An Investigation into the use of Intelligent Systems for Currency Trading Analysis

Submitted in partial fulfilment
of the requirements of the degree of

Bachelor of Science (Honours)

of Rhodes University

Jonathan James Millin

Grahamstown, South Africa
November 2008

Abstract

Investors use a number of technical trading tools to help them in their decision-making. This thesis aims to enhance this decision making process through the application of Genetic Algorithms (GAs) and Artificial Neural Networks (ANNs). The signals generated by technical trading tools are optimised for maximum profit through the use of GAs. The optimised signals are fed into a variety of fully connected feedforward ANNs, which combine these signals and output a single set of signals of whether to buy, hold or sell in the current market state. The different solutions produced are compared and contrasted, to determine the best ANN architecture for this type of signal amalgamation problem, and the optimal population size and mutation function for the GA.

The result is an autonomous trading system with intelligence. This system, as created in this thesis, has proven to be profitable based on data presented to it - which spans ten currencies over a five-year period. The profit margins are statistically significant when compared to un-optimised trading rules as suggested by literature. Further, the margins are statistically significantly more profitable than other no-risk investment strategies.

Acknowledgements

I would like to thank Michael Winn for his extensive help in understanding the Economics and Econometrics behind this system; and the friends who took the time to proofread this thesis.

My supervisor, Dr. Hannah Thinyane, for her input, guidance and most importantly her patience with me throughout the year.

I would also like to thank the Computer Science department of Rhodes University for the resources that made this research possible; this includes the departmental sponsors: Telkom SA, Comverse SA, OpenVoice, Stortech, Tellabs, Amatole, Mars Technologies, Bright Ideas Projects 39 and THRIP through the Telkom Centre of Excellence at Rhodes University.

Last but not least I would like to thank my parents: Roger and Wendy Howson for their sponsorship of my academic career and continued support in all aspects of my life.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Problem Statement

Although academics argue as to its effectiveness, technical analysis is used as a successful trading strategy by many foreign exchange traders [6]. While there are many technical tools available, few systems exist which intelligently link analysis from a number of tools, providing a single specific outcome. Even fewer are profitable, and all come under severe criticism [49]. The aim of this research is to build a system which will generate significantly profitable buy, hold and sell signals for the foreign exchange market through the use of technical analysis, while taking into account all previous criticisms. Artificial Intelligence (AI) provides the optimisation and interpretation techniques to improve and combine the signals generated by technical trading rules.

1.2 Background

> "The goal of *artificial intelligence* (AI) is the development of paradigms or algorithms that require machines to perform cognitive tasks, at which humans are currently better" [21].

The AI techniques investigated in this thesis are those of Evolutionary Computation (EC) and Artificial Neural Networks (ANNs). EC has been successfully employed as an optimisation tool in many different fields [3], including the field of technical analysis

[48]. ANNs are proven universal approximators [11], thus if a function or mechanical thought process exists, it can be approximated by an ANN. These two AI techniques will be applied to optimise and amalgamate signals generated by technical trading tools, to provide technical analysis of the foreign exchange market.

Technical analysis is a collection of algorithms and mechanical rules which attempt to aid investors in forecast future market movements, using only historic data. Academia however, has found mixed results from technical analysis, arguing that such attempts find no sufficient basis for profitability [14].

A great deal of research into the efficacy of technical trading rules has been recently undertaken, of which many studies find evidence of profitability [49]. These results discount the academic belief that asset price behaviour is determined solely by market fundamentals, supporting the arguments of practicing investors that asset price information can be analysed and exploited to obtain profit [46].

Several previous attempts have been made to aggregate the information generated by technical trading tools in order to produce more powerful prediction tools. These systems have however, been scrutinised for either data snooping, or overlooking the transaction costs involved in trading [49].

This thesis aims to build such a system, taking criticisms of previous systems into account. The system will analyse data using technical analysis, to produce a set of buy, hold and sell signals for each tool used. The signals will then be combined and optimised with AI techniques. It is the aim of the system to produce profit to a statistically significant level when compared to a no-risk investment strategy.

1.3 Specific Objectives

In order to develop the proposed system, a number specific goals need to be achieved. The first requirement is that technical trading tools need to be optimisable. If they are optimisable, then it will be possible to tailor the parameters of these tools to best suit the given data set. The second requirement is exploring the effectiveness of Genetic Algorithms (GAs) as optimisers in the context of technical analysis. To further improve the GA, optimal parameters need to be found. Once the GA is able to optimise the trading rules effectively, it is necessary to intelligently combine the generated signals in the most

profitable manner. To find the most profitable signal combiners, different architectures for the ANN need to be explored and exploited.

Once a profitable intelligent trading tool has been developed, it needs to be compared to the un-optimised trading rules to identify whether the addition of intelligence provides a statistically significant increase in the profitability of the system.

For the system to be deemed a success, it needs to generate profits greater than a no-risk investment strategy, and greater than the un-optimised trading rules - to a statistically significant level.

1.4 Scope of Research

As this thesis is primarily concerned with the intelligence aspect of an autonomous trading system, different technical trading tools will not be compared and contrasted; rather a few are selected and implemented, based on recommendations from literature. Therefore, the resultant system is not intended to be an autonomous trader, but rather to test the feasibility of such a system. The resulting system applies AI to technical analysis, and assess the significance of its profitability.

Consequently, the system will only analyse asset price information. No other information about the currencies or economies of the respective countries is analysed.

Further, it is out of the scope of this thesis to investigate different types of ANNs, different methods of training them, and alternative optimisation techniques.

1.5 Contributions

This thesis finds that it is possible to create a profitable autonomous trading system. This is accomplished by creating a technical analysis tool that generates signals which are profitable to statistically significant levels.

In creating this intelligent trading tool, different architectures of fully connected feedforward neural networks are examined, providing insight into the problems of overtraining ANNs.

This project also investigates relevant parameters for GAs, to identify the most efficient method of mutating the chromosomes as well as the optimal population size.

1.6 Thesis Structure

The remainder of this thesis is organised as follows: Chapter 2 introduces important concepts and work relating to this study. Chapter 3 documents the design and implementation of the developed system. Chapter 4 analyses this implementation, providing insight into the optimal architecture of the system. Chapter 5 presents the results which were discovered from the optimised system, with particular investigation into its performance. Finally, the thesis is concluded in Chapter 6.

Chapter 2

Introduction to Concepts and Related Work

The first section of this chapter investigates the history of *Evolutionary Computation* (EC), different methods of EC, and how they apply to this work. This is extended in the next section which describes *Technical Analysis* (the use of technical trading tools). After which, the same investigation and comparisons are made for Artificial Neural Networks. Previous systems of technical analysis are discussed, and particular mention is made of their criticisms. Finally the review of literature and related work is concluded by summarising the findings of this chapter.

2.1 Evolutionary Computing

The *Darwinian theory of evolution* explains the adaptation of species to their environment by means of *natural selection*. This system favours those species which are best adapted to their environmental conditions [12]. Darwin also recognized small, apparently random variations in the characteristics of parents and their offspring, termed *mutations*. If these mutated characteristics cause the offspring to be better suited to the current environment, they prevail through the selection process; if the resulting offspring are less suited to the environment, they perish [12]. In the evolutionary framework, the individuals in the population who survive are said to be fitter than those who perish, as they are better suited to their current environment. Thus the fitness of an individual is measured indirectly by its growth rate in comparison to others (its propensity to survive and reproduce) [3].

By applying natural selection and genetic variation, it becomes possible to evolve a population of candidate solutions to almost any problem; so long as that problem is able to represent the fitness of that individual. Together, evolution strategies, evolutionary programming (which later evolved into genetic programming) and genetic algorithms form the backbone of the field of evolutionary computation [40]. These will be further discussed in Section 2.1.3, and summarised with regards to application in Section 2.1.4.

2.1.1 History of Evolutionary Computing

Throughout the 1950's and 1960's, several computer scientists independently studied evolutionary systems with the intention of optimising engineering problems. There were also a number of evolutionary biologists who used computers to simulate evolution through controlled experiments during the same time period [40].

In the 1960's and 1970's Ingo Rechenberg [53, 54] introduced *Evolution Strategies* as a method of optimising real-valued parameters for airfoils. This was done by adding a normally distributed random value to each vector component of the real-valued parameters, and selecting the fittest of the parent and child [53, 54].

Fogel, Owens and Walsh [16] developed a method of automatic-programming they called *Evolutionary Programming* in 1966 by representing possible solutions as finite-state machines which evolved by randomly mutating state-transition diagrams and selecting the fittest ones.

Genetic Algorithms (GAs) were invented by John Holland in the 1960's [40]. They were further developed at the University of Michigan in the 1960's and 1970's by Holland, his students and his colleagues [23]. Holland was primarily interested in the study of adaptation and evolution as it occurs in nature, and the ways in which these features might be imported into computer systems rather than the solving of specific problems [40]. Holland presented the GA as an abstraction of biological evolution, and introduced the notion of a population-based algorithm as well as the genetic operators of crossover and inversion [23]. Before Holland's introduction of a population-based algorithm, evolutionary computation had only ever used a population of two individuals: one parent and one offspring, where the offspring was a mutated version of the parent [40]. It is now common to see large populations of individuals in all forms of evolutionary computing [40].

D. B. Fogel extended the works of his father in the realm of Evolutionary Programming in the 1980's [40]. In 1985, Cramer was the first to use the principles of GAs in conjunction

with the principles of Evolutionary Programming to breed solutions as simple sequential programs [10]. Fujiki and Dickinson, [18] among others, worked on similar problems in the same time period [40].

In the early 1990's, John Koza [31] spurred a resurgence in automatic-programming by his works on evolving recursive Lisp programs using GAs and Evolutionary Programming [31], via what he termed, *Genetic Programming* [40].

2.1.2 Notation for Evolutionary Computing

Notation for different methods of Evolutionary Computing (EC) can be fully characterized by the term (A)-B where A represents the selection and recombination methods, and B represents the method of EC: ES for Evolutionary Strategies, GP for Genetic Programming, and GA for Genetic Algorithms. As ES was the first method to be used, the following notations will use ES to describe them, but the same notation applies to the other EC methods.

The first mechanisms for EC were very basic, with a population size of two: the parent, and an offspring which is a mutated and/or recombined version of the parent. The fittest of the two became the parent of the next generation, and the weakest was disregarded. This mechanism is fully characterized by the term (1+1)-ES [3].

$(\mu + 1)$-ES was the first method of EC having a population greater than two individuals, in which μ parent individuals recombine to form one offspring. The offspring is then mutated, and replaces the least fit parent individual if it is fitter.

$(\mu + \lambda)$-ES selects the best μ individuals from the union of the parents and offspring to form the parent population for the next generation. The parent population is then recombined to produce λ offspring which are then mutated. (μ, λ)-ES is very similar, and differs only in the selection procedure, where the best μ individuals are selected only from the offspring population. Note $\lambda > \mu$ is necessary. Thus the selection process is represented in one of two ways: selection takes place from the offspring population only (comma notation), or selection takes place from the offspring and parent populations together (plus notation) [3].

2.1.3 Different Methods of Evolutionary Computing

As was previously introduced, there are three main branches of EC: Evolution Strategies, Genetic Programming (which replaced Evolutionary Programming) and Genetic Algorithms. This section describes the different methods in detail.

2.1.3.1 Evolution Strategies

Evolution Strategies (ES) were created in the 1960's and 1970's by Ingo Rechenberg and co-workers Bienert and Schwefel [3]. ES was originally used to optimise shapes for minimal drag bodies in a wind tunnel at the Technical University of Berlin in Germany [3]. The resulting ES was a very basic (1+1)-ES [3].

This was followed by $(\mu + 1)$-ES, which was inspired by Holland's multimembered GAs. $(\mu + 1)$-ES was never widely used, but provided the foundation for the more common $(\mu + \lambda)$-ES and (μ, λ)-ES methods [3].

Currently (μ, λ)-ES characterises the latest in ES research as it tends to converge faster in a population greater than ten, and the optimal standard deviation can be calculated numerically [3].

In ES, the parameters are represented as real-valued vectors. The evolutionary changes between populations are due to mutation alone. Each element is mutated by adding a random number taken from a Gaussian distribution with mean 0. The specialisation of ES is that the step size of the mutation is self-adapting. z values are drawn from a normal distribution $N(\xi, \sigma)$, where the mean ξ is set to 0, and the variance σ (which is also termed the mutation step size) is varied dynamically by the $\frac{1}{5}$ success rule. This rule resets σ after every k iterations according to Equation 2.1.

$$\begin{cases} \sigma = \sigma/c & if\, p_s > \frac{1}{5} \\ \sigma = \sigma * c & if\, p_s < \frac{1}{5} \\ \sigma = \sigma & if\, p_s = \frac{1}{5} \end{cases} \tag{2.1}$$

In 2.1, p_s is the percentage of successful mutations; for example those which have improved the fitness, where $0.8 < c < 1$. The effects of the step size on the Gaussian distribution can be seen in Figure 2.1.

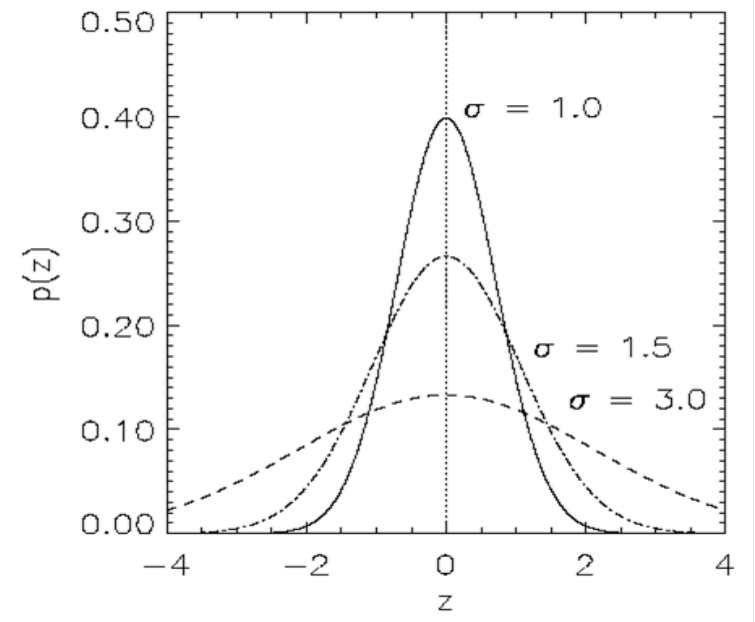


Figure 2.1: The effects of the change in step size on the Gaussian distribution.

It should be noted that σ is appended to the end of the chromosome, and thus co-evolves with the solution. σ is first mutated into σ' before it is used to mutate x into x' where $x' = N(0, \sigma')$. x' is deemed good if $f(x')$ returns a better result than $f(x)$, and σ' is deemed good if the x' it created is deemed good. Thus σ needs to be mutated before x can be mutated. σ' can be calculated-as in Equation 2.2, where τ is the learning rate, and σ' is bound below by some ε_0. There are alternative ways of mutating σ - all of which involve some method of multiplying σ by the exponential of some scaled normal distribution. However many normal distributions and learning rates can be used - the exploration of which is out of the scope of this work.

$$\sigma' = \sigma * e^{(\tau * N(0,1))} \tag{2.2}$$

EC uses a uniform random parent selection. This means that each parent chromosome has an equal chance of creating an offspring. Parents are recombined by either choosing one of the parent values, or by averaging the two values. This is determined by a randomly generated number.

Survival occurs in an elitist manner. The child population is reduced in size by keeping only the fittest individuals.

In general, Evolution Strategies represent their parameters as real values which are constrained by arbitrary inequalities. Individuals are created by recombining parents dis-

cretely to form offspring, and mutating these offspring by adding normally distributed random values to the parameters [3].

2.1.3.2 Genetic Programming

Evolutionary Programming was first developed by L. J. Fogel in 1966, which remained dormant until his son D. B. Fogel extended it in the 1980's [3]. Genetic algorithms and Evolutionary Programming were used in conjunction by Cramer in 1985 [10], and others during a similar time period [3]. Koza further developed these techniques and developed Genetic Programming [40].

Koza claims that the solving of problems can be reformulated as: the search for highly fit individual computer programs in the space of all possible computer programs; and that Genetic Programming (GP) provides the means to do so [31].

In GP, Neo-Darwinian principles are applied to the breeding of large populations of computer programs to find the fittest solution to a given problem [31]. The fitness is measured by running the generated program over a number of test cases and applying some objective function to the results of the generated program; the higher the output, the fitter the solution [31].

The programs which are evolved are traditionally represented as tree structures. These tree structures are connected directed acyclic graphs where each vertex (node) has an indegree of 1 or 0. A node is either a *function* or a *terminal*. A function node has an outdegree of 1 or more and has a path to a terminal node. Terminal nodes have an outdegree of zero. Function nodes are representative of operator functions, and terminal nodes are representative of operands. As each subtree can be viewed as a tree, these tree structures are highly recursive, and thus they are easily evaluated in a recursive manner [31]. This can be seen in Equation 2.3 and Figure 2.2.

$$\frac{A+3}{B*6} * A^2 \rightarrow (*(/(+A\,3)(*B\,6))(*A\,A)) \tag{2.3}$$

GP uses genetic operators of *reproduction*, *recombination* and *mutation* to breed solutions.

Reproduction involves selecting a program from the population of programs proportionate to its *fitness*. This fitness is usually measured by running each program over a number of *fitness cases* to get a more representative fitness of the solution [31].

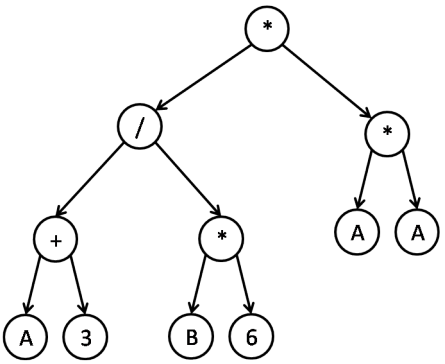


Figure 2.2: An example of a program represented by a tree structure.

The **recombination** of these computer programs is achieved by swapping randomly selected subtrees between parent programs. This often results in a large variance in the length of the programs [31].

Mutation is achieved either by changing a randomly selected terminal to a different terminal, or by changing a randomly selected function to a different function. The set of possible terminals and functions is created before the GP executes [31].

In general, the Genetic Programming paradigm breeds computer programs to solve problems through the recombination and mutation of trees of functions and terminals. This makes them well suited to machine learning [31].

2.1.3.3 Genetic Algorithms

Genetic Algorithms (GA) were developed by Holland, his colleagues and his students in the 1960's and 1970's at the University of Michigan [40]. Holland's intentions were to explain the adaptive process of natural systems, and to develop artificial systems which use these adaptive processes to solve engineering problems [3].

These systems are generally used to optimise objective functions. The underlying parameters for these functions are encoded as bitstrings. Following biological terminology, these bitstrings are called *chromosomes* [19]. A *population* is made up of a number of individuals, where each individual is a chromosome.

Genetic operators of *reproduction*, *crossover* and *mutation* are applied to successive *parent* populations, to create new *child* populations. This results in two parents being selected and recombined to form two children. These children are then mutated to provide genetic

variation. This process is repeated until a suitable fitness is reached, or a maximum number of epochs is reached [40].

The suggested manipulation of these individual chromosomes at the string level exploits similarities among high-performing chromosomes. This performance is termed the *fitness* of a chromosome, and the better it performs, the fitter it is. [19].

The aforementioned chromosomes are usually bitstrings of binary 1's and 0's which represent real valued parameters. They can however, be represented in a number of different ways. One such method is to comprise chromosomes as real-valued vectors. The formulation of a bitstring from a number of parameters is called *encoding* [40].

As binary encoding is the most popular method of **encoding**, this method will be explained. When encoding a parameter into an n-bit chromosome, one is able to represent $2^n - 1$ different alternative values for that parameter; this is termed *resolution*. If all possible values for the parameter are between an upper bound (UB) and lower bound (LB), then there are $2^n - 1$ steps between LB and UB [40]. Hence each step has a value, calculated by Equation 2.4.

$$\frac{UB - LB}{(2^n - 1)} \tag{2.4}$$

Thus to calculate the actual value of an n-bit bitstring, first find the decimal representation of the bitstring (x), then take that amount steps away from the lower bound [40]. Hence the value of that parameter is represented by Equation 2.5.

$$LB + x\frac{UB - LB}{(2^n - 1)} \tag{2.5}$$

Reproduction is the process by which the two parent chromosomes are selected from the population, according to their objective function values, and copied. Biologists call this objective function the *fitness function*. The chromosome is decoded into its parameter values, and evaluated by the fitness function, yielding a *fitness value*. By copying strings according to their fitness values, a string with a higher value has a higher probability of contributing one or more offspring to the next generation. This process mimics natural selection, whereby a creature's fitness function refers to its ability to survive and reproduce in its current environment [19].

There are many methods of selecting the fittest individuals from the population:

- *Stochastic Universal Sampling* is Holland's original method of selection. In this situation, the probability that an individual will be selected from the population is fitness-proportionate. This means that the individual will be selected with the probability of its fitness relative to the average fitness of the population. One explanation of this is: the algorithms lay out a line in which each parent corresponds to a section of the line, where its length is proportional to its scaled fitness value. The algorithm moves along the line in steps of equal size. At each step, the algorithm allocates a parent from the section it lands on. The step size is a random number generated by a normal distribution, and is less than a maximum step size. [40].

- *Roulette Wheel Sampling* is a computer-friendly implementation of the Stochastic Universal method. Each individual is allocated a percentage fitness of the total population's fitness. A random number between 0 and 1 is then generated, and all the elements of the population are iterated through, adding their respective fitnesses together until the random number is reached. The individual whose fitness is the last to be added is then selected. This is representative of the expected selection probabilities [40].

- *Sigma Scaling* includes the population mean and standard deviation, as well as the fitness of the individual and the population when selecting individuals. This ensures that the population does not converge too quickly [40].

- *Elitism* retains the same number of best individuals at each generation, to ensure that best solutions are not randomly lost [40].

- *Boltzmann Selection* is similar to Sigma Scaling, but it varies the selection pressure. This allows for a high selection variance at the beginning of the breeding session, and lowers it as generations progress. This reduces premature convergence [40].

- *Rank Selection* ranks individuals based on their fitness, and then selects them based on their rank, instead of their actual fitness. This prevents the GA from converging too quickly [40].

- *Tournament Selection* pits two randomly selected individuals from the population against each other. The fittest of the two will be selected as a parent for the next generation, but both will be returned to the population to be selected again [40].

Crossover is the process by which two newly reproduced (parent) chromosomes are combined to form offspring in the next generation. This is achieved by randomly swapping

corresponding elements from both parent chromosomes to form two child chromosomes [19].

- *Single point crossover* is when all elements before a randomly selected point on the parent chromosome are combined with all the elements after the crossover point from the other parent, and *vice versa*. This method has its shortcomings, as not all possible combinations can be produced this way [40].
- *Two point crossover* is when the chromosomes cross over at two points along the bitstring, and effectively swap a chunk of information, which closely mimics mitosis (the recombination of chromosomes in sex cells). This allows for considerably more combinations than single point crossover, but still does not allow for every possible combination to occur [40].
- Some practitioners are advocates for *parametrised uniform crossover* or *scattered crossover*, where a probability is calculated for each bit to swap over. However, having too many crossover points can be disruptive when trying to find an optimal solution [40].
- It is also possible to create *crossover hotspots* where crossover is far more likely to occur at certain hotspots than at any other point, which is useful to retain dependencies [40].

Mutation is necessary because, even though reproduction and crossover effectively search and recombine extant notions, they occasionally lose potentially useful genetic material. In artificial systems, mutation prevents such irrecoverable losses from occurring. This is achieved by randomly changing a value in the child chromosome [19].

If a real-valued vector is used as the encoding mechanism, then a normally distributed value is added to a randomly selected element in the vector. The constraints of the normal distribution responsible for generating these mutation values may be dynamic, and thus may fluctuate depending on parameters such as the variance of the population, or the number of generations elapsed.

There are many different methods of mutation:

- *Uniform Mutation* is a two-step process. First, the algorithm selects a fraction of the vector entries of an individual for mutation, where each entry has a probability

of being mutated. *Mutation rates* are set to control how much mutation occurs. If a randomly generated number is within the mutation rate, then that chromosome will be mutated. In the second step, the algorithm adds a random number from a normal distribution about 0 to each selected entry.

- *Dynamic Gaussian Mutation* is different to uniform mutation in that there is no mutation rate. Every individual is mutated after crossover. The impact of the mutation however, is dynamic. The algorithm mutates each element of each parent by adding a random number taken from a Gaussian distribution with mean 0. The standard deviation of this distribution is determined by the parameters *Scale* and *Shrink*. *Scale* is inversely proportional to the standard deviation of the parent vector. This ensures that when a population becomes very similar, the mutation rate increases. *Shrink* reduces the mutation rate for every completed generation, producing a more mature solution with increased stability.

- *Adaptive Feasible Mutation* adds a randomly generated number to each element in the child population. The direction (positive or negative) of the random number is adaptive with respect to the last successful or unsuccessful generation. The feasible region is bounded by the relative constraints and inequality constraints.

In general, Genetic Algorithms represent their parameters as strings of binary values which are constrained by the encoding mechanism. Individuals are created by recombining parents using crossover functions to form offspring. The children are subsequently mutated by either randomly inverting bits in the binary string, or adding randomly generated values to elements in the real-valued vector [3].

GAs as a means of parameter optimisation De Jong, [13], finds that genetic algorithms out-performed local search algorithms, especially on multimodal, noisy functions. Subsequently, genetic algorithms have been used successfully in a wide variety of optimisation tasks, including numerical and combinatorial optimisation problems, as well as circuit layout and job-shop scheduling [40]. GAs have also been successfully used in optimisation of technical trading tools by Allen and Karjalainen [1], Neely, Weller and Dittmar [43] and Shazly and Shazly [59], to name a few.

2.1.4 Summary and Applications

As seen in Section 2.1.3.2, Genetic Programming breeds a program as a solution to a well defined problem. The primary concern in this thesis, with regards to evolutionary computation, is that of parameter optimisation of technical trading tools. Thus genetic programming is not an optimal method for implementation, and so will no longer be discussed.

According to Bäck [3], the encoding mechanism used in genetic algorithms seems to extend their range of possible applications beyond the capabilities of evolution strategies. Bäck [3] also states that Genetic Algorithms have been researched in far greater depth than evolution strategies, and that genetic algorithms are a better representation of natural evolution.

Because of this, and the previous applications of GAs in technical trading, GAs are deemed an appropriate method of parameter optimisation and will be used as the primary method of function optimisation in this work.

As the selection, crossover and mutation operators are all problem-specific, they will all have to be examined when determining the best implementation of GAs.

2.2 Technical Analysis

Technical trading tools are an example of multi-parametered functions which can be optimised by Genetic Algorithms. These, together with Fundamental Analysis, are amongst the two broad categories of tools used to guide practicing investors on when to buy, hold or sell their assets. While Fundamental Analysis uses information about markets (such as price-earnings ratios and interest rates) to gain an understanding of a market, Technical Analysis disregards all of this information, only reviewing trading data: such as price levels and volumes [45].

One can use signals generated by fundamental analysis however, technical analysis has orders-of-magnitude more data available. This makes technical analysis a much more suitable test-bed, as the GAs and ANN will have more data on which to train.

In essence, Technical Analysis attempts to forecast the movements in future markets, using historic data as a basis. According to Brock, Lakonishok and Lebaron [6]:

> "Technical analysis is considered by many to be the original form of investment analysis, dating back to the 1800s. It came into widespread use before the period of extensive and fully disclosed financial information, which in turn enabled the practice of fundamental analysis to develop. In the United States, the use of trading rules to detect patterns in stock prices is probably as old as the stock market itself."

Technical analysis is still widely employed in the market by practicing investors, who believe that profits exist in the short-run, and that these are exploitable by technical strategies [46].

In spite of this, academics have treated Technical Analysis with skepticism. According to Park and Irwin [49], this skepticism can be linked to the *Efficient Market Hypothesis* [14], which postulates that attempting to make profits by exploiting currently available information is futile, as any possible opportunity would already have been exploited due to the efficiency of the market. Early studies of technical analysis in the stock market from Fama and Blume (1966) [15], Van Horne and Parker (1967, 1968) [64, 65] and Jensen and Bennington (1970) [28], amongst others, support this statement, with the conclusion that prices fluctuate randomly and hence cannot be predicted.

More recent literature however, opposes these conclusions. In a review of the profitability of technical analysis, Park and Irwin [49] find that, of 95 modern studies: 56 find positive results, 20 obtained negative results, and 19 indicated mixed results.

The technical trading system used to find these positive results consists of a set of trading rules. These rules generate long-term and/or short-term trading signals, according to various parameter values. Popular technical trading tools include trending strategies, filters and oscillators. Practitioners generally use them in a practice known as charting; plotting them with the actual price levels, to determine market trends. However, the information is equally, if not more valuable if employed as raw signals [49].

2.2.1 Different Methods of Technical Analysis

There are many different tools which fall under the umbrella of technical analysis, but this is not the primary concern of this study, and hence only five rules have been investigated in this thesis. They were chosen because of their proven usefulness in papers such as Park and Irwin [49] and Sweeney [61]. The reasoning behind, and explanation of, each tool is discussed below.

2.2.1.1 Filter Rules

Filter rules encourage a trader to buy when an asset value rises a certain percentage above a previous local high, and sell if the price declines a certain percentage below a local low [15].

These were first introduced by Alexander in 1961, who reported returns significantly higher than a simple buy and hold strategy. Although this was refuted by Fama and Blume [15], it has been subsequently found to deliver significant profitability by both Sweeney [61] and Levich and Thomas [32].

2.2.1.2 Trending

Trending generates trading signals at the intersection of a short- and a long-run moving average. A buy is indicated when a the short-run moving average intersects the long-run moving average from below, indicating that the asset value is trending upwards. The converse is true for sell signals [62].

Trending has also seen mixed success when tested empirically, but Levich and Thomas [32], Pukthuanthong-Le, [**?**] and Okunev and White [46] find that trending is profitable to a significant level.

2.2.1.3 Relative Strength Index (RSI)

The Relative Strength Index (RSI) indicates the Relative Strength of an asset as a value between 0 and 100. If the RSI value is low, it indicates that an asset has been over-sold, and thus it is a good time to buy the asset. If the RSI value is high, it indicates that an asset has been over-bought, and thus it is a good time to sell the asset [33]. Levy [33] finds success in using RSI to predict opportune points to buy and sell in the stock market. This can be seen in Figure 2.3.

2.2.1.4 Moving Average Convergence / Divergence (MACD)

The Moving Average Convergence / Divergence (MACD) shows the difference between a fast and slow exponential moving average (EMA) of an asset's closing price information. Two moving averages are compared with one another, to produce the MACD. The MACD

Figure 2.3: An example of the RSI rule. Used with permission [47]

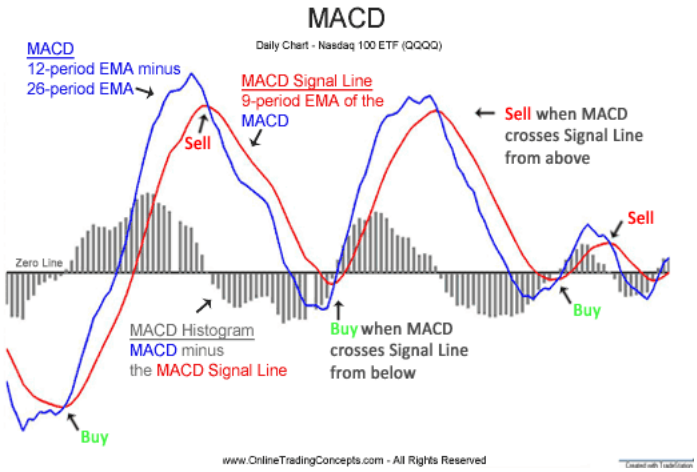


Figure 2.4: An example of the MACD rule. Used with permission [47]

is then compared to a *signal* line, which is an EMA of the MACD. Once the two are compared, the result is treated in a similar fashion to that of trending. A buy is signaled when the MACD line intersects the signal line from below, and a sell is signaled when the MACD line intersects the signal line from above [50]. This can be seen in Figure 2.4. Little empirical research has been done on the efficacy of this rule, but it is has been suggested as a worthwhile technical trading rule [49].

2.2.1.5 Momentum (MOM)

Momentum (MOM) shows the momentum of the movements of an asset price. This is calculated by looking at the change in asset price from the current day against that of a few days prior. If the MOM returns a negative value, this means that the asset price is

Figure 2.5: An example of the MOM rule. Used with permission [47]

downward trending, which is signaled as a sell; whereas if the MOM is positive, the asset price has an upwards trend and is signaled as a buy [50]. This can be seen in Figure 2.5. Momentum was found to be a profitable indicator by Jegadeesh, and Titman [27].

2.2.2 Summary and Application

Although technical trading is deemed dubious by some, the above rules have been shown to have some success. These rules are also mechanical in nature, and hence can be programmed using a modern computer programming language.

2.3 Artificial Neural Networks

Nauck, Klawonn, Kruse and Klawonn [42] define Artificial Neural Networks (ANNs) as *Universal Approximators,* as they are able to approximate any given continuous mapping from inputs to outputs. This is accomplished by using connectionist models which make use of some of the known and/or expected organizing principles of the human brain [42].

These nonlinear mapping systems are comprised of a number of independent and very simple processors which mimic a biological neuron, and are thus called *neurons.* These neurons are networked together, and communicate via weighted connections called *synaptic weights* [42]. Although far simpler than the inner workings of a human brain, they can simulate complex behaviours when massive systems of these simple units are linked

in appropriate ways [55]. The approximating function of ANNs is achieved by *training* the ANN. These ANNs can learn in one of two major learning paradigms: learning *with* a teacher, and learning *without* a teacher [21]. These will be further discussed in Section 2.3.3.

The end result is an information driven adaptive processing system which can learn from experience and/or generalise from previous examples. This has massive application in automation problems including adaptive control, optimisation, medical diagnosis, and decision-making; as well as information- and signal-processing, including speech processing and pattern recognition [21, 42]. These ANNs out-perform traditional rule-based artificial intelligence approaches, but are trivial when compared with even the simplest biological system [26].

The remainder of this section will provide a brief history of ANNs, followed by an examination of the different architectures for ANNs and how to train them. Finally, a summary of the following information will be presented with mention of the application of ANNs in relation to this work.

2.3.1 History of Artificial Neural Networks

Neural Networks as they are known today, began in 1943 with the pioneering works of McCulloch and Pitts [21]. In their paper *A logical calculus of the ideas immanent in nervous activity*, McCulloch and Pitts unite the fields of neurophysiology and mathematical logic by describing a logical calculus of neural networks. They assumed that a neuron followed an "all-or-none" law. With this, McCulloch and Pitts showed that by properly connecting a sufficient number of these simple units which could operate synchronously, it would (in principle) compute any computable function [36]. With this result, it is generally agreed that the disciplines of neural networks and artificial intelligence were born [21].

In 1949, Hebb published a book *The Organization of Behavior*, in which he explicitly stated for the first time a physiological learning rule for *synaptic modification* [22]. Following this in 1958, Rosenblatt introduced a novel method of supervised learning in his works on the *perceptron*; the culmination of which was the so-called *perceptron convergence theorem [56].*

In 1960, Widrow and Hoff introduced the *Least Mean-Square* (LMS) *algorithm*, and used it to formulate the *Adaline* (adaptive linear element), which differs from the perceptron

in its training procedure. *Madaline* (Multiple-Adaline) was one of the earliest trainable multilayered neural networks, the structure of which was proposed by Widrow and his students in 1962 [21].

In 1961, Minsky wrote a paper on AI entitled: *Steps Towards Artificial Intelligence [38]* which contains a large section on what is now termed Neural Networks. Minsky later wrote a book: *Computation: Finite and Infinite Machines*, which put the works of McCulloch and Pitts in the context of automata theory and theory of computation [21].

It seemed, during the classical period of the perceptron (the 1960s), that perceptrons could compute anything. This was until 1969, when Minsky and Papert published a book which demonstrated the fundamental limitations of single-layer perceptrons, using mathematics [39]. They then stated that there was no reason to assume similar problems could be overcome in the multilayer versions [38]. This, coupled with the technological constraints, diminished interest in neural networks until the 1980s [21].

One important emergence from the 1970s was that of *self-organizing maps* using competitive learning [21]. In 1976 Willshaw and von der Malsburg published the first paper on self-organizing maps, with motivation drawn from topologically ordered maps in the brain [66].

The controversial paper by Hopfield in 1982, along with the two-volume book by Rumelhart and McClelland, in 1986, were the two most influential publications - and were responsible for the resurgence of interest in neural networks in the 1980s [21]. In his paper, Hopfield made explicit the principle of storing information in dynamically stable networks. He also showed insight into the spin-glass model, by examining recurrent networks with symmetric connections, which guaranteed their convergence to a stable condition [24]. The two-volume book by Rumelhart and McClelland made mention of the *back-propagation algorithm* [58], developed by Rumelhart, Hinton and Williams in that same year [57]. The book has been a major influence in the use of the back-propagation algorithm, which is currently the most popular learning algorithm for training multi-layer perceptrons [21].

Ackley, Hinton and Sjenowski developed the first successful realisation of a multi-layer neural network in 1985, known as the *Boltzmann machine*. This was based on the idea of *simulated annealing*, developed by Kirkpatrick, Gelatt and Vechi in 1983 [21].

1988 saw the development of a new principle for self-organisation in a perceptual network by Linsker, in which he formulated the *maximum mutual information* (Infomax) principle

[34]; as well as a new procedure for the design of layered feedforward networks, in which *radial basis functions* (RBF) were used c[7]. 1988 also saw the seminal proof by Cybenko [11] that multilayered perceptrons are universal approximators. This meant that any underlying process or function could be approximated by a multilayered perceptron to an arbitrary accuracy. In the early 1990s Vapnik and colleagues invented *support vector machines*: a computationally powerful class of supervised learning networks for pattern recognition, regression and density estimation problems [21].

2.3.2 Network Architectures

Although the structure of a neural network is closely linked to the learning algorithm used to train the network [21], there are still many different architectures which are worth discussing. This section will go into brief detail on a number of popular network architectures, leaving the learning algorithms to the following section.

The Perceptron is the most simplest of neural network. It is used for the classification of linearly separable patterns. It consists of a single neuron with an adjustable synaptic weight and bias. The perceptron convergence theorem introduced by Rosenblatt [56] proves that if the weights and bias of a perceptron are systematically altered, they can classify a linearly separate pattern into two classes.

Single Layer Perceptrons extend the perceptron by introducing a layer of perceptrons, which is able to separate patterns into a number of classes. These are based on the input pattens and the number of perceptrons in the layer. The pattern must still be linearly separable for the perceptrons to work [21].

Multilayer Perceptrons, also called **multilayer feedforward networks**, generalise the single layer perceptron discussed above. They are comprised of a set of sensory units which constitute the *input layer*, one or more *hidden layers*, and an *output layer*. Inputs patterns are fed into the input layer, and propagate through the network on a layer-by-layer basis. They have been successfully applied to a number of problems by training them in a supervised manner with the back-propagation algorithm, which will be discussed in Chapter 2.3.3.1 [21].

Radial-Basis Function Networks make use of *radial-basis functions* (RBF) to separate data instead of the hyper-planes used by perceptrons. Thus they are very good at approximating data where clusters of points are found. Their structure consists of an

input layer, a hidden layer with a non-linear RBF activation function, and a linear output layer. They also differ from multilayer perceptrons in that they may only have one hidden layer. The non-linear RBF applies a transformation from the input space to the hidden space. In most applications the hidden space is of high dimensionality. RBF networks are also universal approximators. They may be trained by backpropagation, but are also able to train themselves in an unsupervised manner through the use of self-organising maps [21].

Support Vector Machines are linear machines which approximate the implementation of the *method of structural risk minimisation*. Their unique attribute is that they provide a good generalisation performance on pattern classification problems even though it does not incorporate problem-domain knowledge. The algorithm extracts a small portion of data from the input space which provides *support vectors*. The hyper-plane which classifies those sets maximises the distance between the support vectors, producing an *optimal hyper-plane*. The optimal hyper-plane is generally found in a *feature space* of higher dimensionality than the original data set which can be found using *kernel* mapping. Support vector machines may be polynomial learning machines, RBF networks, or two-layer perceptrons in special cases [21].

Committee machines use the engineering principle of divide and conquer, in which the knowledge acquired by many experts is fused together to arrive at an overall decision. The final decision is supposedly superior to that attainable by any one of them acting alone. This combination of experts is said to constitute a committee machine. The structure of a committee machine is divided into two categories: *static structures*, and *dynamic structures* [21].

In **static structured committee machines**, the input is not involved in the combination. The *ensemble averaging* method linearly combines the outputs of different predictors to produce an overall output. The *boosting* method converts weak learning algorithms into ones that achieve arbitrarily high accuracy [21].

Dynamically structured committee machines differ in that the input signal is directly involved in actuating the mechanism which combines the outputs of individual experts into an overall output. The *mixture of experts* structure combines the expert's responses in a nonlinear fashion through the use of a single gating network. The *hierarchical mixture of experts* structure combines the expert's responses in a non-linear fashion through the use of multiple gating networks arranged in a hierarchical fashion [21].

Recurrent Networks are distinguished from feedforward networks in that they have at

least one *feedback loop*. The presence of these feedback loops have a profound effect on the learning capability and performance of the network. The feedback loops make use of *unit-delay elements* which allow a number of previous decisions to have an effect on the current decision-making process, or allow time to be built into the operation of the neural network. This feedback can be local (at the level of a single neuron) or global (encompassing the whole network) [21].

2.3.3 Learning Processes

As ANNs are universal approximators, they need some function or decision/classification process to approximate. In order for the ANN to learn this process, they need to be trained by some learning algorithm on some data set [21].

The data is usually divided into three sets: the *training set*, the *validation set* and the *test set.* The ANN is generally trained using data from the training set, while using data from the validation set to prevent overtraining [21]. Overtraining occurs when the ANN approximates the training data so closely that it is no longer effective in approximating the rest of the data. The validation set, although not used in training, is tested during training. When the accuracy of the results in the validation set begins to drop, and accuracy in the training set improves, it is seen as a good time to stop training in order to keep the ANN more generalised [21]. The ANN can then be tested on the unseen test set to see how well it performs on unseen data.

As was previously stated, ANNs can learn through one of two paradigms: learning *with* a teacher, and learning *without* a teacher. Learning with a teacher is also referred to as *Supervised learning.* Learning without a teacher is further subdivided into *Unsupervised learning* and *Reinforcement learning* [21]. Mitchell proposes *Genetically Trained ANNs*, where GAs can be used to train and build ANNs [40]. Each weight and bias of the ANN is essentially a parameter in a function which could be optimised [40]. The remainder of this Section will discuss these learning paradigms in further detail.

2.3.3.1 Supervised Learning

Supervised learning requires a teacher or supervisor which classifies the training examples into classes. The information on the class membership of each training instance is utalised to detect pattern misclassifications. These pattern misclassifications act as a form of

feedback to themselves, whereby during the learning process correct classifications are rewarded, and misclassifications are punished. This process is known as *credit and blame assignment* [21].

The most well-known and widely used form of supervised learning in neural networks is *backpropagation [42].* The backpropagation network is a multilayer feedforward neural network with a differentiable transfer function in the artificial neuron [17]. Inputs are fed into the backpropagation network, and outputs are generated. These outputs are compared to known targets, and the synaptic weights are altered in order to minimise the errors between the generated outputs and the known targets [55]. The training instance set for the network needs to be presented many times for the synaptic weights to settle into a state for the correct classification of input patterns; but once it does, the neural network approximates the classification system [17].

2.3.3.2 Unsupervised Learning

In unsupervised learning, there is no external teacher or critic overseeing the learning process. Rather, provision is made for a *task independent measure* of the quality of representation that the network is required to learn. The free parameters of the network are optimised with respect to that measure [4]. The most popular methods of unsupervised learning are: *competitive learning*, *self-organising maps* and *adaptive resonance theory (ART)* [17, 21].

A **competitive learning** rule may be used to perform unsupervised learning. This is when a *competitive layer* is added to the end of the network. The neurons in the competitive layer compete with one another (in accordance with a learning rule) for the "opportunity" to respond to features contained in the input data. Basically the network operates in accordance with a "winner-takes-all" strategy, as the neuron with the greatest total input is declared after a predetermined number of epochs, and "wins" the competition. At this point all other neurons in the competitive layer turn off, and the "winning" neuron is turned on [21].

In a **self-organizing map**, neurons are placed at the nodes of a one- or two-dimensional[1] lattice. The neurons become *selectively tuned* to various input patterns, or classes of input patterns through a competitive learning process. The locations of the "winning neurons" become ordered with respect to one another in such a way that a meaningful coordinate

[1]Higher dimensional maps are possible, but are not as common.

system for different input features is generated over the lattice [30]. Thus, a self-organising map is characterised by the formation of a *topographic map* of the input patterns in which the *spatial locations* of the neurons in the lattice are indicative of intrinsic statistical features contained in the input patterns - hence the name "self-organising map". Self-organising maps are generally used in pattern recognition problems [21].

Adaptive Resonance Theory (ART) networks are designed to learn new patterns, and retain knowledge of previously learned patterns simultaneously through the use of pattern resonance. ART networks have two layers, an *input layer* and an *output layer*. The use of resonance between the pattern in the input layer and a pattern in the output layer, is able to establish a good hetroassociative pattern match [17]. The first (input) layer of an ART network receives and holds the input pattern. The second (output) layer responds to the input pattern with a pattern classification or association, termed the *recognition* phase. The classification is verified by sending a return pattern to the first layer from the second layer, termed the *comparison* phase. If the return pattern is similar to the input pattern, then there is a match. If the return pattern is substantially different from the input pattern, then the two layers resonate by communicating back and forth, seeking a match. If a novel input pattern fails to match the stored patterns within a set tolerance level, termed the *vigilance parameter*, a new stored pattern is formed [17]. ART networks are the most useful for pattern clustering, signal classification and recognition problems [8].

2.3.3.3 Reinforcement Learning

Reinforcement learning is a behavioural learning problem whereby the learning system *interacts* with its environment, so that the system seeks to achieve a specific goal despite the presence of uncertainties [2, 60]. As this interaction is performed without a teacher, reinforcement learning is particularly attractive for dynamic situations where it is costly or impossible to gather a satisfactory set of input-output examples [21].

There are two main approaches to reinforcement learning: the *classical approach*, and the *modern approach* [21].

The **classical approach** is rooted in psychology, going back to the early work of psychologists like Thorndike and Pavlov [21]. Learning takes place through a process of reward and punishment, with the goal of achieving a highly skilled behaviour [21]. This represents a class of algorithms for learning automata. The automaton takes one set of actions

according to a corresponding probability. The environment responds to it by showing "success" (+1) or "failure" (-1) as a form of feedback termed *the global reinforcement signal*. The automaton learns by altering the probabilities corresponding to the action taken as a response to the reinforcement signal [17].

The **modern approach** builds on a mathematical technique known as dynamic programming, to decide a course of action through considering possible future stages without actually experiencing them. Thus the emphasis in the modern approach is on *planning* [21]. Dynamic programming is a technique where decisions are made in stages, with the outcome of each decision being predictable to some extent before the next decision is made. Decisions cannot be made in isolation, rather the desire for a low cost in the present situation must be balanced with the undesirability of high costs in the future. Thus dynamic programming addresses the question: How can a system learn to improve long-term performance when this may require sacrificing short-term performance? The modern approach is termed *Neurodynamic programming* as dynamic programming provides a theoretical foundation, and the learning capacity is provided by neural networks [21]. Bertsekas and Tsitsiklis [5] define neurodynamic programming as follows:

> "Neurodynamic programming enables a system to learn how to make good decisions by observing its own behaviour, and to improve its actions by using a built-in mechanism through reinforcement."

A neurodynamic system is able to make good decisions by observing its own behaviour, and improve its actions through reinforcement mechanisms. This is useful for intelligent agents in changing environments, for example as opponents in board games [21].

2.3.3.4 Genetically Trained ANNs

As mentioned above, an ANN can be trained by a GA. The fitness function would create an ANN with the given parameters being weights and biases. The fitness function would then simulate the task on test data, and compare the activations with the known results. The GA can then optimise the error between simulated data and known results [40].

In a study done by Montana and Davis [41], genetic algorithms were successfully used to train feedforward neural networks. Montana and Davis encoded 126 synaptic weights into a single chromosome, and bred solutions using the sum of the square errors (collected over all training cycles) as the fitness function. For this study, genetic algorithms

significantly out-performed the supervised backpropagation method. Montana and Davis conclude that genetic algorithms may not out-perform backpropagation in all cases, but are exceptionally useful in unsupervised learning tasks where test cases are not available to the learning system [41].

Genetically evolved architectures for ANNs. Mitchell [40] also claims that genetic algorithms are successfully used in evolving network architectures (the number of neurons and layers) in neural networks. This is evident in the two most widely used methods of evolving network architectures.

Direct encoding is a method of evolving network architectures, conceived by Miller, Todd and Hedge [37], whereby the connection topology of the network is represented in an N x N *connectivity matrix*, where each entry represents a connection from the "from unit" to the "to unit". Column numbers for the connectivity matrix represent the "from unit", and row numbers represent the "to unit". All values are zero unless there is a connection from the neurons: column number to row number. The matrix is then converted to a bitstring, which is the chromosome. Each network is then given the same initial inputs, and trained for the same number of epochs using backpropagation. The fitness of the chromosome is the sum of the errors squared on the training set at the last epoch. They concluded that the genetic algorithm easily found networks which readily learned to approximate the functions [37]. The functions they were approximating however, were too simple to be a rigorous test of this method [40].

Grammatical encoding is a method illustrated by Kitano [29], who points out that direct encoding methods lead to performance issues and inefficiency in larger sized networks. Kitano's solution was to encode the network architecture as grammars[2], and evolve the grammars using genetic algorithms. Kitano used a grammar called a "graph-generation grammar" to successfully generate and evolve network architectures in a more efficient manner than direct encoding [29].

2.3.4 Summary and Applications

In the context of Technical Analysis, perceptrons and single-layer perceptrons would not be a sensible option to take as there are many other more advanced methods available. Multilayered perceptrons could be used to combine signals effectively as an approximation

[2] A grammar is a set of rules that can be applied to produce a set of structures.

to some non-linear function. RBF networks are primarily concerned in pattern recognition, interpolation and classification systems, and thus do not feature too well in the current problem domain.

Committee machines would be perfect solutions to the problem at hand, as the signals generated by different technical trading tools could be seen as expert knowledge. Thus they could easily combine this knowledge to reach a decision superseding that of the individuals.

Recurrent networks allow time to be built into the decision-making process, and thus would provide interesting information as the problem domain revolves around time series data.

As the intended implementation for ANNs will be in a situation where it is not feasible or in the best interest of the system to generate targets, supervised learning techniques are not considered. This excludes support vector machines as a viable structure, as they require support vectors drawn from the training examples.

This means the system will have to learn without a teacher, leaving unsupervised learning, reinforcement learning and genetic training as the only viable options.

As unsupervised learning lends itself to clustering, it is optimal for pattern recognition and classification problems [21]. Due to the complex nature of reinforcement learning, genetic training is chosen as the best method of training ANNs for the purposes of this study.

2.4 Artificial Intelligence in Technical Analysis

As discussed above, technical tools provide valuable information about the market. However, few attempts have been made to amalgamate the information generated by a number of tools to create a more powerful technical tool.

The most well-known system which aggregates technical trading rules is CRISMA. It was developed in 1988 by Pruitt and White [52]. CRISMA combines signals from cumulative volume, relative strength indices and moving averages to predict buy and sell points in equities. This system finds statistically significant profits, even after accounting for risk and transaction costs. Pruitt, Tse and White [51] supported this with more evidence of profitability during the period between 1986 and 1990.

CRISMA came under scrutiny by Goodacre and Kohn-Speyer [20], who finds that all profits generated by CRISMA on FTSE 350 stocks in the period 1987–1996 were lost, after market movements and elevated risk were accounted for. Marshall, Cahan and Cahan also refute CRISMA with their conclusion that "the profit attributed to CRISMA by the original Pruitt and White studies is unique to the sample of stocks they study" [35]. Although CRISMA appears to generate mixed results, it still raises the significant concept of combining rules to find profitable trading systems.

CRISMA used a simple conditional strategy to generate signals in which a condition needs to be met for each rule before a signal is generated. This is limiting in that strong signals from two rules would be ignored if the third signal did not prove strong enough.

Chenoweth and Obradovic [9] overcame this issue by using a multi-component rule out of two neural networks which fed into a decision rule. This too provided signals which achieved an annual rate of return in excess of a buy and hold strategy. They did not however, take transaction costs into account [9].

2.4.1 Summary

It has been shown that previous systems have been successful in finding profit through technical analysis, but these are not without their criticisms. In general, the criticisms of these systems have due to overlooking transaction costs and interest rates, and data snooping by writing only on profitable data samples.

2.5 Summary

This chapter has shown that technical trading tools are examples of multiparametered functions which can be optimised with EC, using combinations of historic market data to predict future price movement. Modern studies indicate that they are profitable on their own, and hence their optimisation should only prove stronger.

It also provided an overview of GAs, showing that they are a commonly used form of parameter optimisation. In particular, they are used for functions over a multi-modal, noisy, discontinuous solution space. They have been successfully used, since the 1960's, to optimise a number of engineering problems. GAs have also been applied to the optimisation of technical trading tools, thus they are well-suited to the needs of this work.

Since each of these trading tools uses different combinations of historic data, the tools provide different and often conflicting signals, thus they need to be combined somehow to produce a single signal.

Artificial Neural Networks, as proven universal approximators, are able to do this. If there exists a "thought" process which combines these different signals to produce a profit, it can be approximated using ANNs. This choice avoids the criticisms of the CRISMA system, which used a simple conditional strategy to generate signals, as ANNs use a more non-linear approach.

The following chapter will apply the knowledge gained in this chapter to produce a profitable system of technical analysis. This system will rely on GAs and ANNs to optimise and amalgamate signals from: filter rules, trending, RSI, MACD and MOM.

Chapter 3

Design and Implementation

This section describes the overall design of each individual component of the system and explains how they work together to produce positive results.

As can be seen in Figure 3.1, the data (asset price information) is fed into each Technical Trading Tool along with the required parameters for that tool. Each tool then uses this set of parameters and data to generate its own set of buy, hold and sell signals. The signals generated by each tool are simultaneously fed into an ANN to be intelligently combined. This ANN takes a number of parameters to create: one parameter for each weight and bias as well as parameters for the architecture of the ANN. The ANN produces a single set of buy, hold and sell signals from the set of signals sent to it from the trading rules. This final set of signals is then simulated on the given data series, with the given signals by a Simulation Environment (SE). The SE then returns the profit per annum for the specified signals and risk thresholds over the given asset price information, transaction costs and interest rates.

As mentioned in Section 2.1.3.1, GAs have been chosen as the means of optimisation. The GA is used to optimise the parameters for each Technical Trading Tool, as well as the ANN and risk thresholds to find maximum profit. This is illustrated in Figure 3.2.

The rest of this chapter goes into detail about how each component is designed and built, what parameters are used, and the considerations behind them.

These components will be described in the order of the process shown in Figures 3.1 and 3.2. First the design and implementation of each of the technical trading tools is discussed, as these tools generate the signals. This includes: filter rules, trending, RSI,

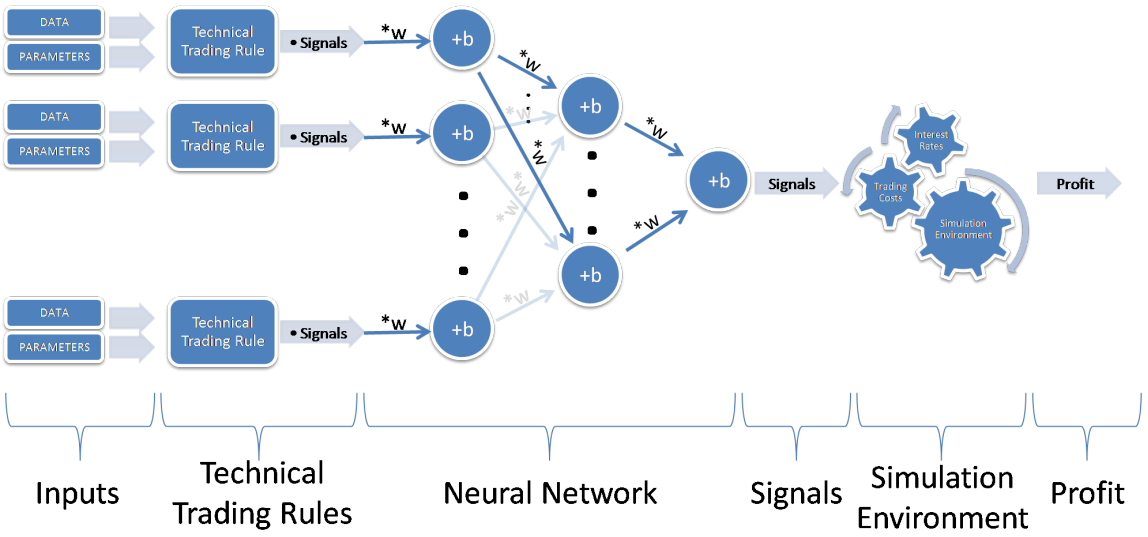


Figure 3.1: Overall Design of the System

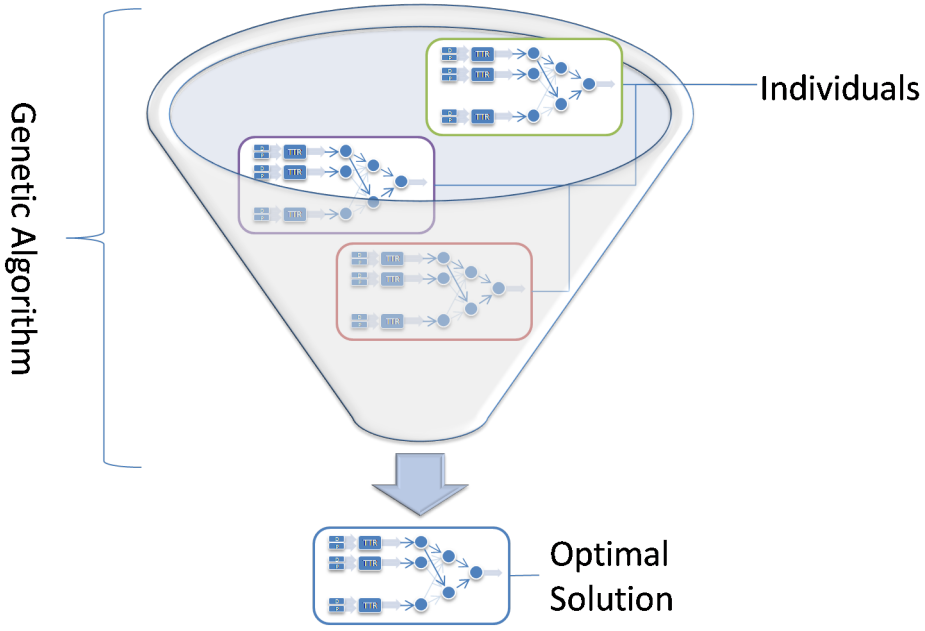


Figure 3.2: Overall Design of the Genetic Optimiser

MACD and MOM. This is followed by an investigation into the type and architecture of the ANN, which is used to amalgamate the signals generated by the trading rules to produce a single set of signals. The simulation environment is explained, as it assesses the profit made by the single set of signals on the given data set. The final component to be discussed is the genetic optimiser; detailing the parameters tested and the reasoning behind them. Finally the methodology of the tests is explored, after which the chapter is summarised.

3.1 Technical Trading Tools

As illustrated in Figure 3.1, the first component of the system to be evaluated are the technical trading tools. Each tool requires a number of parameters as well as the asset price information. These are used to mechanically perform technical analysis on the data; returning a set of buy, hold and sell signals for each tool. This section details the mechanical nature of each of the trading rules, the parameters they require, and how they were created.

3.1.1 Filter Rules

As stated in Section 2.2.1.1, filter rules encourage a trader to buy when an asset value rises a certain percentage above a previous local high, and sell if the value declines a certain percentage below a local low.

This is programmed by searching through the data at intervals of the given *window size* for maxima and minima in that interval. The maxima and minima found are then stored in an indexed array of the same size as the data. The data at each point is then compared to the indexed maxima and minima for each point.

A *filter size* is then chosen as the threshold by which a data value needs to exceed its local boundary point. If the data value is higher than its local maxima by a factor of *filter size*, it becomes a buy signal; and if it is lower than its local minima by a factor of *filter size*, it becomes a sell signal. For example: if the *filter size* was 10%, the data point would have to be 10% higher than a local maxima to be registered as a buy signal. The recommended values for the *filter size* is 15 days [49].

3.1.2 Trending

As previously discussed in Section 2.2.1.2, trending generates trading signals at the intersection of a short- and a long-run moving average. A buy is indicated when the short-run moving average intersects the long-run moving average from below, indicating that the asset value is trending upwards. The converse is true for sell signals.

These signals are generated by the distance between the two moving averages. The closer they become, the stronger the signal. If the short-run is above the long-run, and they are close together, then a sell signal is generated. If the short-run is below the long-run, then a buy signal is generated. The recommended values for the window sizes of the long- and short-run moving averages are five and 20 days respectively [49].

3.1.3 RSI

Section 2.2.1.3 noted that RSI indicates the Relative Strength of an asset as a value between 0 and 100. If the RSI value is low, it indicates that an asset has been over-sold, and thus it is a good time to buy the asset. If the RSI value is high, it indicates that an asset has been over-bought, and thus it is a good time to sell the asset.

This is calculated by first finding the upward and downward changes.

Let Up be the set of all upward changes and $Down$ be the set of all downward changes described by Equations 3.1, 3.2 and 3.3.

$$change = close_t - close_{t-1} \tag{3.1}$$

$$Up = change \tag{3.2}$$

$$Down = -change \tag{3.3}$$

The ratio of the averages is termed the *Relative Strength* (RS), and is calculated as shown in Equation 3.4, where $EMA(n)$ is the Exponential Moving Average with an n-day smoothing factor.

$$RS = \frac{EMA(n)\,of\,Up}{EMA(n)\,of\,Down} \tag{3.4}$$

This is then converted to a *Relative Strength Index* (RSI) between -1 and 1 to be interpreted as buy and sell signals, shown in Equation 3.5.

$$RSI = (2 - (2 * \frac{1}{1 + RS})) - 1 \tag{3.5}$$

This can be done in a single step, as the RSI expresses the upward movements as a proportion of the total upward and downward movements, as illustrated in Equation 3.6.

$$RSI = 2 * \frac{EMA(n)\,of\,Up}{(EMA(n)\,of\,Up) + (EMA(n)\,of\,Down)} - 1 \tag{3.6}$$

This produces buy and sell signals based on the RSI of the asset value. The recommended value for n is five days [49].

3.1.4 MACD

As mentioned in Section 2.2.1.4, the MACD shows the difference between a fast and slow exponential moving average (EMA) of an asset's closing price information. Two moving averages are compared to eachother: an s-day short run moving average, and an l-day long run moving average. The distance between the two is the $MACD$, calculated as shown in Equation 3.7.

$$MACD = EMA(s)\,of\,price - EMA(l)\,of\,price \tag{3.7}$$

This then needs to be compared to the signal line, calculated as indicated in Equation 3.8, which is an m-day moving average of the $MACD$ line.

$$signal = EMA(m)\,of\,MACD \tag{3.8}$$

Trending is then applied to the $MACD$ and *signal* values. Thus a buy is signaled when the $MACD$ line intersects the *signal* line from below, and a sell is signaled when the $MACD$ line intersects the *signal* line from above. The recommended values for s, l, and m, are 12, 26 and 9 respectively [49].

3.1.5 MOM

Section 2.2.1.5 explained that MOM shows the momentum of an asset price. This is calculated by looking at the change in asset price from the current day against n-days ago. This is calculated as shown in Equation 3.9.

$$momentum = close_t - close_{t-n} \tag{3.9}$$

If the *momentum* returns a negative value, this means that the asset price is downward trending which is signaled as a sell, whereas if the *momentum* is positive, the asset price has an upwards trend and is signaled as a buy. The recommended value for n is twelve days [49].

3.1.6 Summary

In this section, the mechanical nature of the technical trading tools is highlighted by describing the manner in which they are implemented in the system.

Each of these tools takes a number of parameters as inputs, and outputs a set of fuzzy signals which act as buy, hold and sell indicators. A large, negative number represents a strong sell signal, and a large positive number represents a strong buy signal.

3.2 Artificial Neural Networks

As a number of signals have been generated by the individual trading rules, they need to be combined. This is necessary, as it is difficult to follow a number of different, and possibly conflicting signals in a mechanical manner. This task is performed by the next tool in the process: the ANN. The ANN takes a number of sets of signals as inputs, which it intelligently combines to create an output of a single set of buy, hold and sell signals, as is illustrated in Figure 3.1.

As mentioned in Section 2.3, there are many different types and architectures of ANN. The remainder of this section will detail the choice of ANN type, the architectures tested and the reasoning behind these decisions.

3.2.1 Type of Artificial Neural Network

A fully connected feedforward Neural Network was selected to be used as the combining function, due to its conceptual simplicity, and computational efficiency. This is because all neural networks are universal approximators. One type of ANN will not have approximation properties that make it better than another, they are only differentiated by their training mechanisms. Since the ANN is trained genetically, this bears no consequence.

3.2.2 Architecture of the Artificial Neural Network

The architecture of this ANN however, does have an effect on the outputs of the system. Unfortunately, it is not viable to genetically optimise the architecture of the ANN as running a single test case takes a number of hours. Extrapolating from this, a standard GA with a population size of 20 over 100 generations, would take a number of years to optimise. Thus the number of layers will be tested first, and once the optimal number of layers is found, the number of neurons in each layer will be optimised.

The reason that the numbers of layers and neurons in each layer need to be optimised, is to prevent overtraining. This is a situation when the ANN becomes so specialised for the training set that it loses accuracy in the test set. According to Reed and Marks [55], one can generalise an ANN by *pruning* it. This is a process whereby neurons are removed until an optimal general solution is found which performs equally well in the training and test data sets. Thus, a number of different architectures will be investigated, and they shall be tested in decreasing levels of intelligence.

As overtraining may occur, even in the most simple ANN, an even simpler ANN is constructed by removing the ability to add a bias at each neuron. This leaves the *Simple ANN* as a linear scaling machine. The resulting signal generated by the Simple ANN is a linear combination of the other signals.

3.2.3 Summary

ANNs are used to intelligently combine the signals generated by the technical trading tools. The architecture of the ANN plays a role in its accuracy and generalisability, and hence different architectures are tested to find the optimal architecture for this system.

The full set of ANN architectures tested include the Simple ANN, and ANNs with zero, one and two hidden layers. The architecture which produces the greatest profit of the above will then be pruned until an optimal number of neurons is found.

3.3 Simulation Environment

Now that a single set of signals has been generated, it is necessary to find how profitable these signals are on the current data set. This is the task of the Simulation Environment (SE). The SE is able to do this by taking in a number of parameters: a time-series of currency data; a set of fuzzy buy, and sell signals for that time-series; the trading costs for that currency; and the buy and sell risk thresholds. The SE returns the geometric profit per annum for the data sent to it.

As shown in Section 2.2, existing Seas are often criticised for their lack of realism [15, 49]. This SE was designed to take factors such as trading costs, interest rates and realistic profits calculations into account, to provide the most realistic environment possible.

3.3.1 Formatting Signals

The SE takes the fuzzy buy and sell values outputted by the ANN as its inputs. As the signals are meant to represent buy, hold and sell signals, it becomes necessary to classify the fuzzy signals into crisp buy, hold or sell signals. To convert the fuzzy signals into crisp signals, there needs to be some form of decision-boundary or threshold value which delineates a weak signal (one that should be ignored) from a strong signal (one that should be adhered to).

This is the role of the risk thresholds. If a value is below the sell threshold, it is converted to a crisp sell signal. If the value is above the buy threshold, it is converted to a crisp buy signal, and if a value is between the two thresholds, it is converted to a hold signal. Thus these two risk thresholds create a space for hold signals in the continuous signal spectrum.

As the SE is used to find the profit for a given set of signals on a given set of data, it is counter-intuitive to buy currency without selling it. This is because it then becomes difficult to calculate the profit made in a year, as a sell at the end of the simulated period is not signaled. It is impossible to sell something before it has been bought. For these

reasons, the signals need to be "cleaned". This involves removing all sell signals before the first buy signal, and removing all buy signals after the last sell signal.

As the profits are being calculated geometrically, duplicate buy signals are redundant, as only the first buy signal will be adhered to. The same is true for sell signals. Thus to make it easier to find and use the relevant buy and sell signals, all duplicate signals are removed. A signal is considered a duplicate if there are consecutive signals of the same type. If a duplicate signal is found, it is converted into a hold signal. This means that between any two buy and sell signals, there are a number of hold signals (and only hold signals). This ensures that there are matching pairs of signals, and allows for easy calculation of profit for each transaction.

3.3.2 Calculating Profit

Profits for each transaction are then calculated by taking the geometric difference between the data for each buy and sell pair. To calculate the profit, let ζ_{γ_t}be the spot price of currency ζ when signal γ is found at time for transaction t. At time t purchasing price is thus $\zeta_{\gamma_{t-1}}$ and the selling price is ζ_{γ_t}, giving a return R for currency ζ at time t of $R_{\zeta,t}$, which is calculated as shown in Equation 3.10.

$$R_{\zeta,t} = \frac{\zeta_{\gamma_{t-1}}}{\zeta_{\gamma_t}} - 1 \tag{3.10}$$

Many papers are criticised as they fail to take trading costs and interest rates into consideration [49], thus this project sought to include trading costs and interest rates in the calculation of profits. Trading costs are the differences between the price for which a currency can be bought and sold at any point in time. The difference between the two is called the spread, and is measured in *pips* - one ten thousandth of a unit of exchange. These costs were determined using the OANDA electronic foreign exchange service [44], and are taken as twice the minimum cost facing non-institutional buyers. Transaction costs are factored in at each point by subtracting the transaction fee τ from selling prices and adding it to buying prices, altering the calculation for returns Equation 3.11.

$$R_{\zeta,t} = \frac{\zeta_{\gamma_{t-1}} + \tau}{\zeta_{\gamma_t} - \tau} - 1 \tag{3.11}$$

$$\iota = (1 + i_t)^{\frac{d}{365}} \tag{3.12}$$

$$R_{\zeta,t} = \frac{\zeta_{\gamma_{t-1}} + \tau}{\iota(\zeta_{\gamma_t} - \tau)} - 1 \tag{3.13}$$

$$R_{\zeta} = \prod_{t=1}^{n} R_{\zeta,t} \tag{3.14}$$

$$R_{\zeta,D} = (1 + R_{\zeta})^{\frac{D}{365}} \tag{3.15}$$

Interest effects are taken into account using the overnight rate i for each currency at the time for transaction t. Thus the interest effect, ι, is the compounded interest of the fraction of the number of days d in a year that the investment is in that currency, with an interest rate of i_t, as shown in Equation 3.12. Since the interest rate increases the value of currency holding, returns for transaction t are calculated as shown in Equation 3.13. As the returns are geometric in nature, and a series of returns per transaction is generated, one can calculate total profit for the period by continuously compounding the returns for each transaction. Equation 3.14 shows the calculation for the total return R_{ζ} for n transactions. This is then converted into annual profit by taking the power of the return as the fraction of the total time D that the investment existed in years, as illustrated by Equation 3.15. Thus the SE is able to calculate the total profit for the given data and fuzzy signals, while incorporating trading costs and interest effects.

3.3.3 Summary

The SE assesses the profitability of the inputted signals on the relevant data-set. First, the signals are prepared by: converting fuzzy signals into crisp signals; removing premature sell and overdue buy signals; and removing duplicate signals. Following this, profit per transaction is calculated, taking transaction costs and interest rates into account. The individual transactional profits are then combined to return the overall profit per annum for the inputted signals.

3.4 Genetic Algorithms

Once the profit for a set of trade rule parameters can be calculated, it is necessary to optimise these parameters to maximise profit. This is done by GAs, which apply the

Darwinian principles of evolution in order to optimise multi-parametric functions over noisy and discontinuous data sets, as mentioned in Section 2.1.3.3.

To create a GA, a number of parameters are required: a method of encoding the chromosomes, the fitness function used to calculate the fitness values of the chromosomes, the population size, initial population, maximum number of generations, selection method, scaling method, elite count, crossover fraction, mutation method, crossover method and stopping conditions. The remainder of this section details the parameters used and tested for this project.

- The entire signal generation, amalgamation and simulation process forms the *fitness function* for this GA, as is illustrated in Figure 3.2, and the *fitness value* is the profit generated by the parameters used to create the system.
- The GA represents the function's parameters as chromosomes. As mentioned in Section 2.1.3.3, there are many different methods of *encoding* these chromosomes. Encoding double values into binary strings increases the computational overhead, as they have to be converted back into double values to be evaluated. As such, this project will encode parameters as double vectors.
- The greater the *population size*, the more simulations need to be performed per generation, and thus the longer the run-time. However, the greater the population is, the more diversity will be present in the population, and the greater the chance that an optimal value will be reached [19]. Thus population size is a trade-off between speed and accuracy. This study investigates the optimal population size for the speed/accuracy trade-off.
- The *initial population* was randomly generated within all defined bounds and linear constraints. These constraints limit values to a continuous range which is different for different variables. All window lengths are strictly positive and are limited above by a maximum window length of 100 days.
- After some empirical research, a *maximum number of generations* of 150 was chosen, as results seldom required more than 100 generations to converge to a solution. Thus the maximum number of generations will not be a limiting factor in the generation of solutions, but will prevent the improbable case in which convergence does not occur.
- *Stochastic uniform selection* was used to select parents for the next generation, as it is recommended as the fairest method of selection by Mitchell [40].

- *Scaling* is done by rank. The algorithm scales the raw scores based on the rank of each individual instead of its score. This removes the effect of the spread of the raw scores.

- *Reproduction* is done with: an *elite count of two*. This ensures that the best two of the population survive to the next generation, and hence the best solution to date is never lost. This is not worth investigating, as this value needs to be greater than, or equal to one so that a best solution is never lost; but it is counter productive to have it greater than two, as benefits from the principles of GAs are lost if solutions are constantly copied.

- A *crossover fraction* of C is used, specifying the fraction of the next generation, other than elite children, that are produced by crossover. This means that $(C * 100)\%$ of the following generation will be combinations of parents, and $(100 - C * 100)\%$ will be direct copies of parents. Goldberg [19] recommends a crossover fraction of 0.8 be used.

- In order to prevent local minima being found, it becomes necessary to *mutate* the children. As mentioned in Section 2.1.3.3, there are many different methods of mutation. This study will investigate *dynamic Gaussian mutation*, *uniform mutation* and *adaptive feasible mutation*.

- In order to allow for all possible combinations, one and two point *crossovers* were rejected along with crossover hotspots. This left scattered crossover as the only viable crossover method - the implementation of which creates a random binary vector. Genes where the vector is a 1 are selected the from the first parent, and the genes where the vector is a 0, from the second parent. These are combined to form one of the children, and the remaining genes are combined to form a second child.

- The algorithm has two *stopping conditions*. Optimisation is halted if the weighted average change in the fitnesses over a given number of generations is less than a set function tolerance of $1 * 10^{-5}$; or if the maximum number of generations has been reached.

3.4.1 Summary

GAs have a number of parameters which affect their operations. This project applies those parameters which are recommended by literature, and investigates optimal population size

and mutation strategy. The fitness function is the process of creating the trading rules, building and setting the ANN, and simulating the resulting system - with profit per annum being the fitness value. The full set of parameters investigated consist of the population size and mutation strategy. By building a GA with the above parameters, the profit generated by the system can be optimised.

3.5 Methodology

As this chapter has shown, the trading system created for this project is comprised of individual trading tools which are intelligently combined to produce a single set of trading signals. These tools are are slowly fine-tuned by changing their input parameters through the mechanism of a GA. Each time any parameter in any individual tool is changed, the whole system must be tested.

To allow comparisons to be made across the parameter sets, the system must be tested in the same manner and with the same data. The remainder of this section describes the data used in these tests, the procedure in which the tests were conducted and the way in which the system shifts between long and short positions on the market as well as a summary of the above.

3.5.1 Test Data

In order to fully test the aforementioned system and avoid criticism based on data snooping, a number of different currencies were chosen over an extended period of time.

This data was obtained via the OANDA electronic foreign exchange service [44] as direct quotes relative to the USD at the daily interbank rate for each day in the five year period from 1st July 2003 to 31st June 2008.

The currency set includes: four major currencies (majors), the Euro (EUR), Japanese Yen (JPY), British Pound (GBP) and Swiss Franc (CHF); two minor currencies (minors), Canadian Dollar (CAD), and Australian Dollar (AUD); and four emerging market currencies, including the Brazilian Real (BRL), Mexican Peso (MXN), Polish Zloty (PLN) and the South African Rand (ZAR). These test currencies were chosen as the majors, minors and emerging markets as all exhibit different characteristics. As such, they all have

Currency	Interest Rate	Average Rate %	Transaction Cost (pips) USD/XXX	STD
USD	Us Overnight Repo Mm 10:00 Gc Rate - Middle Rate	3.21	N/A	N/A
EUR	Euro Overnight Deposit (Ecb) - Middle Rate	1.72	0.90	0.06
GBP	Uk Interbank Overnight - Middle Rate	4.73	2.50	0.03
CHF	Swiss Interbank 3m (Zrc:Snb) - Bid Rate	1.27	2.50	0.08
JPY	Tokyo Interbank Euroyen 3m - Offered Rate	0.33	1.60	5.68
CAD	Canada Money Mktovernight - Middle Rate	3.28	2.50	0.12
AUD	Australia Interbank 3 Month - Middle Rate	6.14	1.80	0.11
BRL	Brazil Financing Overnight Selic - Middle Rate	15.75	10.00	0.44
MXN	Mexibor Rate 3 Month - Middle Rate	7.53	40.00	0.29
PLN	Polish Zloti 3 Month - Middle Rate	4.23	15.00	0.49
ZAR	SA Rand Overnight Deposit Rate - Middle Rate	8.21	55.00	0.55
Mean		**5.13**	**13.18**	**0.79**

Table 3.1: Interest rates, transaction costs and standard deviations of each of the 10 currencies

different transaction costs, interest rates, and some are more volatile than others as shown in Table 3.1.

Transaction Costs were determined as twice the minimum cost faced by a small non-institutional trader. That is half the minimum spread, measured in pips (one ten thousandth of a unit of exchange) determined via OANDA [44]. Interest rates were taken as the overnight interest rates, or closest proxy for the period July 2003 through June 2008 [63]. This was typically a three month bond. Transaction costs and interest rates for each currency can be seen in Table 3.1.

3.5.2 Test Procedure

In order to compare changes in the architecture of the system, each test must be run in the same manner, on all of the data. Each test run, was on all of the ten currencies, over twelve different periods of data. These data-periods are of two years in length, and are uniformly selected from the five available years of asset price information. Each currency-period pairing is then repeated 30 times, and the mean of the repeats is taken in order to account for the stochastic nature of GAs. The system is trained on the first 365 days of the two-year period, by optimising the parameters of the technical trading tools, the ANN, and the SE with the GA. These optimised parameters are then tested on the next 365 days in the data series. This ensures that the system is tested on unseen data. It also ensures that the unseen data comes from the future relative to the test data to avoid

retrospective data snooping.

3.5.3 Shifting between Currencies

Drawing from Levich and Thomas [32], when transactions are simulated, they switch between long and short positions on the foreign currency. This means that when trading between two currencies, the one currency is bought with the other currency. For example, if the EUR/USD currency pair were being investigated, a buy signal would suggest buying EUR with USD, and a sell signal would suggest selling EUR for USD (or buying USD with EUR). All currencies are traded against USD, as all information gained from OANDA is in terms of USD [44].

3.5.4 Summary

To allow fair comparisons to be made between systems with different settings, the test procedure must be consistent, and test a wide variety of data. The procedure detailed above tests the system over twelve two-year periods uniformly selected from five years of asset price information for ten currencies. This is repeated 30 times, and the mean values of the repeats are taken, to account for the stochastic nature of GAs. The system is presented with 365 days of data training data, and tested on the following 365 days of data. The system has been configured to shift between the long and short positions when transactions occur.

3.6 Summary

As illustrated in Figures 3.1 and 3.2, each technical trading rule requires a number of parameters as well as the asset price information as inputs. The resulting signals generated by the technical trading rules are fed into the ANN to be amalgamated into a single set of signals. This ANN takes a number of parameters to create; one for each weight and bias, as well as parameters for the architecture of the ANN. The combined signals are then passed to the SE, along with the buy and sell risk thresholds. The SE then returns the profit per annum for the specified signals and risk thresholds over the given asset price information, while accounting for transaction costs and interest rates. A GA is used to optimise the entire system to gain maximum profit. The fitness function for the GA

comprises of the entire process of creating the trading rules and ANN, as well as simulating them. The output of the fitness function is profit per annum, in geometric terms, as the fitness value. The parameters that the GA is to optimise include: the parameters for each technical trading rule, the architecture of the ANN, all the weights and biases of the ANN, and the risk thresholds.

The system is tested over twelve periods uniformly selected from five years of asset price information for ten currencies and repeated 30 times to account for the stochastic nature of Genetic Algorithms. The system is trained on 365 days of data, and tested on the following 365 days. The inclusion of trading costs and interest rates mitigate criticisms of previous papers, and the wide variety of data used prevents any data snooping.

Chapter 4

Optimal Systems Architecture

Having developed a system for the optimisation of trading rules using GAs and ANNs in Chapter 3, this chapter will now focus on the analysis of the developed system to determine feasibility and optimal parameters.

This chapter is broken into the following sections. Section 4.1 discusses the optimisation of trading rules. Section 4.2 then investigates the optimal architectures of the developed ANN. Section 4.3 looks at the use of GAs as optimisation tools and investigates the optimal settings for the developed GA, after which the chapter is summarised.

4.1 Trading Rule Optimisation

In order to show that the trading rules used are optimisable, this section investigates the whether the trading rules employed are optimisable or not, by investigating three of the trading rules discussed in Section 3.1. These were studied by performing a simple grid search to optimise the possible inputs, and analysing the results. This study was taken over the same period of asset price information, for ten different currencies over the last five years of asset price information.

4.1.1 Filter Rules

Filter rules are optimisable on two dimensions: their window size and the filter size. By optimising the window size alone, it can be shown that this rule is optimisable. This

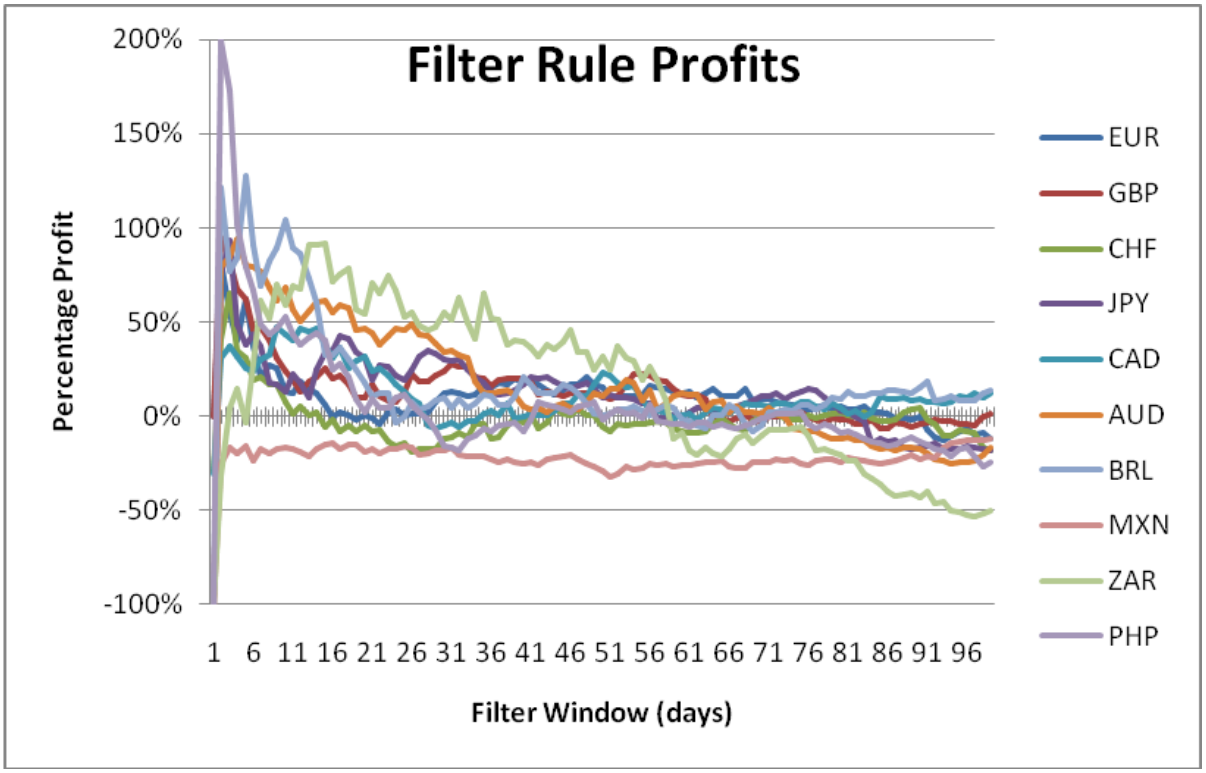


Figure 4.1: Filter Rule Profit at window lengths one through 99 days, for each of ten currencies

study shows profits returned over the entire data set, for each currency, with window sizes between one and 99 days. The results of this study are shown in Figure 4.1.

As shown in this figure, most of the currencies tested find their maximum profit with window sizes under five days, with the most popular window sizes being two and three days. The ZAR however, achieves maximum profit at 15 days, and the MXN never achieves profit. The PHP varies from -100% profit to 200% profit after the window size is increased by a mere two days.

These results show that, for filter rules, each currency has a maximum profit at different window sizes, and also that individual series are unique. Filter rules are therefore optimisable, and due to the dramatic changes in profit which may result from minor changes in filter window, filter rules are worth optimising.

4.1.2 Trending

Trending is a function of two inputs: the window length for the short-run moving average, and the window length for the the long-run moving average. These parameters are altered

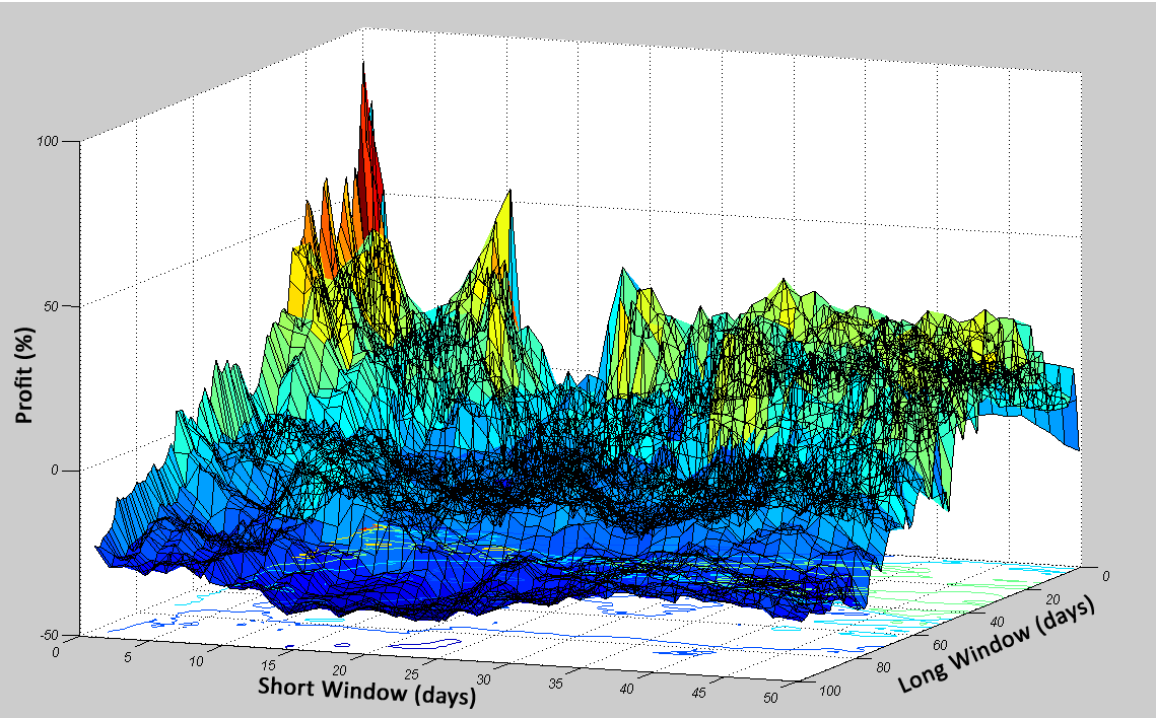


Figure 4.2: Trending Rule Profit at window lengths one through 99 days, and one through 50 days, resulting in the mean profitability over the ten currencies

via a grid search to show that trending is an optimisable trading rule. The short-run window length varies between one and 50 days and the long-run window length varies between one and 99 days. The results of this study are shown in Figure 4.2.

To show the results for all ten currencies on a single graph, the mean profitability was calculated and plotted as a surface with contours lines below. It is notable that the moving average pair of a one- and three-day window produced a profit of 89%, while other window lengths produced negative profitability. This shows that it is possible to find optimal values for the window lengths when using trending as a trading rule.

4.1.3 RSI

RSI is a function of one input: the window length. This parameter is altered to values between one and 99 to show that RSI can be optimised. The results of this study are shown in Figure 4.3.

This rule also performed well for shorter window lengths, but the results are a lot more erratic. Average profitability was retained up to a window length of 50 days. Shorter

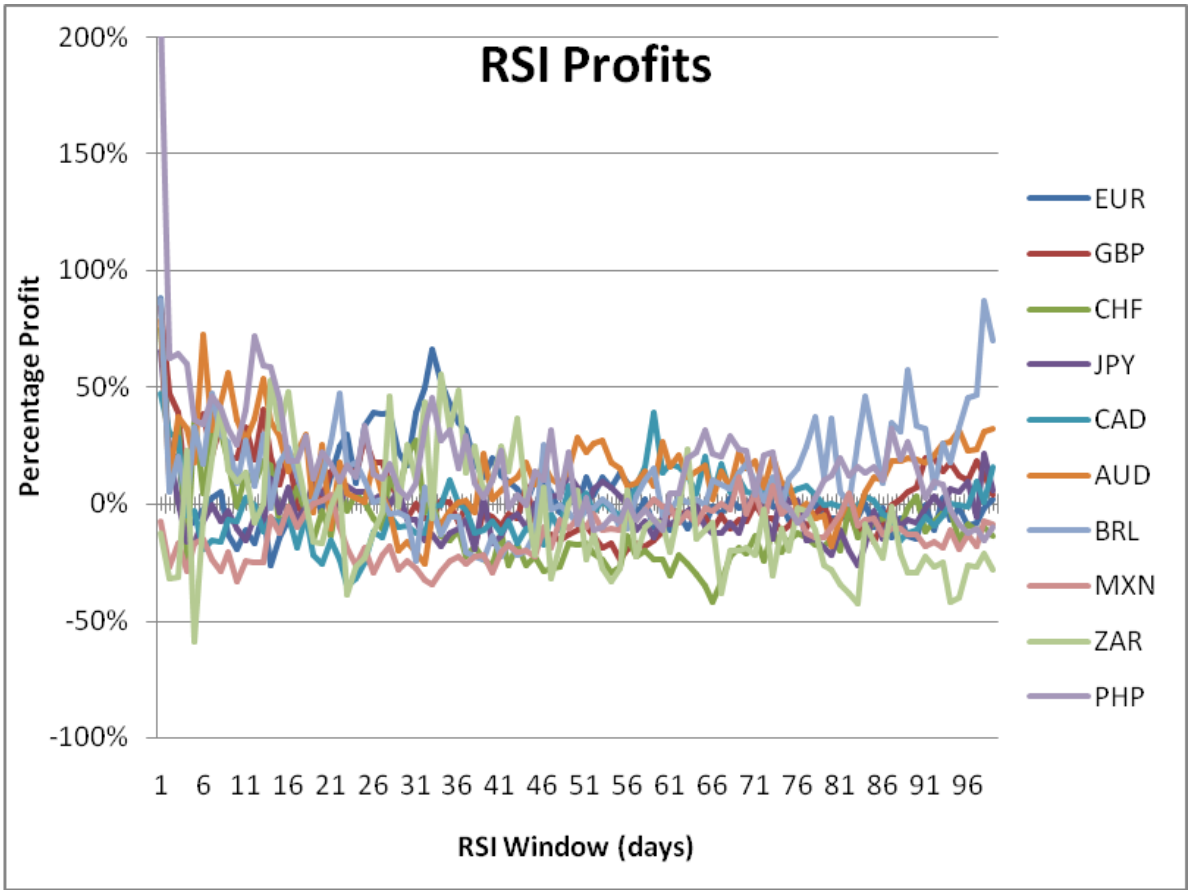


Figure 4.3: Relative Strength Profit at window lengths one through 99 days, for each of ten currencies

window lengths of two to three days are found to be optimal, reaching an average profitability of 48% and 47% respectively. The BRL finds its greatest profit at very long filter windows (between 70 and 99 days) with its maximum at 97 days. The JPY finds similar profits throughout the study, and the ZAR finds mean losses, but with a maximum profit at a window length of 34 days.

It is again obvious that the individual series are unique, and are profitable over different window lengths, thus RSI is an optimisable trading rule.

4.1.4 Summary

This section highlights that by altering the parameters of a trading rule, it is possible to alter the profitability of the rule. It has been seen that different currencies are optimisable at unique points for different parameters, therefore trading rules are optimisable for different data sets.

This result has important consequences. If optimal parameters can be found for a given investment period, greater profit will be attained. As the data set was for five years worth of data, and profit was found over the five year period, it is feasible that profitable parameters can be found with some consistency.

4.2 Artificial Neural Network Optimisation

The ANN is responsible for combining the different signals from the different trading rules, to form a single set of signals. These signals are tested by being simulated in the SE. Thus, in order to optimise the whole system, it becomes necessary to optimise the ANN. This means not only optimising the weights and biases to achieve maximum profit, but also adjusting the architecture of the ANN.

As stated in Section 3.2, the different architectures to be tested are: the Simple ANN, and ANNs with zero, one and two hidden layers. The results are presented below.

Table 4.1 shows the average profits generated per currency over the training and test data sets for each of the ANN architectures. Figure 4.4 shows a histogram of the average profits per period for the training sets, while Figure 4.5 shows the same, but for the test set. In the training set, there is a general trend that the more layers there are (and thus,

	Simple ANN		0 Hidden Layers		1 Hidden Layer		2 Hidden Layers	
Currency	Training	Test	Training	Test	Training	Test	Training	Test
EUR	16.3601	10.1533	30.354	5.5257	26.7294	5.7214	37.2426	6.2649
GBP	22.3344	11.3379	28.8627	7.0296	28.9344	7.0298	36.8112	6.8632
CHF	20.3527	11.6748	33.3197	4.3079	33.6051	8.5231	41.7721	6.2781
JPY	20.6926	14.1567	31.1253	5.7299	30.2234	7.2762	39.7961	8.5103
CAD	14.7921	7.0382	30.3686	1.9547	25.9393	6.2899	35.0154	6.0214
AUD	22.9903	15.5265	36.7332	6.2441	33.0782	9.1018	48.1643	10.1997
BRL	35.5415	19.5978	61.1777	6.7106	46.4157	9.5229	63.9352	12.0864
MXN	12.2033	3.743	28.0963	3.4257	20.7113	4.4095	35.0948	7.6991
PLN	29.858	7.1609	46.2388	5.4387	41.0052	6.3427	56.5073	8.9504
ZAR	34.5854	10.9346	111.4591	23.4889	52.2655	4.7211	75.9991	5.1613
Mean	22.97104	11.13237	43.77354	6.98558	33.89075	6.89384	47.03381	7.80348

Table 4.1: Overall Results for Neural Networks with varying architectures

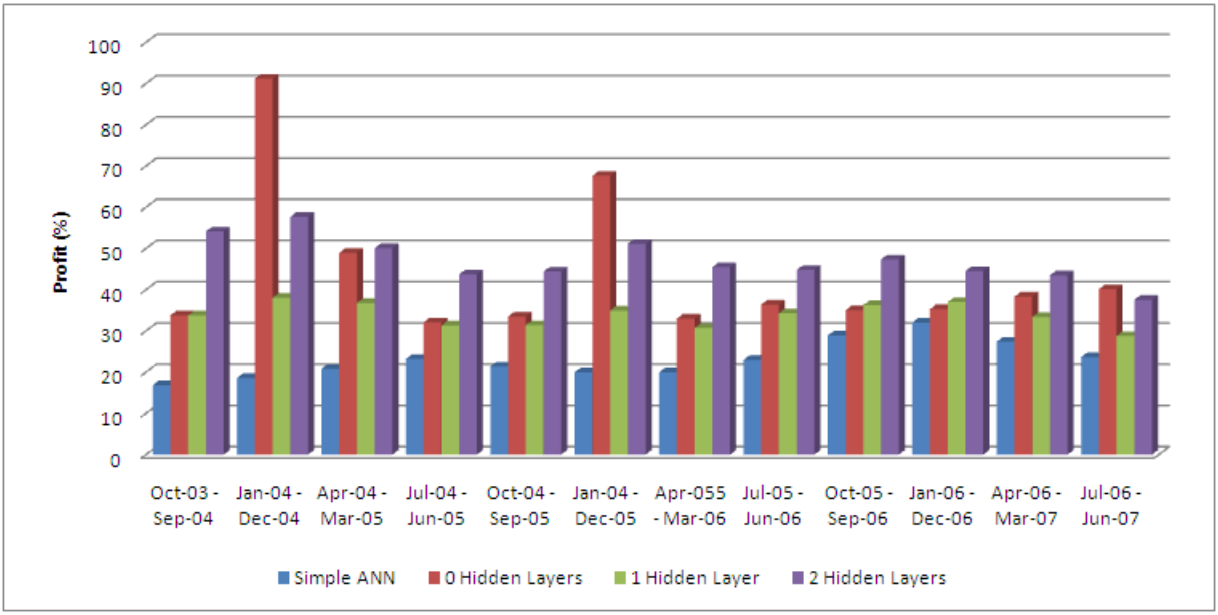


Figure 4.4: Histogram comparing the profit per period for the training data set of the four different ANN architectures

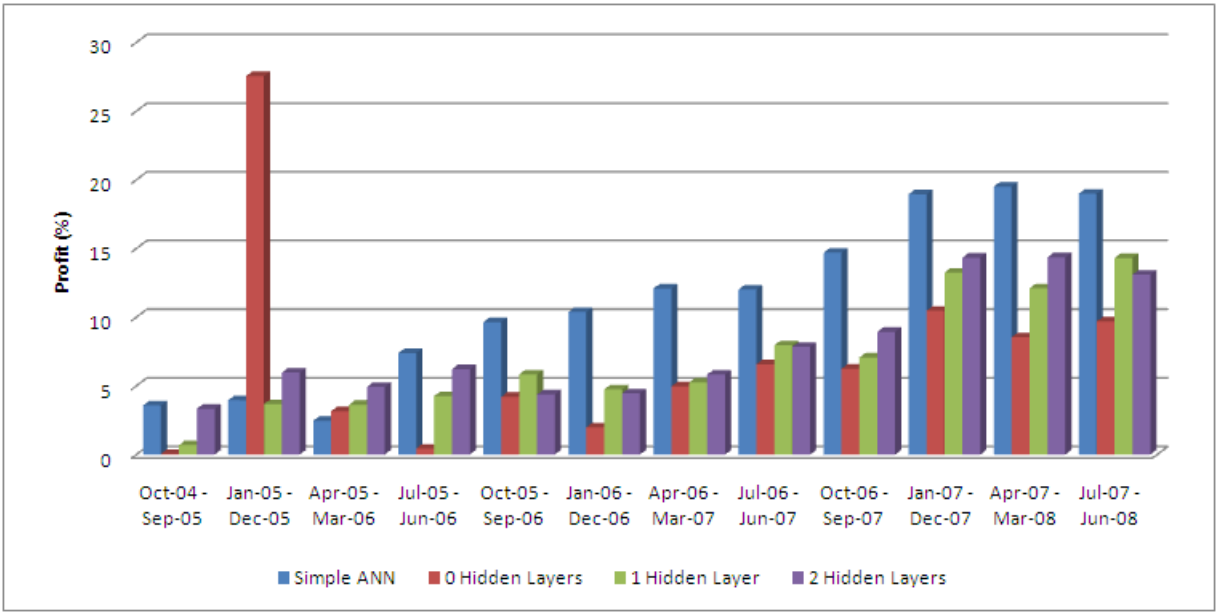


Figure 4.5: Histogram comparing the profit per period for the test data set of the four different ANN architectures

more neurons), the greater the profit. While in the test set the inverse is true, indicating the fewer layers, the greater the profit.

In Table 4.1, one can see that the "more intelligent" solutions have a bigger difference between profit generated over the training sets, and test sets. Although this is true in general, it should be noted that in some situations, the "more intelligent" solution clearly outperformed the Simple ANN.

This can be accounted for by the phenomena of *overtraining*. As explained in Section 2.3.3, overtraining occurs when the ANN approximates the training data so closely that it becomes highly specialised. This specialised solution however, is less effective in approximating the test data, as the two data sets are disconnected. Thus, in highly disconnected data sets, a more generalised solution performs better.

According to Reed and Marks [55], the more neurons and hidden layers there are (until a maximum of two hidden layers), the better an ANN is able to approximate complex functions. Thus the more the complex the ANN, the more specialised it can become. Although this is useful in finding specific solutions, it is counter-productive in this situation, where general solutions produce more profit in the test sets.

4.2.1 Summary

In this research, overtraining is evident in the architecture of the ANN. The more complex the architecture, the better it can approximate any non-linear function and thus the greater the possibility of overtraining. This is due to the stochastic nature of GAs. The greater the possibility of overtraining (due to increased complexity), the greater probability a solution will be overtrained, which causes an increase in the number of overtrained solutions .

The Simple ANN out-performs the "more intelligent" solutions in general, but has a few exceptions. It can be concluded that the network architecture itself can be likened to a trading rule, and thus needs to be optimised for the individual currency and period by the overarching system.

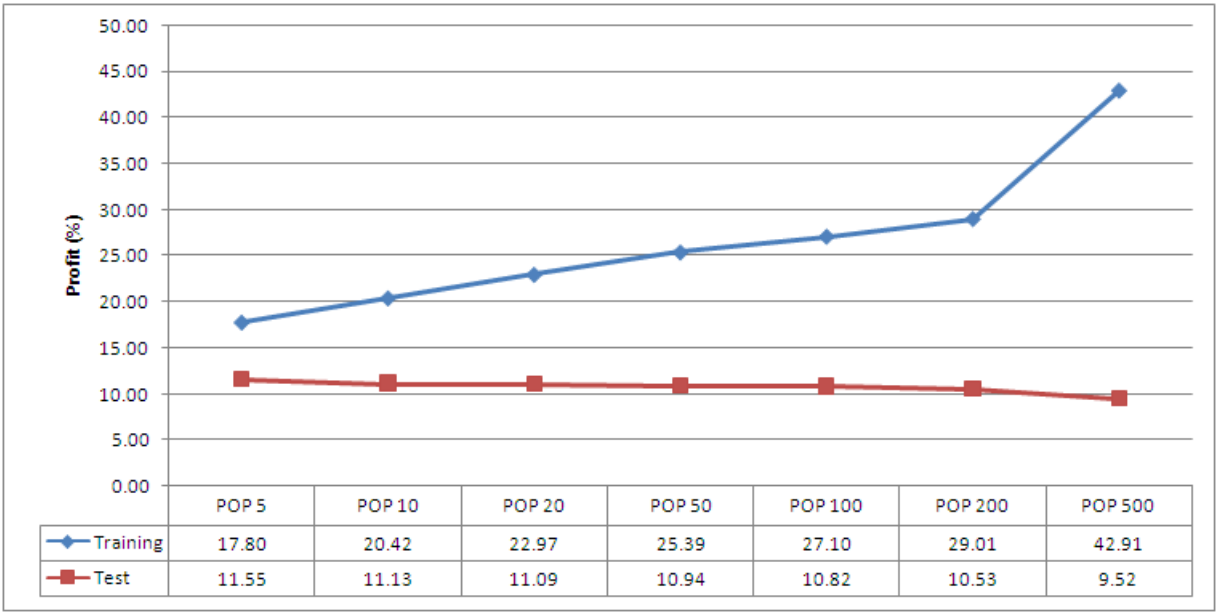
4.3 Genetic Algorithm Optimisation

As is evident from the literature in Section 2.1.3.3, GAs are effective optimisation tools, especially for noisy multimodal functions [40]. They have also been successfully applied to technical analysis in the past [1, 43, 48, 59]. Thus, GAs are seen as effective optimisers in the context of technical analysis.

As noted in Section 3.4, this thesis investigates the optimal population size, and mutation function for the genetic optimiser. The rest of this section examines each of the above, and implements the optimal parameters. These optimal parameters are then compared and contrasted.

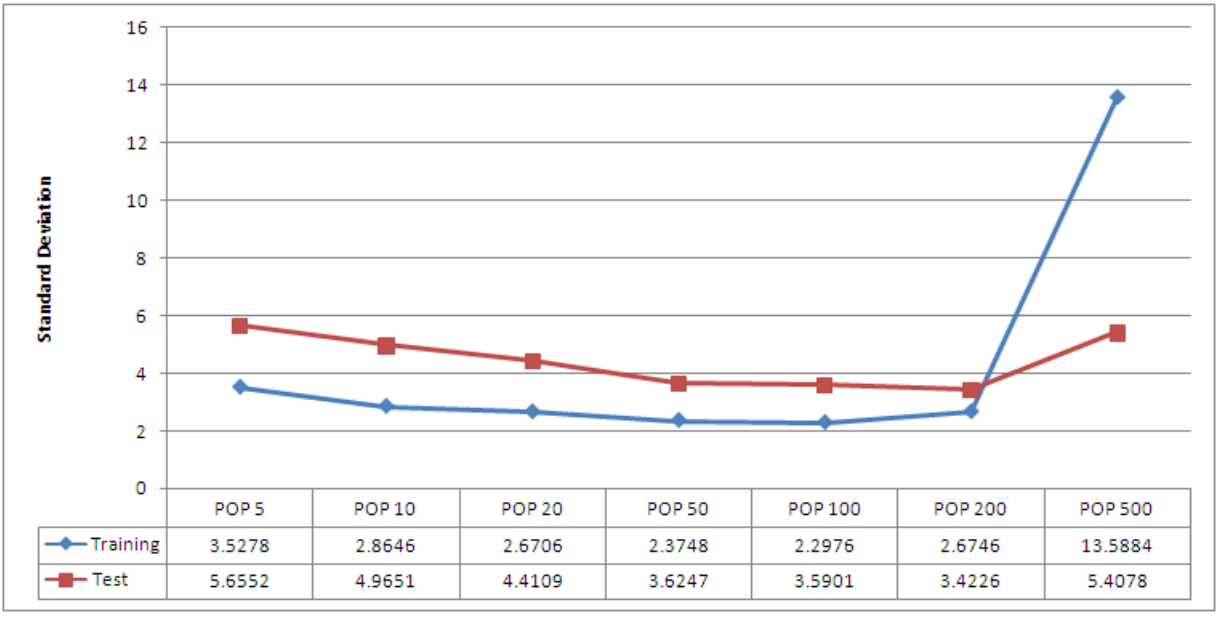
4.3.1 Varying Population Size

The first parameter for the GA which is examined is the *population size*. A set of populations between five and 500 were tested; the results are shown in Figures 4.6 and 4.7. This study shows that there is a clear benefit to having a greater population size, as they are able to find better maxima on the same data set and reduce the standard deviation of the results. This however, is problematic for the same reason that the "more intelligent" ANNs didn't perform as well in Section 4.2: overtraining.



	POP 5	POP 10	POP 20	POP 50	POP 100	POP 200	POP 500
Training	17.80	20.42	22.97	25.39	27.10	29.01	42.91
Test	11.55	11.13	11.09	10.94	10.82	10.53	9.52

Figure 4.6: Graph showing the profits generated when altering the population size of the GA



	POP 5	POP 10	POP 20	POP 50	POP 100	POP 200	POP 500
Training	3.5278	2.8646	2.6706	2.3748	2.2976	2.6746	13.5884
Test	5.6552	4.9651	4.4109	3.6247	3.5901	3.4226	5.4078

Figure 4.7: Graph showing the Standard Deviation of the profits generated when altering the population size of the GA

By finding the absolute global maximum, one is isolating very unique and specific points in which the parameters perform well. This is obvious in the sudden upward spikes in both the profits and standard deviations when a population size of 500 is used. Both spikes are due to the GA finding very specific and profitable combinations that other population sizes did not. This is due to the stochastic nature of GAs: the larger the population size, the greater the chance that random generators and mutations will find better results, and the more likely it is that unique maxima will be found.

The profits generated on the test set diminish as the population size is increased, while they increase as the population size is decreased. This can be seen in Figure 4.6. This is because more generalised/specialised solutions are being found. The specialised solutions perform well on the training set, but due to the disconnected nature of the data, their specialisation is limited to the training set. Standard deviation is highest with a smaller population. This is due to the stochastic nature of GAs; with smaller population sizes, GAs are less likely to find the same parameters twice - and so a greater variance is observed. Thus, a smaller population size produces more profit, but a larger population size produces greater stability, up until the point where highly specialised results are found.

4.3.2 Changing the Mutation Function

The second parameter of GA investigated is the *mutation function*. Dynamic Gaussian mutation, uniform mutation and adaptive feasible mutation are considered. The results of these are presented in Figure 4.8.

The study shows that there is relatively little difference in results between the different mutation options. Uniform mutation appears to have better profits and lower standard deviations than the others, with Gaussian mutation coming second, and Adaptive Feasible Mutation in last place. These results are inconclusive however, due to the stochastic nature of GAs, and the small deviations in the results.

4.3.3 Optimised Parameters vs Current System

After investigating a change in population size and mutation function, it is necessary to compare the best results of the above investigation with one another in order to ascertain the optimal parameters for the GA.

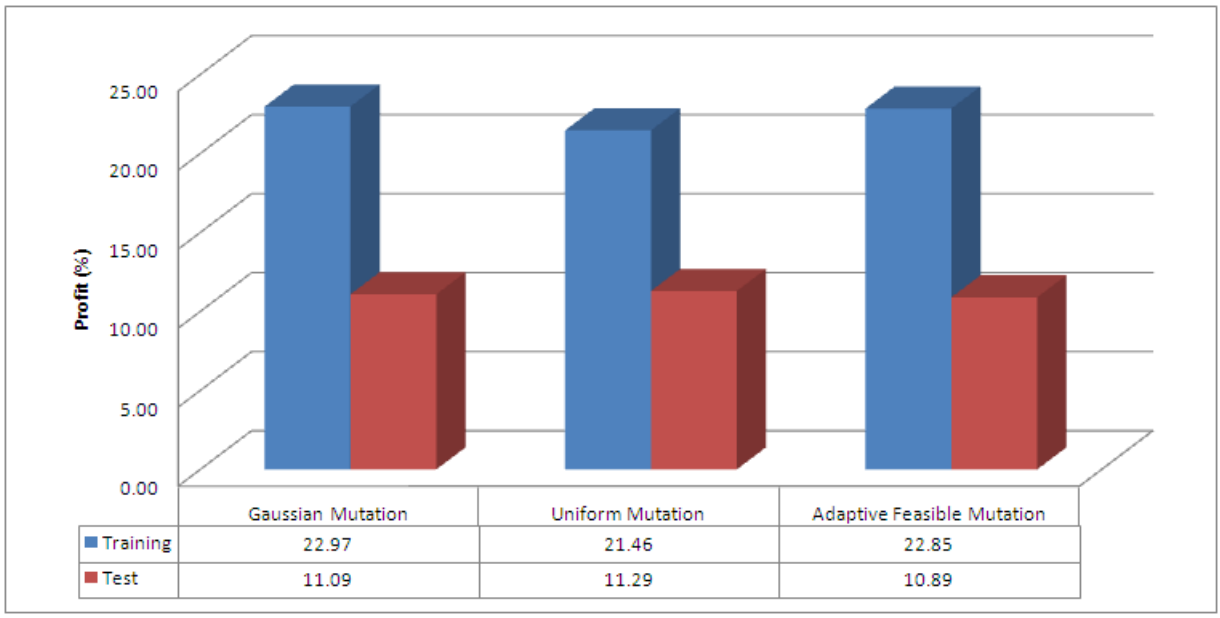


Figure 4.8: Graph showing the profits generated when altering the mutation function of the GA

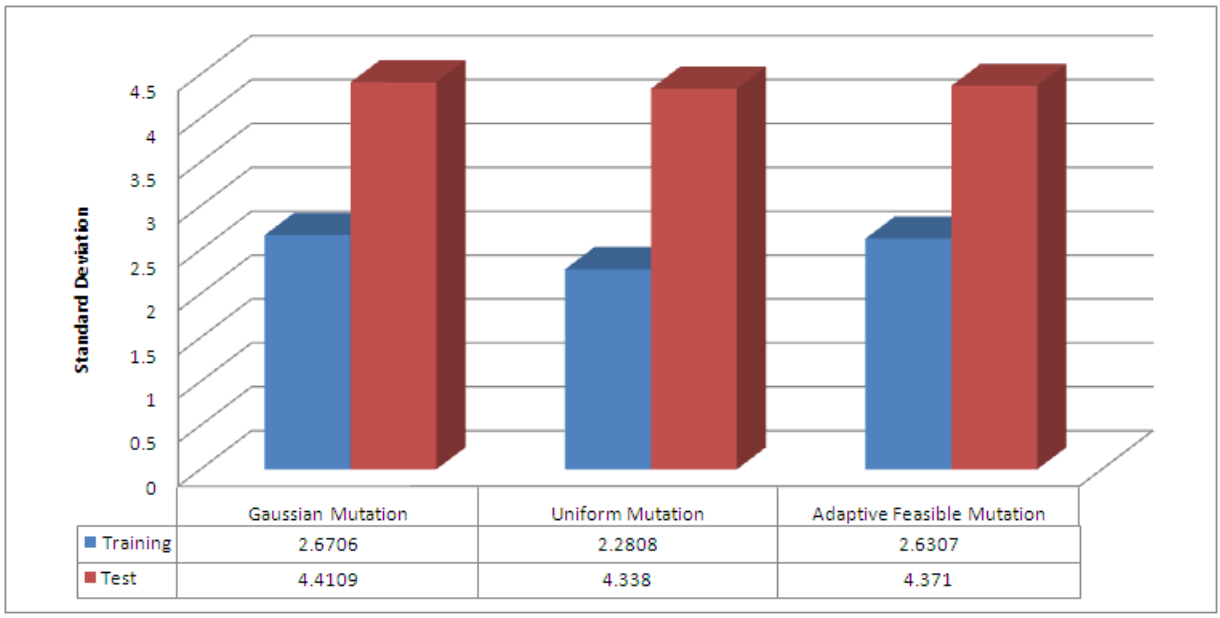


Figure 4.9: Graph showing the standard deviation of the profits generated when altering the mutation function of the GA

		Population of 20, Gaussian Mutation			Population of 20, Uniform Mutation			Population of 5, Gaussian Mutation			Population of 5, Uniform Mutation		
Currency	Tresury Bill (%)	Returns (%)	STD (%)	T-Value on T-Bill	Returns (%)	STD (%)	T-Value on T-Bill	Returns (%)	STD (%)	T-Value on T-Bill	Returns (%)	STD (%)	T-Value on T-Bill
EUR	3.59	10.15	3.49	1.88	9.74	4.07	1.51	10.75	5.41	1.33	10.43	4.48	1.53
GBP	3.59	11.34	3.34	2.32	11.31	3.24	2.39	11.97	3.88	2.16	12.07	4.24	2.00
CHF	3.59	11.67	3.94	2.05	10.75	4.54	1.58	11.30	4.97	1.55	11.21	4.73	1.61
JPY	3.59	14.16	4.14	2.55	12.88	4.51	2.06	14.20	5.07	2.09	14.04	5.24	1.99
CAD	3.59	7.04	3.28	1.05	7.59	4.02	1.00	8.55	4.79	1.03	9.33	4.74	1.21
AUD	3.59	15.53	5.28	2.26	15.24	5.51	2.12	16.62	6.22	2.09	16.61	6.47	2.01
BRL	3.59	19.60	5.21	3.07	19.33	7.48	2.10	18.89	7.59	2.02	18.58	8.90	1.68
MXN	3.59	3.74	2.95	0.05	2.71	3.43	-0.26	3.35	3.81	-0.06	2.90	3.70	-0.19
PLN	3.59	7.16	5.41	0.66	6.95	5.20	0.65	10.13	6.27	1.04	9.54	6.52	0.91
ZAR	3.59	10.93	6.55	1.12	10.86	7.53	0.97	9.76	8.54	0.72	10.13	8.25	0.79
Mean	**3.59**	**11.13**	**4.36**	**1.70**	**10.74**	**4.95**	**1.41**	**11.55**	**5.66**	**1.40**	**11.48**	**5.73**	**1.36**

Table 4.2: Results comparing the most profitable and stable parameters for the GA

A population size of five is chosen, as it achieved a greater profit than the other population sizes on the unseen test data. As the standard deviation of the profits decreased when the size of population was increased, and the inverse was true for the profits, a population size of 20 is chosen as an intermediate parameter. As this population size has an acceptable balance between profits and standard deviations.

The investigation into mutation functions suggested that the most profitable and stable mutation function was uniform mutation. This is compared to the next most profitable and stable mutation function: dynamic Gaussian mutation.

In order to compare the results of returns and standard deviations, the T-value was used. This measure shows the statistical significance of both the returns and standard deviations, when compared to the no-risk investment strategy of a US one-year Treasury Bill.

Table 4.2 compares the systems produces using parameters suggested by the investigations above. The table shows the average returns per currency, standard deviation of the returns, and the T-value against a one-year US Treasury Bill for each system. Further, it shows that the increase in profit generated by a population of size five is not proportional to the increase in the standard deviations of resulting profits. This is evident in the lower T-Values produced by the smaller population. This is because the population of size five, although generating higher returns, does so with a higher risk; and the increase in profits is not proportional to the increase in risk.

Even though uniform mutation proved more profitable and less risky than Gaussian mutation in Section 4.3.2, when running the tests again to compare the two for both population sizes, it performed worse, although not to a significant level. Thus no conclusions can be drawn from the change in mutation function.

4.3.4 Summary

In this section both changes in population size and mutation function were investigated and compared. The smaller population sizes produced more profits, but with higher risk; and the increase in profit was disproportional to the increase in risk, making profits generated by smaller populations more profitable, but disproportionately more volatile. A population of size 20 was found to be optimal, as it balances the increase in profits with the increase in risk. The change in mutation function however, provided inconsistent results which can be attributed to the stochastic nature of GAs. Hence, these results are inconclusive.

4.4 Summary

This section investigated a number of important queries - the answers of which would provide insight into the feasibility of an autonomous trader, as well as providing optimal parameters for such a trader.

The first investigation proved the necessary result that the technical trading tools used in the system are indeed optimisable. This was achieved by showing that currencies have unique series of results when parameters are changed, and they have unique parameters which return their maximum profit. Thus it is feasible to find profitable parameters for different data sets.

As the trading rules are optimisable, the next investigation delved deeper into the optimisation of the rest of the system, by altering the architecture of the ANN. This investigation showed that overtraining is of great concern, and there exists a relationship in that the "more intelligent" solutions were more likely to specialise and thus be overtrained, while the simple solutions proved more general.

Since optimal parameters for the entire system had been found, the next investigation looked into the optimal parameters for the optimising function. This investigation compared different population sizes and mutation functions for the GA to achieve optimal results. It was found that small population sizes provide more general solutions, but with a higher degree of risk; whereas larger population sizes produce specialised and stable solutions. Thus the optimal population size is approximately 20 individuals.

The investigation into the mutation function produced mixed results of little statistical significance, which was attributed to the stochastic nature of GAs. As such, the results of the study into mutation functions were inconclusive.

Chapter 5

Results

This chapter presents the performance results of the system developed in Chapter 3 and optimised in Chapter 4. The performance of the developed system is presented and analysed in Section 5.1, with respect to the statistical significance of the system compared to a no-risk investment strategy. Section 5.1 also analyses the effects that risk, and the number of transactions per annum have on returns. The developed and optimised system is then compared to the un-optimised trading strategies, using parameters suggested by literature.

5.1 System Performance

As the trading rules are optimisable, and the optimal settings for the GA and ANN have been found, it is necessary to investigate the results generated by this optimised system. The rest of this section examines the profits generated for each currency and period, as well as the statistical significance of these results. The trade-off between risk and return is discussed, as well as the relationships between the number of transactions and returns.

Table 5.1 describes the returns generated by the system, showing the average returns per currency, the standard deviations of those returns and the average number of transactions performed per year, over all periods. The transaction costs for each currency, and the statistical significance of the returns when compared to a no-risk investment strategy of a one-year US Treasury Bill are also shown.

The study finds relatively consistent results across all currencies, with an average annual return of 11.13%. The USD/BRL is found to be most profitable with an average of 19.6%

Currency	Returns (%)	STD (%)	Num Transactions	Transaction Cost (pips)	Tresury Bill (%)	T-Value on T-Bill
EUR	10.15	3.49	59.86	0.90	3.59	1.88
GBP	11.34	3.34	55.38	2.50	3.59	2.32
CHF	11.67	3.94	61.27	2.50	3.59	2.05
JPY	14.16	4.14	64.02	1.60	3.59	2.55
CAD	7.04	3.28	59.55	2.50	3.59	1.05
AUD	15.53	5.28	51.41	1.80	3.59	2.26
BRL	19.60	5.21	74.06	10.00	3.59	3.07
MXN	3.74	2.95	25.08	40.00	3.59	0.05
PLN	7.16	5.41	64.30	15.00	3.59	0.66
ZAR	10.93	6.55	22.96	55.00	3.59	1.12
Mean	**11.13**	**4.36**	**53.79**	**13.18**	**3.59**	**1.70**

Table 5.1: Results of most profitable system, returned as means of result per currency

profit per annum, and the USD/MXN is found to be the least profitable with an average annual profit of 3.74%. The USD/BRL is found to be the most statistically significantly profitable, with a T-value of 3.07 on comparison to a one-year US Treasury Bill. This is due to the huge returns and relatively low risk achieved by the USD/BRL.

The standard deviation is found to average of 4.36%, with the majors having an average standard deviation of 3.72%. This risk is lower than the minors' average standard deviation of 4.27% and the emerging markets 5.03%, and can be attributed to the majors having the largest volume traded. This causes non-market changes to have a smaller effect on trades, making the majors less volatile. The same concept can be applied when comparing the minors with the emerging markets.

The USD/MXN achieved the lowest profit of 3.74% but also at the lowest risk, with a standard deviation of 2.95%. This lower risk is specifically due to an increased signal tolerance produced by the genetic training function, which causes the USD/MXN to respond only to heavily positive or negative signals. This results in fewer transactions and greater stability in trading.

Table 5.2 describes the returns generated by the system, showing the average returns per period and the standard deviations of those returns for all currencies. The transaction costs for each currency, and the statistical significance of the returns when compared to a no-risk investment strategy of a one-year US Treasury Bill are also shown.

The time period during which the test took place also heavily affected the profitability of the system. Periods in 2007 and 2008 showed significantly more profit than those in the three years prior. A maximum of 19.5% average annual return was achieved for the year

Period	Returns (%)	STD (%)	Num Transactions	Tresury Bill (%)	T-Value on T-Bill
Jul-07 - Jun-08	18.99	3.80	63.81	3.59	4.05
Apr-07 - Mar-08	19.50	3.40	62.75	3.59	4.68
Jan-07 - Dec-07	18.95	2.65	77.86	3.59	5.81
Oct-06 - Sep-07	14.69	3.89	75.30	3.59	2.86
Jul-06 - Jun-07	12.01	5.36	71.51	3.59	1.57
Apr-06 - Mar-07	12.08	3.96	63.09	3.59	2.14
Jan-06 - Dec-06	10.38	3.62	38.37	3.59	1.87
Oct-05 - Sep-06	9.64	4.38	41.33	3.59	1.38
Jul-05 - Jun-06	7.38	4.22	49.42	3.59	0.90
Apr-05 - Mar-06	2.45	5.29	29.53	3.59	-0.21
Jan-05 - Dec-05	3.95	5.09	36.87	3.59	0.07
Oct-04 - Sep-05	3.57	6.65	35.66	3.59	0.00
Mean	**11.13**	**4.36**	**53.79**	**3.59**	**2.09**

Table 5.2: Results of most profitable system, returned as the mean values of result per period

between April 2007 and March 2008, with the least average annual return being 2.45% for the year between April 2005 and March 2006. Between these points there is a clear trend, with each successive time period being more profitable than the last.

The average number of transactions changes dramatically across the periods: from an average of 77.86 transactions per currency per year, in the period January through December 2007; to a mere 29.53 per currency per year, for the period April 2005 to March 2006 - coinciding with the lowest return of 2.45%. The reason for this reduced profitability is unclear and should be investigated further; although by examining at the standard deviations of the profits, it has been hypothesised that the foreign exchange markets have been more predictable in recent periods (April 2006 till June 2008), than they have been in the past (October 2004 till March 2006).

The profitability of the rules was seen to be significantly different (to a 1% level on average) from a no-risk Treasury Bill investment strategy, with the latest three periods being statistically significant to a 0.05% level.

The overall risk return scatter-graph 5.2 shows what seems to be an upward trend of risk to returns. This is in alignment with financial theory, which states that the greater the risk, the greater the returns [25].

In Figure 5.1, the majority of returns are between -5% and 30%, with risks between 0% and 12%. The USD/MXN pair showed the greatest stability, with a clustering around 4% return and 3% risk.

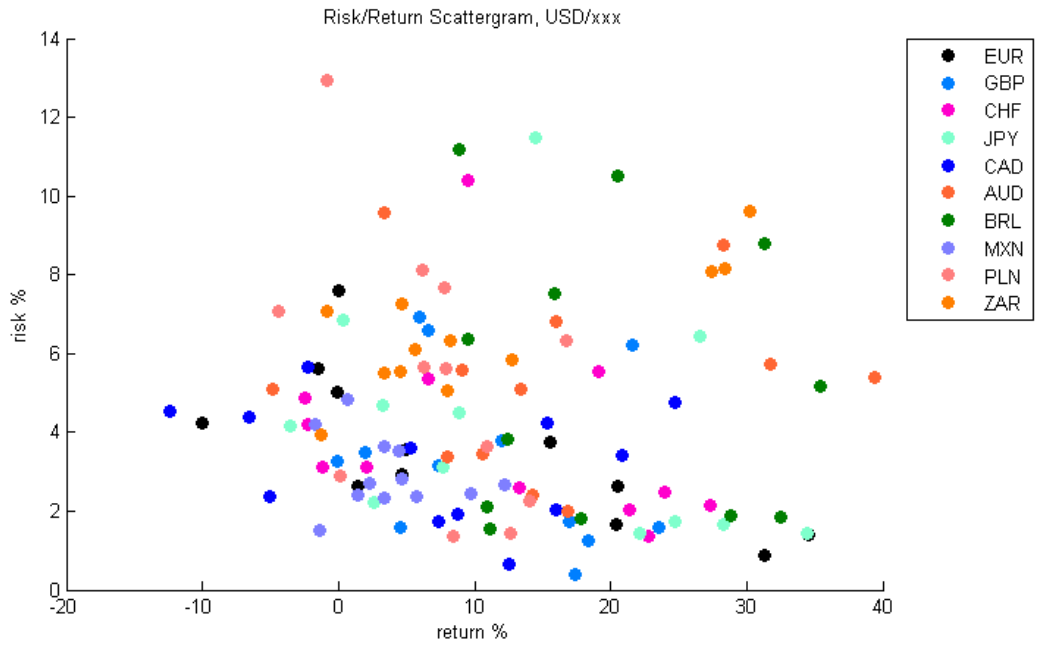


Figure 5.1: A Scattergram showing the relationship between risk and returns for all currencies over all periods

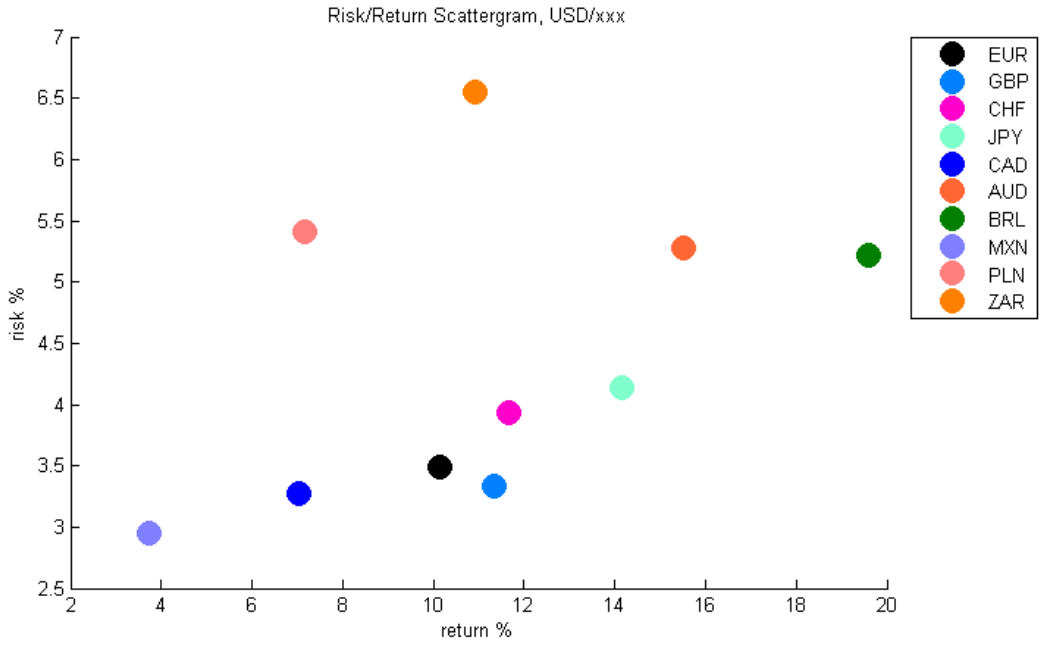


Figure 5.2: A Scattergram showing the relationship between the average risk and returns for all currencies

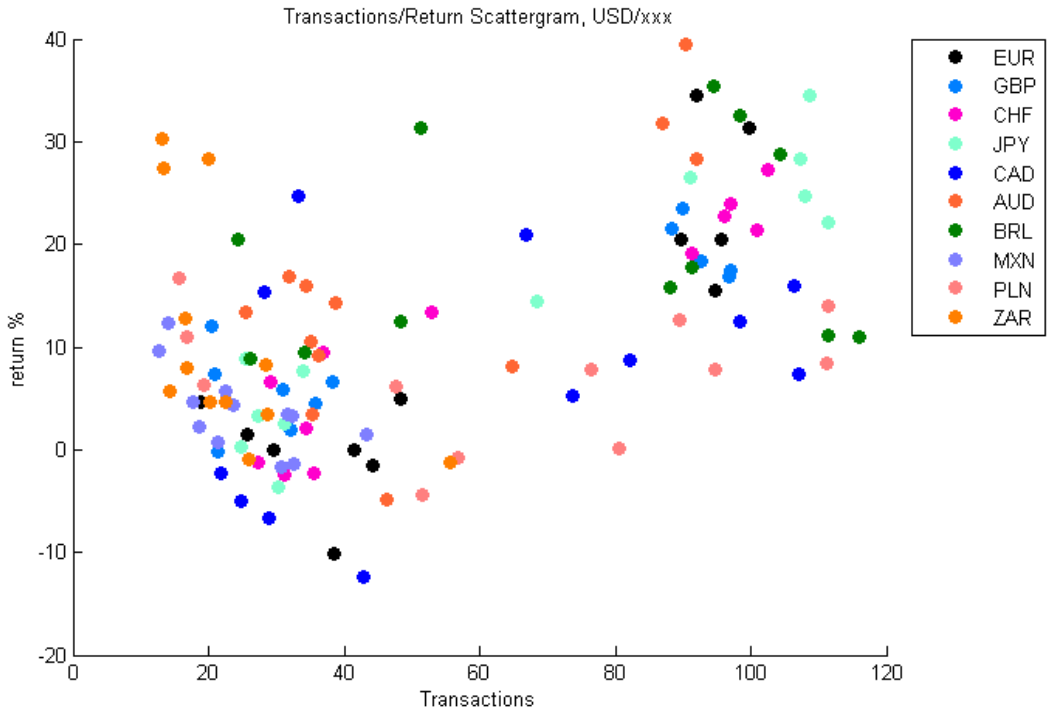


Figure 5.3: A Scattergram showing the relationship between the number of transactions and returns for all currencies over all periods

Figure 5.3 shows the number of transactions relative to returns for each currency and period. There appears to be an upward trend in this figure, showing that investment strategies which are more frequent traders, show higher returns.

5.1.1 Summary

By looking at these results, it is clear that some currencies and some periods are more profitable than others. In general, the majors are the most profitable and least risky currencies to trade in. The same is true for the data periods between October 2006 and June 2008.

There appears to be a positive linear relationship between risk and return, which is in alignment with financial theory [25], and a positive linear relationship between the number of transactions and returns, indicating that strategies which more frequent traders show higher returns.

In general the system is profitable to a statistically significant level of 5% when looking per currency, and 1% when looking per period - with the latest three periods being statistically

Currency	Intelligent Trader	Trend(5,20)	Filter(15)	RSI(5)	MACD(12,26,9)	MOM(12)
EUR	10.15	1.66	-4.15	3.55	-1.84	-2.47
GBP	11.34	6.52	4.62	2.60	9.35	2.22
CHF	11.67	-1.32	-8.09	0.90	3.03	-4.54
JPY	14.16	1.96	-0.36	-2.98	0.00	5.63
CAD	7.04	-4.80	3.15	2.90	-5.74	-4.31
AUD	15.53	11.01	8.22	-0.05	3.51	13.03
BRL	19.60	3.92	15.86	20.20	12.64	12.85
MXN	3.74	1.43	0.13	0.26	2.60	0.58
PLN	7.16	0.68	5.41	12.11	4.51	6.20
ZAR	10.93	-3.83	7.82	-7.50	2.76	-3.27
Mean	**11.13**	**1.72**	**3.26**	**3.20**	**3.08**	**2.59**

Table 5.3: Comparison between profits generated by the Intelligent System and those generated by un-optimised trading rules with parameters recommended in literature

significant to a 0.05% level.

5.2 Comparison with Existing Trading Strategies

In investigating the use of Intelligent Systems for Currency Trading Analysis, it is necessary to see whether the addition of intelligence, and the required computing power is justified when compared against the same trading rules with un-optimised values recommended by Park and Irwin [49].

Table 5.3 compares the intelligent system developed in Chapters 4 and 5 with un-optimised trading rules showing the average profit generated for each currency over all periods.

Trending analysis is performed with short and long window sizes of five and 20 days respectively; filter rules are applied with a window length of 15 days; RSI is used with a window length of 5 days; MACD is employed with window lengths of twelve, 26 and nine days; and MOM is tested with a window length of 12 days. These results are compared to the most significantly profitable system using the Simple ANN architecture, a population size of 20, and dynamic Gaussian mutation. The results of which are tabulated in Table 5.3.

It is clear from these results that the addition of intelligence dramatically improves results, and that the optimised combination of the trading rules has far greater returns than those of the individual components. The returns of the intelligent system are statistically

significant to the 1% level when compared to the un-optimised trading rules, with a T-Value of 2.01.

This proves that there is definitely merit in the added computational time and effort required to optimise the trading system.

5.2.1 Summary

The system performs statistically significantly better to a 1% level than the individual rules with parameters recommended in literature. This shows that there is merit in introducing AI, as it significantly improves the profitability of the system.

5.3 Summary

After optimal values had been found for the GA and ANN, the most statistically significant solution was investigated further. This chapter has shown that some currencies and periods are more profitable than others and in general, the more stable currencies and periods were the most profitable and significant. There are also positive linear relationships between risk and returns, as well as between the number of transactions per year and returns.

In general, the system is profitable to a statistically significant level of 5% when looking per currency, and 1% when looking per period - with the latest three periods being statistically significant to a 0.05% level.

Once it was established that the system is significantly profitable, an investigation into the value that AI contributes to this profitability was launched. It was found that the system performs better than any of the individual rules with parameters recommended in literature, to a statistically significant level of 1% .

Chapter 6

Conclusion

This thesis set out to build a system which intelligently combines signals generated by a number of optimised technical trading tools, to produce a single set of buy, hold and sell signals for the foreign exchange market. The intelligent combining of the signals was done with ANNs, and the optimisation was performed by GAs. The system attempted to overcome the criticisms of earlier tools, while still remaining significantly more profitable than a no-risk investment strategy. In so doing, the system used real-world transaction costs, as well as overnight interest rates (or closest proxy) to simulate a market in which traders shift between long and short positions.

Filter rules, trending, RSI, MACD and MOM were all implemented mechanically, producing fuzzy buy and sell signals. These signals were intelligently combined by an ANN, before being assessed by a simulation environment. The entire system was then optimised by a GA to ascertain the optimal parameters for the trading rules, as well as the weights and biases for the ANN, and the risk thresholds in the simulation environment.

Following the optimisation, the system was tested over twelve two-year periods of uniformly selected data from five years of asset price information for ten currencies. This was repeated 30 times, and the mean values of the repeats were taken into account for the stochastic nature of GAs. The system was presented with training data from 365 days, and tested on the following 365 days of data. This was done so as to address criticisms of data snooping.

6.1 Problem Statement Revisited

This study set out to achieve significantly profitable buy, hold and sell signals for the foreign exchange market through the use of technical analysis, while taking into account all previous criticisms (see Section 1.1).

In order to actualise these objectives, a number of intermediary goals were achieved. Technical trading tools were shown to be optimisable, making it possible to tailor the parameters of these tools to produce maximum profit on the given data set. GAs were shown to be effective optimisers in the context of technical analysis.

Optimal parameters for the GA were investigated, with results indicating that lowering the population size produced more generalised solutions, but with increased standard deviations. It was also shown that altering the mutation function of the GA has relatively little effect on the results.

ANNs proved to be good signal amalgamators, with overtraining being their only downfall. It was found that the more complex the ANN, the more specialised the resulting system; generating increasing profits on the training sets. This specialisation however, caused a decrease in profits in the test sets, as the resulting system was not finding general optimums, but rather unique and highly profitable solutions limited to the training set. Thus the least complex ANN turned out to be the most profitable, as it was unable to specialise as highly, and was thus a more general solution.

The optimal parameters identified in the above investigations were then compared to the un-optimised trading rules recommended by literature. The results of this comparison show that the addition of intelligence provides a statistically significant increase in the profitability of the system to a 1% level.

The general results for the optimal system were then investigated. It was found that the system was statistically significantly more profitable than a no-risk investment strategy - to a 5% level on average when looking per currency; and to a 1% level on average when looking per period, with the latest three periods (January 2007 to June 2008) being statistically significant to a 0.05% level.

Based on these results, it can be concluded that the project was a success, proving that it is indeed possible to create a profitable autonomous trading system.

6.2 Future Work

As the ANNs used in this thesis were found to be prone to overtraining, alternative types of ANNs should be investigated along with appropriate training methods. Some of these ANN types include Support Vector Machines, and dynamically structured Committee Machines, using Reinforcement Learning and other methods of unsupervised learning. Another option would be to use methods other than ANNs to amalgamate and classify signals. Examples include methods used for building fuzzy Knowledge Based Systems, and a non-linear combiner built and optimised using Genetic Programming techniques.

One other area that deserves attention is the identification of other suitable technical trading tools that can be used. To this end, principle component analysis should be performed on the Technical Trading Tools to see which tools are useful/useless. Following this investigation, the rule base should be restructured, and expanded to include more rules that have a similarly significant impact on profits. This includes creating new rules through the application of Genetic Programming techniques.

A further area that can be explored is potentially increasing the dimension of the data used by the system. Currently, the data presented to the system provides a single dimension of the current market. Therefore, means of extending the system to include other related information should be explored. For example, the use of a lexical analyser within the system could be investigated. The role of this analyser would be to pick up news headlines from RSS feeds, corresponding to a database of countries paired with currencies, and words with semantic understanding. This could serve as an indicator of possible future trends. Other dimensions such as prime interest rates, GDP, and inflation could be presented to the system as indicators. These indicators could be used to identify the future trends of an economy, and thus signal a strengthening or weakening of that economy's currency.

Further, the current system could be extended to trade on the best opportunity amongst all currencies. This could be accomplished by pitting all investment possibilities against one another. This allows the system to exploit the optimal transactions by trading between a number of currencies, instead of simply shifting between the long and short positions on a single currency.

Finally, it would be beneficial if the system was able to autonomously obtain the latest data from the Internet, and thus base its decisions on data it has collected itself. If this proves profitable, the system could be allowed to make its own trading decisions, and have access to one of the online trading APIs - thus completely automating the trading process, and creating a passive money-making machine.

References

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

[2] A.G. Barto, R.S. Sutton, and C.W. Anderson. Neuronlike elements that can solve difficult learning control problems. *IEEE Transactions on Systems, Man, and Cybernetics*, 13(5):834–846, 1983.

[3] Thomas Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press, USA, 1996.

[4] S. Becker. Unsupervised learning procedures for neural networks. *International Journal of Neural Systems*, 2(1):17–33, 1991.

[5] D.P. Bertsekas and J.N. Tsitsiklis. *Neuro-Dynamic Programming*. Athena Scientific, 1996.

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

[7] DS Broomhead and D. Lowe. Radial basis functions, multi-variable functional interpolation and adaptive networks. 1988.

[8] GA Carpenter and S. Grossberg. The art of adaptive pattern recognition by a self-organizing neuralnetwork. *Computer*, 21(3):77–88, 1988.

[9] T. Chenoweth and Z. Obradović. A multi-component nonlinear prediction system for the s&p 500 index. *Neurocomputing*, 10(3):275–290, 1996.

[10] N.L. Cramer. A representation for the adaptive generation of simple sequential programs. *Proceedings of the 1st International Conference on Genetic Algorithms table of contents*, pages 183–187, 1985.

[11] G. Cybenko. Approximation by superpositions of a sigmoidal function. *Mathematics of Control, Signals, and Systems (MCSS)*, 2(4):303–314, 1989.

[12] C. Darwin. The origin of species. 1859.

[13] K.A. De Jong. Analysis of the behavior of a class of genetic adaptive systems. 1975.

[14] E.F. Fama. Efficient capital markets: A review of theory and empirical work. *Journal of Finance*, 25(2):383–417, 1970.

[15] E.F. Fama and M.E. Blume. Filter rules and stock-market trading. *Journal of Business*, 39(S1):226, 1966.

[16] L.J. Fogel, A.J. Owens, and M.J. Walsh. *Artificial Intelligence Through Simulated Evolution*. John Wiley & Sons Inc, 1966.

[17] L.M. Fu. *Neural Networks in Computer Intelligence*. McGraw-Hill, Inc. New York, NY, USA, 1994.

[18] C. Fujiki and J. Dickinson. Using genetic algorithms to generate lisp source code to solve prisoner's dilemma, genetic algorithms and their applications. *Proceedings of the Second Int. Conf. on Genetic Algorithms. Cambridge, MA: Lawrence Erlbaum Associates*, pages 236–240, 1987.

[19] D.E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc. Boston, MA, USA, 1989.

[20] A. Goodacre and T. Kohn-Speyer. Crisma revisited. *Applied Financial Economics*, 11(2):221–230, 2001.

[21] S. Haykin. *Neural Networks: A Comprehensive Foundation*. Prentice Hall PTR Upper Saddle River, NJ, USA, 1994.

[22] DO Hebb. *The Organization of Behavior*. John Wiley & Sons Inc., 1949.

[23] J.H. Holland. *Adaptation in natural and artificial systems*. University of Michigan Press Ann Arbor, 1975.

[24] JJ Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences of the United States of America*, 79(8):2554–2558, 1982.

[25] PGA Howells and K. Bain. *The Economics of Money, Banking and Finance: A European Text*. Financial Times/Prentice Hall, 2005.

[26] Lakhmi C. Jain and N.M. Martin. *Fusion of Neural Networks, Fuzzy Sets, and Genetic Algorithms: Industrial Applications*. International Series on Computational Intelligence. The CRC Press, 1999.

[27] N. Jegadeesh and S. Titman. Cross-sectional and time-series determinants of momentum returns. *Review of Financial Studies*, 15(1):143–157, 2002.

[28] M.C. Jensen and G. Bennington. Random walks and technical theories: Some additional evidence. *Journal of Finance*, 25(2):469–482, 1970.

[29] H. Kitano. Designing neural networks using genetic algorithms with graph generation system. *Complex Systems*, 4(4):461–476, 1990.

[30] T. Kohonen. The self-organizing map. *Proceedings of the IEEE*, 78:1464, 1990.

[31] John R. Koza. *Genetic Programming: On the programming of computers by means of natural selection*. Sventh Printing, 1992.

[32] R.M. Levich and L.R. Thomas. The merits of active currency risk management: Evidence from international bond portfolios. *FINANCIAL ANALYSTS JOURNAL*, 49:63–63, 1993.

[33] R. Levy. Relative strength as a criterion for investment selection. *Journal of Finance*, 22(4):595–610, 1967.

[34] R. Linsker. Self-organization in a perceptual network. *Computer*, 21(3):105–117, 1988.

[35] B.R. Marshall, J.M. Cahan, and R.H. Cahan. Is the crisma technical trading system profitable? *Global Finance Journal*, 17(2):271–281, 2006.

[36] W.S. McCulloch and W. Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biology*, 5(4):115–133, 1943.

[37] GF Miller, PM Todd, and SU Hedge. Design neural networks using genetic algorithms. *Proceedings of the Third International Conference on Genetic Algorithms*, pages 379–384, 1989.

[38] M. Minsky. Steps toward artificial intelligence. *Proceedings of the IRE*, 49(1):8–30, 1961.

[39] M.L. Minsky and S. Papert. *Perceptrons: an introduction to computational geometry*. MIT Press Cambridge, Mass, 1969.

[40] M. Mitchell. *An Introduction to Genetic Algorithms*. Bradford Books, 1996.

[41] D.J. Montana and L. Davis. Training feedforward neural networks using genetic algorithms. *Proceedings of the Eleventh International Joint Conference on Artificial Intelligence*, 123, 1989.

[42] D. Nauck, F. Klawonn, R. Kruse, and F. Klawonn. *Foundations of Neuro-Fuzzy Systems*. John Wiley & Sons, Inc. New York, NY, USA, 1997.

[43] C.J. Neely, P. Weller, R. Dittmar, and Centre for Economic Policy Research (Great Britain). *Is Technical Analysis in the Foreign Exchange Market Profitable? A Genetic Programming Approach*. Centre for Economic Policy Research, 1996.

[44] OANDA. Currency histories. Online: `http://www.oanda.com/convert/fxhistory` Accessed: 06/08/2008, 2008.

[45] T. Oberlechner. Importance of technical and fundamental analysis in the european foreign exchange market. *International Journal of Finance & Economics*, 6(1):81–93, 2001.

[46] J. Okunev and D. White. Do momentum-based strategies still work in foreign currency markets. *Journal of Financial and Quantitative Analysis*, 38(2):425–447, 2003.

[47] OnlineTradingConcepts.Com. Technical analysis. Online: `http://www.onlinetradingconcepts.com/TechnicalAnalysis` Accessed: 16/09/2008, 2008.

[48] S. Papadamou and G. Stephanides. Improving technical trading systems by using a new matlab-based genetic algorithm procedure. *Mathematical and Computer Modelling*, 46(1-2):189–197, 2007.

[49] C.H.O. Park and S.H. Irwin. The profitability of technical analysis: A review. *Urbana*, 51:61801, 2004.

[50] M.J. Pring. *Introduction to Technical Analysis*. McGraw-Hill, 1998.

[51] S.W. PRUITT, KS MAURICE TE, and R.E. WHITE. The crisma trading system: The next five years. *Journal of Portfolio Management*, (SPRING 1992), 1992.

[52] S.W. Pruitt and R.E. White. The crisma trading system: who says technical analysis can't beat the market. *Journal of Portfolio Management*, 14(3):55–58, 1988.

[53] I. Rechenberg. Cybernetic solution path of an experimental problem. royal aircraft establishment. *Library Translation*, 1122, 1965.

[54] I. Rechenberg. Evolutionsstrategie. *Frommann-Holzboog, Stuttgart*, 1973.

[55] Russell D. Reed and Robbert J. Marks. *Neural Smithing: Supervised Learning in Feedforward Artificial Neural Networks*. Bradford Books, 1999.

[56] F. Rosenblatt. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol Rev*, 65(6):386–408, 1958.

[57] D.E. Rumelhart, G.E. Hintont, and R.J. Williams. Learning representations by back-propagating errors. *Nature*, 323(6088):533–536, 1986.

[58] D.E. Rumelhart and J.L. McClelland. Parallel distributed processing: explorations in the microstructure of cognition: foundations. *Mit Press Computational Models Of Cognition And Perception Series*, 1:547, 1986.

[59] M.R.E. Shazly and H.E.E. Shazly. Forecasting currency prices using a genetically evolved neural network architecture. *International Review of Financial Analysis*, 8(1):67–82, 1999.

[60] R.S. Sutton and A.G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[61] R.J. Sweeney. Beating the foreign exchange market. *Journal of Finance*, 41(1):163–182, 1986.

[62] M.P. Taylor and H. Allen. The use of technical analysis in the foreign exchange market. *INTERNATIONAL LIBRARY OF CRITICAL WRITINGS IN ECONOMICS*, 143:379–389, 2002.

[63] Thomson-Reuters. Thomson datastream. Online: `http://www.thomsonreuters.com/products_services/financial/datastream` Accessed: 20/09/2008.

[64] J.C. Van Horne and G.G.C. Parker. The random walk theory: an empirical test. *Financial Analysts Journal*, 23:87–92, 1967.

[65] J.C. Van Horne and G.G.C. Parker. Technical trading rules: A comment. *Financial Analysts Journal*, 24(4):128–32, 1968.

[66] DJ Willshaw and C. von der Malsburg. How patterned neural connections can be set up by self-organization. *Proceedings of the Royal Society of London. Series B, Biological Sciences*, 194(1117):431–445, 1976.