators, and the unary operator `f_not`, and the current days index value `day`. The operations `f_and`, `f_or`, and `f_not` return the minimum, maximum, of the arguments, and 1 - the argument, respectively.

The signals generated for each day, Buy, Sell, or Do Nothing, are post-processed using fuzzy logic. The trading rule, a fuzzy trading rule, returns values in the range 0 to 1. We use pre-determined membership functions, in this case, to determine what the meaning of this value is. The membership functions adopted were as follows:

$$Buy = Value < .33$$
$$DoNothing = .33 >= Value < .66$$
$$Sell = .66 >= Value$$

4.1 Data Preprocessing

The values of the indices changed substantially over the training and testing period. Before the trading rules were constructed, these values were normalised using a two phase preprocessing. Initially the daily values were transformed by dividing them by a 75 day lagged moving average. These transformed values are then normalised using linear scaling into the range 0 to 1. This procedure is a variant on that adopted by Allen and Karjalainen [Allen & Karjalainen, 1999] and Iba and Nikolaev [Iba & Nikolaev, 2000].

4.2 Selection of Fitness Function

A key decision in applying an EAP methodology to construct a technical trading system is to determine what fitness measure should be adopted. A simple fitness measure such as the profitability of the system both in and out of sample is inadequate as it fails to consider the risk associated with the developed trading system. The risk of the system can be estimated in a variety of ways. One possibility is to consider market risk, defined here as the risk of loss of funds due to a market movement. A measure of this risk is provided by the maximum drawdown (maximum cumulative loss) of the system during a training or test period. This measure of risk can be incorporated into the fitness function in a variety of formats including: (return / maximum drawdown) or return - 'x'(maximum drawdown), where 'x' is a pre-determined constant dependent on an investor's psychological risk profile. For a given rate of return, the system generating the lowest maximum drawdown is preferred.

This study incorporates drawdown in the fitness function by subtracting the maximum cumulative loss during the training period from the profit generated during that period. This is a conservative approach which will encourage the evolution of trading systems with good return to risk characteristics. This will provide a more stringent test of trading rule performance as high risk / high reward trading rules will be discriminated against.

5. Results

Thirty runs were performed on each of the three datasets using a population size of 500 individuals over 100 generations. Bit mutation at a probability of 0.01, and a variable-length one-point crossover probability of 0.9 was adopted. A comparison of the best individuals evolved for each dataset to the benchmark buy and hold strategy can be seen in Table 1.2, Table 1.3, and Table 1.4, for the FTSE, DAX and NIKKEI datasets respectively.

In the case of the FTSE and NIKKEI datasets the evolved rules produce a superior performance to the benchmark strategy, while performance over the DAX dataset is not as strong. The poorer performance for the DAX market can be attributed to over-fitting of the evolved rules to the training data. For each of the evolved trading rules the associated risk is less than that of the benchmark strategy as can be seen in the average daily investment figures reported.

Trading Period (Days)	**Buy & Hold** Profit (US$)	**Best-of-run** Profit(US$)	**Best-of-run** Avg. Daily Investment
Train (75 to 440)	3071	3156	3219
Test 1 (440 to 805)	5244	1607	1822
Test 2 (805 to 1170)	-1376	4710	3151
Test 3 (1170 to 1535)	1979	2387	6041
Test 4 (1535 to 1900)	1568	-173	3274
Test 5 (3196 to 3552)	3200	2221	3767
Total	13686	13908	

Table 1.2. A comparison of the buy and hold benchmark to the best evolved individual for the FTSE dataset.

Trading Period (Days)	**Buy & Hold** Profit (US$)	**Best-of-run** Profit(US$)	**Best-of-run** Avg. Daily Investment
Train (440 to 805)	3835	3648	7548
Test 1 (805 to 1170)	-41	-1057	8178
Test 2 (1170 to 1535)	3016	469	8562
Total	6831	3060	

Table 1.3. A comparison of the benchmark buy and hold strategy to the best evolved individual on the DAX datatset.

Trading Period (Days)	**Buy & Hold** Profit (US$)	**Best-of-run** Profit(US$)	**Best-of-run** Avg. Daily Investment
Train (75 to 440)	-6285	3227	9247
Test 1 (440 to 805)	59	-1115	7164
Test 2 (805 to 1170)	-3824	633	9192
Total	-10050	2745	

Table 1.4. A comparison of the benchmark buy and hold strategy to the best evolved individual on the NIKKEI datatset.

6. Discussion

In evaluating the performance of any market predictive system, a number of caveats must be borne in mind. Any trading model constructed and tested using historic data will tend to perform less well in a live environment than in a test period for a number of reasons. Live markets have attendant problems of delay in executing trades, illiquidity, interrupted / corrupted data and interrupted markets. The impact of these issues is to raise trading costs and consequently to reduce the profitability of trades generated by any system. An allowance for these costs ('slippage') has been included in this study but it is impossible to determine the scale of these costs ex-ante with complete accuracy. In addition

to these costs, it must be remembered that the market is competitive. As new computational technologies spread, opportunities to utilise these technologies to earn excess risk-adjusted profits are eroded. As a result of this technological 'arms-race', estimates of trading performance based on historical data may not be replicated in live trading as other market participants will apply similar technology. This study ignores impact of dividends. Although a buy-and-hold strategy will generate higher levels of dividend income than an active trading strategy, the precise impact of this factor is not determinable ex-ante. It is notable that the dividend yield on most stock exchanges has fallen sharply in recent years and that the potential impact of this factor has lessened.

7. Conclusions & Future Work

Grammatical Evolution has been shown to successfully generate trading rules with a performance superior to the benchmark buy and hold strategy on two of the three datasets analysed. In addition the risk involved with the adoption of the evolved trading rules is less than the benchmark.

The risk of the benchmark buy-and-hold portfolio exceeded that of the portfolio generated by the technical trading rules because, the benchmark buy and hold portfolio maintains a fully invested position at all times in the market, whereas the portfolio generated by the evolved technical trading system averaged a capital investment of \$3,546, \$8,096, and \$8,534 over the trading periods on the FTSE, DAX, and NIKKEI datasets respectively.

The results clearly show that there is much potential in this model and that there is notable scope for further research utilising GE in this problem domain. Our preliminary methodology has included a number of simplifications, for example, we only considered a small set of primitive technical indicators. The incorporation of additional technical indicators may further improve the performance of our approach. One factor that can have a large impact on the performance of GE is the choice of grammar. In this case study only one grammar was investigated, and thus trading rules conforming to one syntactical structure were evolved. Future work will involve the investigation of more complex syntactical structures.

A number of additional extensions of this work are left for future development. One extension would be to incorporate a more complex model of learning (forgetting). [Glassman, 1973] suggested that the "fallibility of memory" (p. 88) may represent a useful adaptive device when faced with a dynamic environment. At present in our model, all historic

data observations are given equal weighting which implicitly assumes model stationarity. By a suitable modification of the fitness function, whereby more recent data observations are assigned a higher weighting in the model construction process, model development could be biased towards more recent data [Refenes et al., 1997]. The weighting parameter could also be evolved as a component of the developed model.

Another avenue for exploration is the utility of stacked or multi-stage models [Zhang et al., 1997]. The construction of a single GE trading model, implicitly assumes that there is a dominant global error minimum and implicitly hopes that the constructed model approaches this minimum. Given the limitations of a problem domain in which input/output relationships are dynamic and where input data is incomplete and noisy, the error surface may not have this property. In such a topology there may be potential to improve predictive quality by building multiple models. To implement this approach, a series of GE models could be developed to produce trading signals, using non-homogeneous inputs. The predictions of the individual models could then be utilised as inputs to a second stage model which produces the final trading signal. This second-stage model could be developed using GE or alternatively could employ a different methodology such as neural networks.

References

Allen, F., Karjalainen, R. (1999) Using genetic algorithms to find technical trading rules. *Journal of Financial Economics*, **51**, pp. 245-271.

Bauer R. (1994). Genetic Algorithms and Investment Strategies, New York: John Wiley & Sons Inc.

Brock, W., Lakonishok, J. and LeBaron B. (1992). Simple Technical Trading Rules and the Stochastic Properties of Stock Returns, Journal of Finance, 47(5):1731-1764.

Brown, S., Goetzmann W. and Kumar A. (1998). The Dow Theory: William Peter Hamilton's Track Record Reconsidered, Journal of Finance, 53(4):1311-1333.

Chan, L. K. C., Jegadeesh, N. and Lakonishok, J. (1996). Momentum strategies, Journal of Finance, Vol. 51, No. 5, pp. 1681 - 1714.

Deboeck G. (1994). 'Using GAs to optimise a trading system', in Guido Deboeck (Ed.). Trading on the edge: neural, genetic and fuzzy systems for chaotic and financial markets, New York: John Wiley & Sons Inc.

Dissanaike, G. (1997). Do stock market investors overreact?, Journal of Business Finance & Accounting (UK), Vol. 24, No.1, pp. 27-50.

Glassman, R. (1973). Persistence and Loose Coupling in Living Systems, *Behavioral Science*, 18:83-98.

Iba H. and Nikolaev N. (2000). Genetic Programming Polynomial Models of Financial Data Series, In *Proc. of CEC 2000*, pp. 1459-1466, IEEE Press.

Koza, J. (1992). *Genetic Programming.* MIT Press.

Lewin, Benjamin. (2000). *Genes VII.* Oxford University Press.

Murphy, John J. (1999). Technical Analysis of the Financial Markets, New York: New York Institute of Finance.

Neely, C., Weller P. and Dittmar, R. (1997). Is technical analysis in the foreign exchange market profitable? A genetic programming approach, Journal of Financial and Quantitative Analysis, Vol. 32, No. 4, pp. 405 - 428.

O'Neill M. (2001). Automatic Programming in an Arbitrary Language: Evolving Programs with Grammatical Evolution. Ph.D. thesis, University of Limerick.

O'Neill M., Brabazon, A., Ryan C., & Collins J.J. (2001). Evolving Market Index Trading Rules Using Grammatical Evolution. *LNCS 2037, Applications of Evolutionary Computation: EvoWorkshops 2001*, pp. 343-352. Springer.

O'Neill M., Ryan C. (2001). Grammatical Evolution. *IEEE Trans. Evolutionary Computation.*

O'Sullivan J., Ryan C. (2002). An investigation into the use of different search strategies with Grammatical Evolution. In Proceedings of EuroGP'2002, In print.

Pring, M. (1991). Technical analysis explained: the successful investor's guide to spotting investment trends and turning points, Mc Graw-Hill.

Refenes, A.N., Bentz, Y., Bunn, D. W., Burgess, A.N. and Zapranis A.D. (1997). Financial time series modelling with discounted least squares backpropagation, *Neurocomputing*, 14:123-138.

Ryan C., Collins J.J., O'Neill M. (1998). Grammatical Evolution: Evolving Programs for an Arbitrary Language. *LNCS 1391, Proc. of EuroGP'98*, pp. 83-95. Springer-Verlag.

Zhang, J., Martin, E.B., Morris, A.J. and Kiparissides C. (1997). 'Inferential Estimation of Polymer Quality Using Stacked Neural Networks', *Computers and Chemical Engineering*, 21(Supplement):1025-1030.