University College London

"Profitable, Return Enhancing" Portfolio Adjustments – An Application of Genetic Programming with Constrained Syntactic Structure

MSc Computer Science Project 2002/2003

By Wei Yan

Supervisor: Chris Clack

This report is submitted as part requirement for the MSc Computer Science degree in the Department of Computer Science at University College London. It is substantially the result of my own work except where explicitly indicated in the text. The report will be distributed to the internal and external examiners, but thereafter may not be copied or distributed except by permission of the author.

1. Introduction

This project derives from a problem submitted by an asset management organization wishing to explore if evolutionary programming could contribute a decision support system to exploit the possibility of using technical analysis, in order to enhance the returns of "indexed portfolios".

Technical analysis uses price and related data to decide when to buy and sell stocks. An indicator is a mathematical calculation that can be applied to a security's price and/or volume fields. The result is a value that is used to anticipate future changes in prices.

Evaluating, improving, and discovering new rules is a field that is well suited for evolutionary computation techniques. And, accordingly, there is large body of literature covering the application of technical analysis using genetic algorithm or genetic programming. However, most of the literature deals with cases in which technical analysis is used only in the context of trading single instruments, there are much fewer references addressing the problem of how to exploit "technical analysis" to enhance the return of portfolios. In fact, I was not able to find references relating to the specific problem addressed in this project:

Is it possible to find technical indicators that instead of giving buy and sell signals for single instruments can trigger portfolio adjustments able to generate an excess return over the benchmark alternative of a portfolio totally indexed on the S&P 500?

To address the problem we have formulated a process to link scores, based on technical indicators, to portfolio adjustments defined as incremental overweighting or underweighting of stocks with extreme scores. Just as technical rules give buy and sell signals for individual instruments, here scores give overweighting and underweighting signals for certain stocks.

We are able to take into account the fact that we deal with a portfolio by formulating the overweighting and underweighting process in such a way that the portfolio constraints are observed at all times.

By defining the criterion of success or failure of the rule in terms of the portfolio returns obtained after the adjustments compared to the return of the benchmark alternative we have cast the problem in terms that lend themselves easily to the application of a GP framework.

Most of the cases of GA/GP applications to financial technical analysis are defined in terms of the search for indicators or combination of indicators that generate excess returns over the baseline alternative of buy and hold. Here we search for a combination of indicators, yielding a score which determines portfolio adjustments generating incremental returns over the baseline alternative of the benchmark portfolio.

In this pilot implementation, we have selected the option of defending against the delusions of overfitting and accordingly limited the search space for the Scoring rule to the family of weighted linear combinations of well known and well proven indicators.

There is a clear goal that can be defined for this project:

- Deliver a pilot system illustrating the use of technical factors to make "profitable, return enhancing" portfolio adjustments.

This report will begin with a quick introduction to technical analysis and an in-depth review on genetic programming technique and its applications in the field of finance. Analysis of the problem and construction of an analytical framework for the system design will then be considered. The design of the solution will include a detailed description of the particular genetic programming system that will be used to select trading indicators. The particular implementation issues will then be discussed before looking at both in-sample and out-of-sample results.

2. Background

The domain experience, especially with respect to the description of real life business practices, referred to in this project relies considerably on information communicated by the sponsor and on interviews with practitioners .As the problem handled in this project is partly based on the characteristics of a real life portfolio there is a fair amount of domain experience which impacts the definition of the problem as well as the formulation of the analytical framework.

2.1 Technical Analysis

Stock market prediction analysis can generally be classified either fundamental or technical. The former approach considers the cause of market behaviour, while the latter studies the effect. Technical analysis is based only on quantifiable market data. This paper has focused solely on technical analysis technique.

Technical stock analysis is based on three basic principles (Murphy, 1986), namely:

- Market action discounts everything;
- Price move in trends;
- History repeats itself.

Technical analysis is often involves technical indicators, which are indices formed from combination of current and past price data. This project uses 3 types popular indicators, namely moving averages, Momentum (MACD) and Relative Strength Index (RSI).

2.2 Genetic Programming Review

Genetic Programming (GP) is a technique pioneered by John Koza (Koza 1992) which enables computers to solve problems without step by step, instructions. It is one of a wide range of Evolutionary Computation (EC) techniques and a descendant of John Holland's genetic algorithms (GA). GP (and GA) is one of several problem solving methods based on computational analogy to natural evolution.

GPs automatically generate computer programs. The theory states that there is no need to know anything about the problem one is trying to solve, as long as there is a "black box" which evaluates the solutions proposed. However, in practice the use of these methods should not be a substitute for thought, since taking into account of problem specific knowledge can considerably improve the effectiveness of this methodology.

2.2.1 Evolutionary Computation

The natural world abounds with structures of incredible complexity and apparent cleverness. Since the time of Darwin, it has been increasingly clear that these structures did not just happen, but rather evolved. The concept of natural selection or survival of the fittest has been seen to be a very powerful paradigm, given the existence proof of the biological world (Koza 92).

Any solution of a problem can be conceptualised as a search through the space of all potential solutions. Evolution can also be seen as a search process. This search process consists of a few possibilities, deciding how "fit" they are and using the information to decide which others to try next. The procedure is repeated until an individual is found that solves the problem, i.e. has a high enough "fitness".

2.2.2 Genetic Algorithms (GA)

The original work that led to Genetic Programming was Genetic Algorithms. The GA, pioneered by Holland (Holland 75), is a highly parallel mathematical algorithm that transforms a population of individual objects each with an associated fitness value, into a new generation of population. It uses the Darwinian principle of reproduction and survival of the fittest and naturally occurring genetic operations such as *crossover* (sexual recombination which is in fact the principal genetic operation) and mutation. Each *individual* in the population represents a possible solution to a given problem. The GA attempts to find a very good or best solution to the problem by genetically breeding the population of individuals.

GAs frequently operate on fixed length character strings, often binary, as the structure undergoing adaptation. Fitness is determined by executing task specific routines and algorithms using an interpretation of the character string as the set of parameters.

Crossover is the principle genetic operator employed, with mutation included as an operator of secondary importance. The three dominant different approaches of crossover are *one-point crossover, two-point crossover and uniform crossover*. With one-point cross-over one value representing the position in the string is chosen at random, and the values of the bit string are interchanged to one side of particular point. With two-point crossover two positions representing the positions in the string are chosen at random, and the values of the bit strings in-between the two values are interchanged. In uniform crossover individual vector positions between parents are swapped with a 50% probability. It must be pointed out that individuals are selected for recombination almost always based on their fitness. Figure 2.1 shows an example of the results that each of the operations produces.

Individuals:	A:	A0	A1	A2	A3	A4	A5	B:	B0	B1	B2	B3	B4	B5
Results:														
One point crossover (at position 2):														
	C:	A0	A1	B2	B3	B4	B5	D:	B0	B1	A2	A3	A4	A5
Two-point crossover (at position 2 and 4)														
	C:	A0	A1	B2	B3	A4	A5	D:	B0	B1	A2	A3	B4	B5
Uniform crossover (not unique)														
	C:	B0	B1	A2	A3	A4	B5	D:	A0	A1	B2	B3	B4	A5

Figure 2.1: Example of different crossover mechanisms

Besides crossover, other genetic operations are used to create offspring from parents. Allele mutation, usually called bit-flipping mutation when working on binary vectors modifies a single position in the parent vector by assigning it a new random value from the appropriate range for the feature.

In preparing to use the conventional GA operating on fixed length character strings to solve a problem, the user must determine: the representation scheme, the fitness measure, the parameters and variables for controlling the algorithm, and finally a way of designating the result and a criterion for terminating a run.

Specification of the representation scheme starts with a selection of the string length L and the alphabet size K. The fitness measure assigns a fitness value to each possible fixed-length character string in the population. The primary parameters for controlling the GA are the population size, M, and the maximum number of generations to be run. Each run of the GA requires specification of a termination criterion for deciding when to terminate a run and a method of result designation.

2.2.3 Genetic Programming (GP)

GP is an offshoot of GA, but in some ways it is more of a generalisation rather than a specialisation of its parent discipline, since it is more expressive. Expressiveness is achieved, due to the nature of the representation of a GP, a computer program as opposed to "fixed" length bit string in conventional GAs.

GPs mimicks the principles that govern nature into computer programs. These programs have the flexibility needed to express the solutions to a wide variety of problems, it is an executable genetic material. Also, computer programs can take the size, shape and structural complexity necessary to solve problems. This is because its non-linear tree structure genetic material can very in length. Finally, computer programs have a way to solve the problem of program induction through iterations and controlled recursion (Clack and Yu, 1997).

2.2.3.1 Structure Undergoing Adaptation – Algorithm

Intuitively, if two computer programs are somewhat effective in solving a particular problem, then some of their parts probably have some merit. By recombining randomly chosen parts of somewhat effective programs, we may produce new computer programs that are even fitter in solving the problem. At each stage of this highly parallel, locally controlled, decentralised process, the state of the process will consist only of the current population of individual programs (Benzhaf et al, 98).

The structures that undergo adaptation in GP (the actual computer programs) are active. They are not passive encoding of the solution to the problem as with simple GA. Instead, given a computer on which to run, the structures in GP are active structures that are capable of being executed in the current form. The following steps, from Koza's book, define the algorithm:

1. Generate an initial population of random compositions of the functions and terminals of the problem (computer programs).
2. Iteratively perform the following sub-steps until the termination criterion has been satisfied:

(a) Execute each program in the population and assign it a fitness value according to how well it solves the problem.

(b) Create a new population of computer programs by applying the following two primary operations. The operations are applied to computer program(s) in the population chosen with a probability based on fitness

i. Copy existing computer programs to the new population.

ii. Create new computer programs by genetically recombining randomly chosen parts of two existing programs.

3. The best computer program that appears in any generation (i.e. best so far individual) is designed as the result of GP. This result may be a solution (or an approximate solution) to the problem.

2.2.3.2 Functions and Terminals

The size, shape and contents of these computer programs can dynamically change during the process. The set of all possible structures in GP is the set of all possible compositions of functions that can be composed recursively from the set of *N* functions from $F = \{f_1, f_2 \ldots, f_n\}$ and the set of N terminals from $T = \{ a_1, a_2 \ldots, a_n\}$.

Function and terminal set should be selected so as to satisfy the requirements of *closure* and *sufficiency*.

Closure (Koza 1992) states that any function should be well defined and closed for any combination of arguments that it may encounter. The boundary conditions for the functions are very important. What do the functions do when handed data that is illegal or in some way out of range? *Typing* is one way out of this dilemma, where each node carries its type as well as the types it can call, thus forcing function calling it to cast the argument into the appropriate type.

Sufficiency (Koza 1992) property requires that the set of terminals and the set of primitive functions are capable of expressing a solution to the problem.

2.2.3.3 Initial Structure – First Population

Each program individual can be expressed as a tree. Figure 2.2 shows two examples of such programs expressed as trees. Koza (1992) defined two methods termed *full* and *grow*.

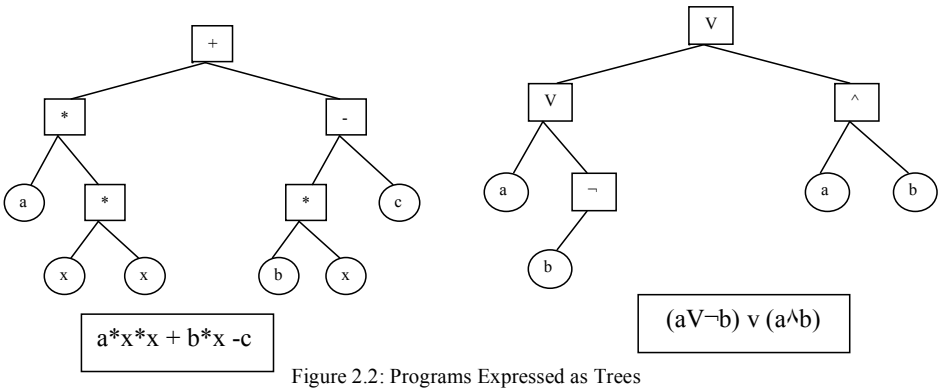


Figure 2.2: Programs Expressed as Trees

The *full* method works by selecting nodes from only the function set until a node is at a specified maximum depth when it then switches to selecting only terminals. Hence every branch reaches the maximum depth, creating more regular-shaped trees.

The *grow* method randomly selects nodes from the function and terminal set which are added to a new individual, until a terminal node is reached, which terminates that branch.

2.2.3.4 Fitness

The fitness is the driving force of GP. It describes how the system will evolve. A fitness function needs to rate an individual on how well it performs on the problem to be solved. It also needs to rate individuals in such a way that they can be compared and more successful individuals are distinguished from less successful ones. Fitness can take a number of different forms.

The *raw fitness* describes the value that is assigned directly relating to the problem. It is quite common that for raw fitness to be described as an *error* measure.

The *adjusted fitness* represents the fitness of an individual as a value between 0 and 1, where 1 is the ideal fitness.

$$adjusted\_fitness = 1 / (1 + raw\_fitness)$$

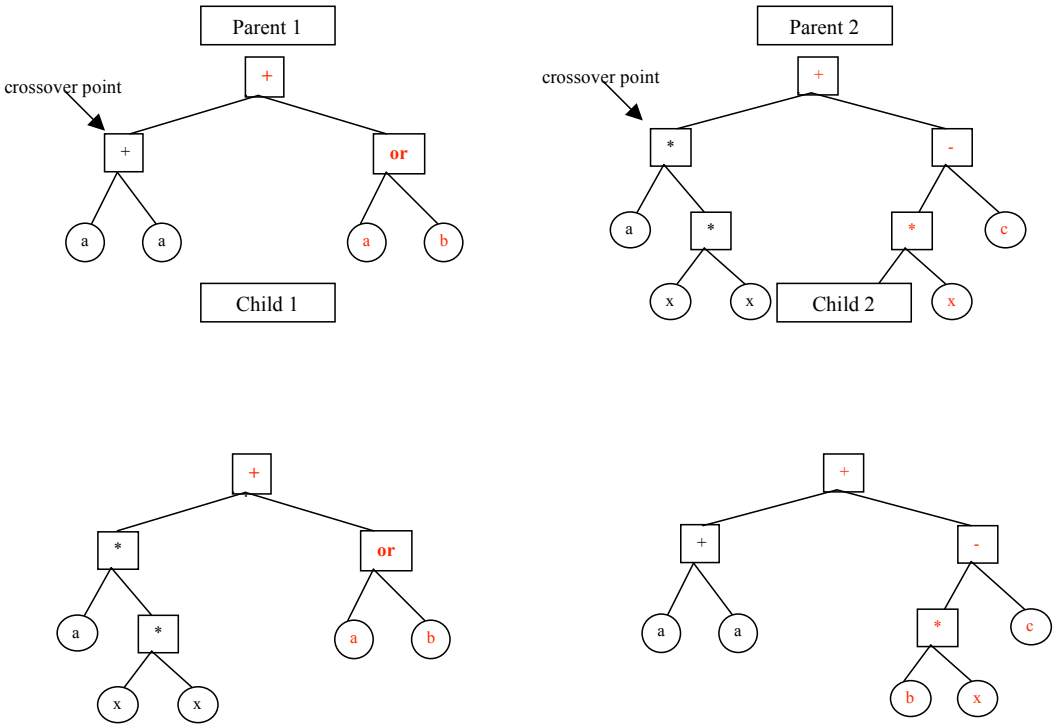
The “1+” is found in the denominator in order to avoid a divide by zero situation. This will also have the effect of increasing the difference between two individuals who are at the higher end of the fitness scale.

2.2.3.5 GP operators

GP shares many similarities with the broader domain of evolutionary algorithms, in particular in using crossover, mutation, and selection operators in order to generate novel evolved structures.

- GP Crossover

Crossover creates new offsprings that consist of parts taken from each parent. It is a sexual operation in that there are two parents combining together. Both individuals are selected using fitness. Then using a uniform probability distribution, one random point from each parent is selected to be the crossover point. Two offspring are then produced.



- **Mutation**

Mutation is often added alongside crossover in order to prevent premature convergence by introducing diversity back into population.

- **Selection Operators**

The *fitness proportionate selection* (Koza 1992) is the most popular selection method which uses adjusted or normalised fitness values of the individual to decide which one to choose.

Rank selection (Koza 1992), where each individual in the population is assigned a rank, hence sorted (from 1 to the size of the population). Selection is then performed so that the best individual receives a predetermined multiple of the number of copies that the worst individual receives.

Tournament selection (Kinnear 1993) does not require a centralised fitness comparison between all individuals. A continuous interaction of individuals takes place with the normal mechanisms of crossover and mutation, with new offspring replacing losers in the tournament process.

2.2.3.5 Control parameters

The finale consideration when looking at using a genetic program is the parameters that we should use to run the program. Some such parameters are the population size, the number of generations and the depth of the parse tree that are used to represent the programs. Each of these should be chosen in a problem specific way. Typically, a population of around 4 – 500 should be used for most problems to allow a large enough diversity in the initial set of individuals. Many problems will require more than this depending of their complexity. The number of generation is typically around 50, this has been sufficient to arrive at a global optimum in most of the problems that use genetic programming. The depth of the tree structure also depends on the complexity of the problem and also the resources available, such as memory. There are no rules to describe what parameters should be used and often it is simply best to try different options and see which best suits a particular problem. It can be considered something of a creative exercise.

2.3 Evolutionary Computation (EC) in Finance

Applications of EC techniques in finance, more specifically GA and GP, are a relatively new development compared to the EC achievement in the field of engineering and applied mathematics. Bauer (1994) pioneered to use GA to model trading strategies more than twenty years after Holland invented GA techniques in 1970's (This work is discussed in details below.) In 1992, Koza published his book "Genetic Programming – On the Programming of Computers by Means of Natural Selection". His work inspired a series of financial application of genetic programming in recent years. Chen and Yeh (1996a) applied genetic programming to "test" the efficient market hypothesis in a financial time series. Keber (1999) showed that genetically determined formulas outperformed most frequently quoted analytical approximations in calculating the implied volatility based on the Black-Schole model. Chidambaran et al. (2000a) and Keber (2000) derived approximations for calculating option prices and showed that GP-model outperformed various other model presented in the literature. In the domain of trading rules or strategies induction, two important work were Neely and al. (1997) and Karjalainen (1994). They adopted a GP approach to discover profitable technical trading rules for the foreign exchange rates and stock market index (S&P 500) respectively. The work of Neely and al. (1997) and Karjalainen (1994) are presented in greater detail below. The results obtained by their research were encouraging and their line of research has been further explored by many researchers. For example, Dempspter and Jones (2000) developed a trading system based on broad range of indicators for FX trading. Becker & Seshadri (2003) used GP to evolve trading rules by simplifying representations used by Allen and Karjakainen for the S&P 500 index. Li and Tsang (1999) attempted to use GP to improve technical rule predictions on historical Dow Jones Average (DJIA) index data.

It is worthwhile noting that, despite the fact of GP-evolved rules being extensively applied either to stock market index or single instrument (either exchange rate or single stock price), applying these genetically selected trading rules to portfolios remains unexplored.

2.3.1 GA Pioneer Work

Bauer (1994) pioneers the application of genetic algorithms to the development of trading strategies. He used GA to encode trading strategies as binary strings. Each trading strategy encoded by GAs has the following form:

(**IF** (CONDS)
THEN (BUY)
ELSE (IF (CONDS)
THEN (SELL)
ELSE (HOLD)))

The CONDS appearing in the trading strategy is a predicate. CONDS itself is a logical composition of several primitive predicates. In Bauer's application, all CONDSes are composed of three primitive predicates. Each primitive predicate can be represented as:

$$Cond\ (Z) = \begin{cases} 1\ (True),\ if\ Z\ \{>=, <\}\ z \\ 0\ (False),\ if\ Z\ not\ \{>=, <\}\ z \end{cases}$$

where Z can be an economic variable, such as unemployment rate. z can be regarded as threshold or critical value (z □ N). The following is an example of a specific trading strategy given by Bauer:

> IF either the Treasury bill rate is less than z1 OR the yield spread between long term Treasury bonds and Treasury bills is less than z2 and the yield spread between corporate bonds and Treasury bonds is less than z3, then invest in common stocks. If not, then invest in Treasury bonds.

This trading strategy is an example of CONDS with three primitive predicators:

CONDS(Z1, Z2,Z3) = (Cond(Z1) OR Cond(Z2)) AND Cond(Z3),

Z1 denotes the Treasury bill rate, Z2 denotes the yield spread between long term Treasury bonds and Treasury bills, and Z3 the yield spread between corporate bonds and Treasury bonds.

A CONDS is encoded by a 21-bit string (3 for logical combinations, 3 for inequalities, 15 for the three thresholds). Since each trading strategy is composed of two CONDSes, it can be represented by a 42-bit string. In this case, the search space D is the collection of all trading strategies encoded as above. Hence, the cardinality of D is

2^42, which is more than 0.1 trillion. This large search space gives the motivation to use GA.

In a GA run, Bauer used a population size of 100, one point crossover with a probability of 0.6, mutation rate 0.001, and 300 generations. Fitness measure was the US dollars value of the assets at the end of the 5-year testing period. The GA did identify profitable trading rules in the training run. However, the performance of these rules in subsequent period (out-of-sample) was "somewhat mixed" (Bauer 94).

Genetic Programming Work

Karjalainen (1994) used genetic programming to infer technical trading rules from past prices. Karjalainen used daily data for the S&P500 index from 1929 through 1995. He divided up the data into successive in-sample periods, consisting of 7 years, further sub-divided into 5 years training and 2 years for selection for each set of ten runs.

The operators used by AK included

1. Arithmetic operators: plus, minus, times, divide, norm, average, max, min, lag
2. Boolean operators: and, or, not, greater than, less than.
3. Conditional operators: if-then, if-then-else
4. Numerical constants and Boolean constants: true, false as terminals.

The fitness measure was the excess return on the data over the buy-and-hold. AK used a population of 500, with a size of 100 nodes and depth of 10, and with a maximum of 50 generations. The out-of-sample results were an average yearly excess return over buy-and-hold of 0.0205.

The distinguish point of this work is that it used a rigorous statistical test to evaluate the performance of GP, which affords a hard evidence for the relevance of GP or GAs to finance. Karjalainen's line of research has been further pursued by Oussaidene, Chopard, Pictet, and Tomassini (1996), and Neely, Weller and Dittmar (1997) in foreign exchange markets. The work of Neely et al. (1997) is particularly important, which not only gave a good introduction to GP but obtained promising results.

Neely et al. (1997) employed genetic programming to find technical trading rules for the following six exchange rates: dollar/Deutsche mark, dollar/yen, dollar/pound, dollar/Swiss franc, Deutsche mark/yen and pound/Swiss franc. There are 17 functions in their function sets, including 9 arithmetic operations, 5 Boolean operations and 2 conditional operations. Such a setting makes it easier to represent a sizable amount of trading strategies, including those commonly used by technical traders. Using their own word, " The advantage of the genetic programming approach is that it allows one to be relatively agnostic about the general form of optimal trading rule, and to search efficiently in a non-differentiable space of rules." (Neely et al. 1997)

The steps involved in the GP run are detailed below:

1. Create an initial generation of 500 random rules.
2. Measure the fitness of each rule over the training period and rank according to fitness.
3. Select the top-ranked rule and calculate its fitness over the selection period. Save it as the initial best rule.
4. Randomly select two rules, using weights attaching higher probability to more highly ranked rules. Apply the recombination operator to create a new rule, which then replaces an old rule, chosen using weights attaching higher probability to less highly ranked rules. Repeat this procedure 500 times to create a new generation of rules.
5. Measure the fitness of each rule in the new generation over the training period. Take the best rule in the training period and measure its fitness over the selection period. If it outperforms the previous best rules, save it as the new best rule.
6. Stop after 50 generations.

The annual excess returns averaged over all 100 rules for each currency obtained from 100 GP trials are all positive. The average across all currency is 2.87%. Four of the $/DM rules generated negative excess return.

3. Analysis And Design

3.1 Analysis

In this section, the material refers to business usages, the description of a portfolio manager decision process and formulation of portfolio adjustment reflect primarily sponsor and interview inputs which are transcribed here.

3.1.1 Problem Definition

There are many ways to try to achieve some "excess return" of portfolio over a benchmark index (like the S&P 500 or the FT 100) and each asset organization claims to have it's own proprietary way to do it. One of such ways is to attempt to use the resources of technical analysis. Although there is a fair body of literature covering the application of technical analysis to trading single instruments, there are much fewer references addressing the problem of how to exploit "technical analysis" to enhance the return of portfolios using a quantitative framework with a interpretable, transparent process which does not simply consist of a black box. The current project is an attempt to provide a "pilot" solution for a specific portfolio problem.

3.1.2 Guidance from Related Research

As indicated earlier in the literature survey there is a fair amount of empirical findings suggesting that technical indicators are indeed useful, contrary to what was once claimed by the devotees of the efficient market theory. As Rode et al. wrote in their Wharton School paper:

"This paper has shown three things: capital markets are not efficient and can be exploited for profit by technical analysis, technical analysis has a valid theoretical foundation and a testable formal theory, evolutionary computation represents a uniquely powerful way to test the theory presented here".

The literature survey found that the authors either start from a pragmatic standpoint (i.e. try to discover new or more profitable rules) or from a theoretical one (show the implication of the controlled application of trading rules for the validity or lack of it

the efficient markets theory).[1] However all case studies in the literature surveyed deal with a trading context in which rules are only exploited to trade single instruments. The guidance on the search and selection of rules for management of portfolios are much more difficult to find. As Rode et al. write in their concluding section devoted to suggestions for further research:

" Finally, it would be advantageous for traders to analyse multiasset portfolios with this methodology (i.e. metarules formed from more common indicators with the help of genetic algorithms). This would involve the incorporation of asset allocation analysis…"

In a certain sense this project is an attempt to follow Rode et al.'s suggestions by adapting a search for "useful" technical indicators to the context of portfolio management rather than to the trading of single instruments.

3.1.3 Relation between Technical Indicators and Portfolio Management

In the context of *portfolio decisions*, indicators do not in and of themselves tell what to do, as they do for instance in the context of *trading*.

Myname 7/9/03 1:12
Deleted: The project goal is to "discover" (and evaluate the merits of using) conventional technical indicators to "enhance" the returns of a portfolio of stocks with an S&P 500 benchmark.

In the context of trading, "buy-sell" signals given by indicators are sufficient to make decisions over any given time period. By following the signals, the trader will generate a return that can be compared to the return of all other trading rules or to a specific base line, like for instance, buy and hold.

But a portfolio manager, operating over a given universe of stocks, would not automatically know what to do even he was convinced he had reliable "buy-sell" signals for every element in his stock universe. Should he "buy" all the stocks for which there are buy signals, when the signal occurs? What does he do if different indicators give different signals for the same stock at the same time? And even when such obvious conflicts are resolved in one way or another, how will the manager invest the cash available to buy each stock for which there are buy signals? Should the

[1] discovering profitable trading rules is prima facie evidence against the validity of the theory of efficient markets and conversely the impossibility to discover such rules even with the help of the most powerful search algorithms.]

manager limit the total purchases to the amount of cash available or should cash be raised by selling stocks already in the portfolio? And if such is the decision, on what basis will the manager choose which stocks will be sold? Should the manager sell only these stocks for which there are "sell" signals or others as well?

In this context, the research approach, pioneered by AK(1994) followed by a number of recent researchers "may crudely be constructed as nothing but intelligent data fitting." (Rode et al, 1995) Instead, we are opting for an analytical approach which takes into account the domain knowledge and the problem specific features to narrow the search space.

The main difficulty is to formulate a process to make portfolio adjustments on the basis of technical analysis rules. As already indicated, the literature offers several examples of profitable trading rules for instruments like the S&P 500 and the US$/UK£ (e.g. see Allen and Neely).But we did not find a single reference dealing with the issue of how to allocate a given amount of risk capital to trading the two instruments.[2]

3.1.4 Role of Technical Indicators and Scoring System in Addressing the Problem

- **Synthetic Indicator**

We focus on the observation that the baseline portfolio is almost identical to a list of stocks "bought and held" according to constant proportions reflecting the composition of the benchmark. Therefore, if instead of keeping the whole list on "buy and hold", we buying more of some stocks and selling some of others, in accordance with profitable trading rules, the end return, would necessarily be better than the benchmark return, since at least for the stock we would have managed to trade the

[2] Indeed this may not be so surprising: the conventional approach to optimal portfolio allocation rests on the assumption that each financial instrument returns can be represented as a random distribution with given mean standard deviation and covariance with other financial instruments. Then maximisation of expected return subject to risk –with risk represented by standard deviation-yields the "optimal allocation" between the multiple assets. However representing financial instruments in this way implicitly assumes that trading rules must be useless –(one of the key implication of the random walk of stock prices or of the efficient markets theory). The relative silence of the theoretical literature on this issue is therefore not surprising.

return would be higher than the buy and hold. Since we must keep overall portfolio balance, we cannot simply sell all the stocks with strong sell signals and accordingly buy. However, if we manage to trade some stocks, subject to the other portfolio constraints, we should still be able to generate some excess return from that limited trading. Accordingly, we settle for a framework which consists of searching for a synthetic indicator ("metarule") that will score each stock in terms of the strength of buy and sell signals.

- **Scoring System**

Empirically asset managers often resort to scoring systems to rank their stocks. For instance a stock may receive a certain number of points based on momentum indicators another number of points for the strength of the balance sheet etc.[3]

GP in turn appears well suited to help in finding a system of scoring to address our problem in view of the following factors:

a) The technique has already been used successfully in the context of trading rules.

b) The formulation of portfolio adjustments as a function of scores yields a portfolio return, which can be used as the fitness criterion

[3] Indeed some well-known financial advisory letters, like Value Line, are based on supplying a ranking of stocks based on their proprietary scoring system. And in the related area of Bond and Credit Ratings such large businesses as the Moody and Standard & Poor's rating agencies are based on such a system of proprietary scoring.

3.1.5 Analytical Framework for Design

The following analytical framework are accordingly formulated:

- **Formulation of Portfolio Adjustments on the Basis of Scores**:

We compare the returns of the indexed portfolio with the return of a portfolio in which, every period, the weighting of the stock with the lowest score is reduced by a given percentage, thereby freeing cash which is then used to overweight by a given percentage the stock with the highest score.

The value of the portfolio from period to period changes only as a result of the changes in stock prices and of the adjustments in the quantities of each stock brought about by portfolio adjustments. In particular there are no cash "additions to" or "withdrawals from" the portfolio. Accordingly, the portfolio return for each period is defined simply as the % change in the value of the portfolio over the period divided by the value of the portfolio at the beginning of the period. Similarly, the return of the portfolio over any time interval is defined as the change in value between the beginning and the end of the interval divided by the initial value.

This portfolio return definition is very simple (and easier to code), than the time weighted return computation that would be required if cash movements in and out of the portfolio were allowed. The path of portfolio adjustments over time defines a time series of successive portfolios. In terms of GP representation, the computation of successive portfolios corresponds to the fitness calculation and the portfolio return to the fitness criterion.

- **Selection of a Metarule**

Fortunately, the literature gives a little more guidance for this task than for the previous one.

Given the two opposite possibilities to rely entirely on the GP to discover a scoring system or to start with proven indicators we adopted the later option following the

lead of Rode et al.(1995) who wrote "..(we use only)..rules which currently exist and have provable foundations. This model does not create technical rules–it creates 'metarules'." Using Rode et al. terminology, the combination of individual indicators based on scoring system is a 'metarule'.

There are many ways to combine individual indicators into a metarule. In keeping with the fear of "overfitting" we have limited the search to weighted linear combinations of indicators.

The universe of individual indicators is represented by common buy or sell indicators, which take as inputs past and current prices and return a value of 1 if they give a buy (respectively sell) signal and 0 otherwise. The metarule is then defined as a linear combination of individual indicators and returns a score equal to the sum of weights.

- **Statement of the Problem and GP Representation**

The pilot "solution" proposed consists of searching for linear combinations of widely used technical indicators. The search is conducted using GP. Each linear combination of indicators is an individual. Each individual determines unambiguously a time path of portfolios and their associated return. The fitness of each individual is measured by the associated portfolio return over the relevant time interval.

Myname 7/9/03 4:07
Deleted: The value of the portfolio from period to period changes only as a result of the changes in stock prices and of the adjustments in the quantities of each stock brought about by portfolio adjustments. In particular there are no cash "additions to" or "withdrawals from" the portfolio. Accordingly the portfolio return for each period is defined simply as the % change in the value of the portfolio over the period divided by the value of the portfolio at the beginning of the period. Similarly the return of the portfolio over any time interval is defined as the change in value between the beginning and the end of the interval divided by the initial value.
This portfolio return definition is very simple (and easier to code), than the time weighted return computation that would be required if cash movements in and out of the portfolio were allowed.
The path of portfolio adjustments over time defines a time series of successive portfolios. In terms of GP representation the computation of successive portfolios corresponds to the fitness calculation and the portfolio return to the fitness criterion.

- **Limitations Simplifications**

In formulating the problem some simplifying assumptions have been made and some features of « real life » operation have been ignored. They are summarised below:

1. It is assumed that there is never any gap between the closing price of one business day and the opening price of the next business day.
   In real life "gap" exists but they are exceptional for the universe of large cap blue chip stocks of the specific quasi-indexed portfolio of this project.[4]

[4] Furthermore, it would be relatively easy to take this feature into account by "coding" into the calculation of the fitness function (i.e. the calculation of the portfolio as modified by the adjustments) an "opening" and "closing" price vector for each period]

2. It is assumed that all the purchases and sales of stock can be done at the opening price.

3. In this pilot implementation transactions costs have also been ignored.
   Given the nature of the project portfolio the impact of those factors must be close to negligible.

4. The project considers only a limited data universe: 32 stocks and 85 time periods.

The restriction to the universe of 32 stocks is not as "unrealistic" as it may seem at first sight given the considerable amount of correlation in the movement of stock prices. For this project the "basket" component of this real life portfolio has supplied the 32 stocks names which are a very close proxy for the S&P 500 benchmark: over the life of the portfolio the basket performance has never diverged by more than ¼ of a percentage point from the behavior of the S&P index.

The time interval limitation is more serious as the length of time does not encompass enough market phases to fully assess the "relevance" of the methodology being explored.

Here the limitation was imposed by the time and resource constraints in the framework of this pilot project. Extending the time period and considering other stock universes are however one of the easiest developments to contemplate as the code can accept the required change in parameters to process data over a much longer time period or a broader universe of securities.

3.2 GP System Design

The following diagram 3.1 summarises the high level system design framework constructed in the Analysis part above.

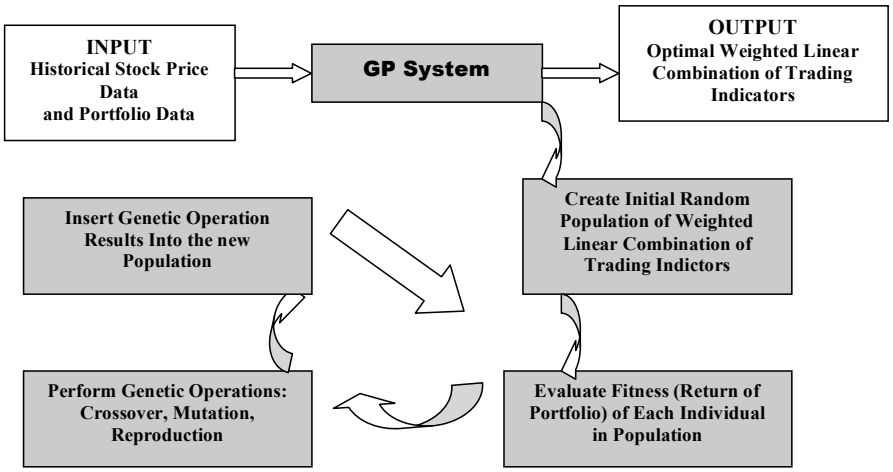


Figure 3.1 System Diagram.

Koza (1992) identified that there are five major steps in preparing to use genetic programming to solve a problem, namely determining:

1) the representation of an individual
2) the fitness function
3) operations for modifying structures – crossover and mutation, and
4) the criterion for designating a result and terminating a run.

These five steps are used as a framework for the discussion of the design of the GP system.

3.2.1 The Basics

Before describing how the genetic programming will discover optimal linear combination of trading indicators from a selected universe of trading rules, we will first consider the data variable environment over which individuals will be evaluated. The table below shows the eight financial rules that will be used. Each row of the data describes one company. 32 companies will be used in the environment. The data is supplied by eSignal and the data period is from 17th, April, 2003 to 27th, June.

Company	Current Price	50 days Moving Average	200 days Moving Average	RSI	MACD
1	59.23	58.28	50.33	40	0.21
2	43.80	45.78	47.89	24	0.98
..	…	…	…	…	…
32	…	…	…	…	…

3.2.2 Representation of an Individual

A function in the form of *weighted linear combination of indicators* is an individual which undergoes evolution in the GP system. An individual can be expressed as follows:

$$R_1 * W_1 + R_2 * W_2 + R_3 * W_3 + \ldots R_n * W_n$$

where R_1, R_2, … R_n are technical indicators. Buy indicators return a value of 1 (buy) or 0 (do nothing). Sell indicators return a value of 1 (sell) or 0 (do nothing) in function of stock price movement; and

W_1, W_2,.. W_n are the numerical weighting coefficient for the technical indicators.

This representation has two advantages: 1) It results into a numerical value representing the Score; 2) It has a fixed syntactic structure that can reduce the search space of GP to overcome the danger of "overfitting".

5/9/03 17:28
Formatted

5/9/03 17:48
Formatted

5/9/03 17:38
Formatted

5/9/03 17:38
Formatted

5/9/03 17:38
Formatted

5/9/03 17:38
Formatted

5/9/03 17:52
Deleted: **Trading Rules as Genetic Programs**

Myname 4/9/03 10:11
Formatted: Bullets and Numbering

3.2.2.1 A Score (CS) System Design

For the sake of simplicity, the CS can be considered as synthetic indicator of the "strength" of BUY versus SELL signals. Hence, it can be formulated as $CS = C_{buy} - C_{sell}$, where CS is overall Confidence Value, C_{buy} is the sum of weighted coefficient of the buy signals and C_{sell} the sum of weighted coefficient of the sell signals.

*CS = (BUY_Indicator_1*W1+BUY_Indicator_2*W1+…+BUY_Indicator_N*W1) – (SELL_Indicator1*W1 + SELL_Indicator_1*W1+…+SELL_Indicator_N*W1)*

3.2.2.2 An Individual is made of two trees: a BUY tree and a SELL tree

The scoring system described above implies that the GP individuals must consist of two distinctive parts: a BUY part and a SELL part. In other words, a GP individual contains a BUY tree and a SELL tree and the two trees evolve in parallel and independently.

A BUY tree consists of n $(0<n\leq5)$ indicators from five BUY technical indicators, $b \square \{ b_1, b_2, b_3, b_4, b_5\}$, and, similarly, a SELL tree consists of n $(0<n\leq5)$ indicators from five SELL technical indicators, $s \square \{ s_1, s_2, s_3, s_4, s_5\}$. BUY trees do not combine with SELL trees and vice versa.

3.2.2.3 Terminals

In this problem, the primary concern is the combinations of indicators and their corresponding weights. Therefore, indicators and weights are the information which the GP system needs to process, which is true for both BUY trees and Sell trees.

Hence, the terminal set (T_b) for a BUY tree is

T = { (BUY_RULE_1), (BUY_RULE_2), (BUY_RULE_3), (BUY_RULE_4), (BUY_RULE_5), (1), (2), (3), (4), (5), (6), (7), (8), (9), (10) },

where (1), (2), (3), (4), (5), (6), (7), (8), (9), (10) are numeric constants representing weights.

And the terminal set (T_s) for a SELL tree is

T = { (SELL_RULE_1), (SELL_RULE_2), (SELL_RULE_3), (SELL_RULE_4), (SELL_RULE_5), (1), (2), (3), (4), (5), (6), (7), (8), (9), (10) },

where (1), (2), (3), (4), (5), (6), (7), (8), (9), (10) are numeric constants representing weights.

3.2.2.4 Functions

The function set (*F*) chosen for BUY tree and SELL tree is identical:

F = { + , *}

where "+" and " *" are arithmetic operators. The data types for both the input arguments and the output arguments of the functions are of integer type. Both "+" and "*" take two input arguments, so that each GP tree is represented by a binary tree.

3.2.2.5 Syntactic Restriction on in Initial Population

In conventional GP, the individual computer programs undergoing adaptation are unrestricted composition of the available functions and available terminals. Therefore, the structure of a tree is not controlled and unpredictable because of randomness in the GP system. However, our goal is to find a function, in the form of a weighted linear combination of variables. Hence, the syntactic structure of GP trees is restricted. The initial population of random individuals must be created so that every individual computer program in the population has the required syntactic structure. And, because in this problem an individual is composed of a BUY tree and a SELL tree, the syntactic restriction needs to be applied to both trees in order to form syntactic valid individuals.

There are three types of points in either the BUY tree or SELL tree for this problem:

- points with weights of technical rules W ($0<W\leq10$),
- points with technical indicators R, and
- points with arithmetic functions ("+", "*")

The rules of construction for representing a valid trading strategy with one output Score are as follows: (rules are the same for both SELL and BUY trees)

- The root of the tree must be arithmetic function "+".

- The only two things allowed at the level immediately below arithmetic function "+" are itself or arithmetic function "*".
- The only thing allowed below the arithmetic function "*" on the left is technical rules R and on the right weights W.

These rules are applied recursively to constrain the tree to be a linear combination of the weighted indicators. Figure 3.2 graphically depicts the structure of an individual with ordered branches.

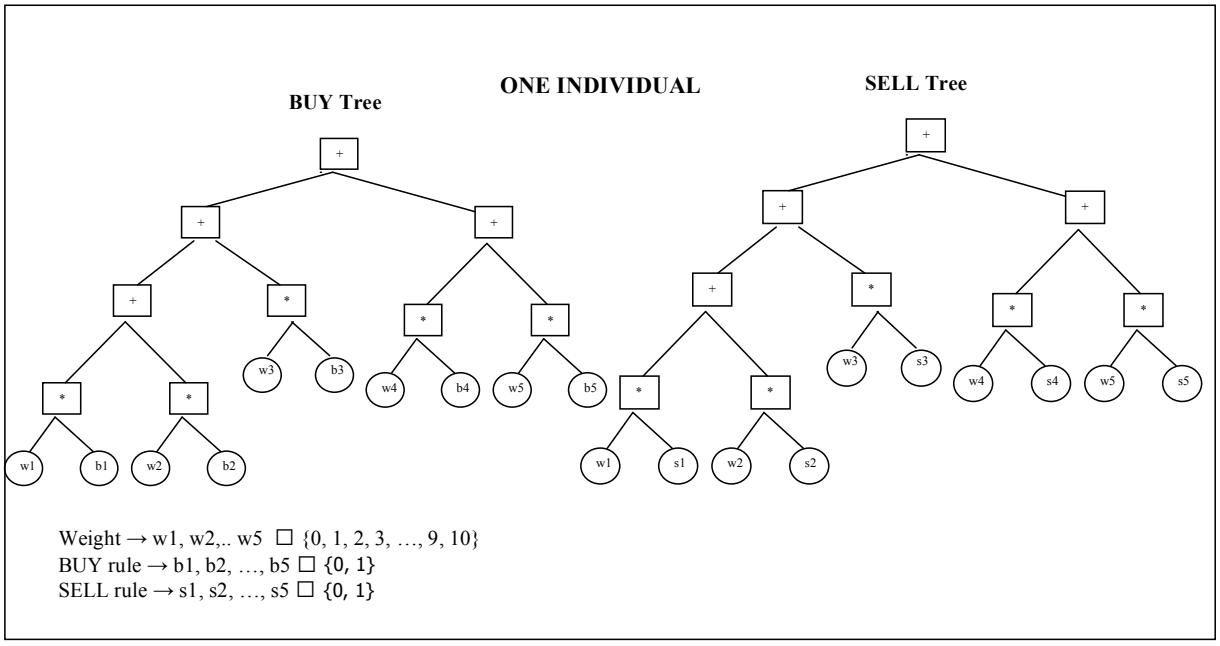


Figure 3.2: Structure of Individuals as Trees With Syntactic Restriction

3.2.3 Fitness Function

3.2.3.1 Fitness Criteria

The fitness criteria is the return of portfolio which is given by:

$$Return\ of\ Portfolio\ (R) = (V_t - V_0) / V_0,$$

where V_t is portfolio value at time t and V_0 portfolio value at the beginning of evaluation period (time zero). It is the measurement for *raw* fitness value (r). A higher numerical value signifies a fitter individual.

3.2.3.2 Fitness Calculation

In order to calculate R, we only need to calculate V_t (V_0 is given by input data). The calculation algorithm is as follows:

V_0: portfolio value at the beginning of evaluation period

V_t: portfolio value at time t (*evaluated at t-1 closing prices=t opening prices)*

V_{t-1}: portfolio value at time t-1

$v_{(i,t)}$: $ value of position in stock i at time t (*evaluated at t-1 closing prices=t opening prices)*

$v_{(min,t-1)}$: $ value of position in the stock with the lowest CS value at time t-1 (*evaluated at t-1 closing prices=t opening prices)*

$v_{(max,t)}$: $ value of position in the stock with the highest CS value at time t (*evaluated at t-1 closing prices=t opening prices)*

$Q_{(i,t)}$: number of shares of stock i in portfolio at opening at time t

$P_{(i,t-1):(closing)}$ price of the stock i at time t-1

$X_{(min,t)}$:target value of position of Stock min opening at time t

π_{min} : benchmark weighting for stock min (a constant)

$X_{(min,t)} = \pi_{min} * (1 - R_{min}) * V_t$

$X_{(max,t)}$:target value of weighting for stock max at opening time at time t

π_{max}: benchmark weighting for stock max (a constant)

$X_{(max,t)} = \pi_{max} * (1 + max) * V_t$

1) Decrease the percentage of the stock with the lowest CS value by R_{min} % by selling a quantity q_{min} of Stock $_{min}$: $q_{min} = v_{(min,t-1)} - X_{(min,t))}/P_{(min,t-1)}$
2) Purchase a quantity q_{max} of stock $_{max}$: $q_{max} = (v_{(max,t-1)} - X_{(max,t))}/P_{(max,t-1))*-1}$

if $q_{min} * P_{(min,t-1)} >= (X_{(max,t)} - v_{(max,t-1)}$ or

$q_{min} * P_{(min,t-1)}$ otherwise.

If $(q_{min} * P_{(min,t-1)} > (X_{(max,t)} - v_{(max,t-1)}$ spend the left over cash pari passu on all the stocks other than Stock $_{min}$ and Stock $_{max}$ according to bench mark weightings.

3.2.4 Operations for Modifying Structures – Crossover and Mutation

3.2.4.1 Crossover

In conventional GP, the crossover operation is rather straightforward and involves randomly choosing a node from each parent and swaping their sub trees to produce two children (see detailed explanation in the Background Chapter). However in this problem, crossover is more complex.

- Firstly, because the BUY and SELL parts of an individual need to evolve in parallel, parents chosen for crossover operation must be of the same category, which means that BUY trees only crossover with BUY trees and SELL trees with SELL trees. It is important that BUY trees do not crossover SELL trees in order to obtain a Confidence Score (CS) which faithfully represents the difference between the return value of a BUY tree (CS_{buy}) and a SELL tree (CS_{sell}), $CS = CS_{buy} - CS_{sell}$.
- Secondly, the crossover operation must be performed so as to preserve the required structure of individual for the problem. That is, the result must be an allowable trading strategy in every case. Structure-preserving crossover is performed as follows: Any point may be selected, without restriction, as the crossover point in the first parent; however, the selection of the crossover point in the second parent is then restricted to a point of the same type as the point chosen in the first parent.

3.2.4.2 Mutation

Similar to crossover, the mutation must be performed in such a way that structure of the individual is preserved. Thus, a like-for-like "swap" mutation method is used: Firstly, a random terminal node in the either BUY tree or SELL tree is chosen and then replace it with a different node of a same type. For example, if the node is chosen is *weight* 5, it can change to 1, 10, 0 or 4 etc. Similarly, if a rule node (ie, BUY_RULE_2) is chosen, it can be mutated to BUY_RULE_3, or BUY_RULE_5 etc). Weight nodes do not swap with Rule nodes.

3.2.5 Termination and Result Designation

3.2.5.1 Termination

In this system, a run of the genetic programming paradigm terminates after number of G generations. The alternative termination criterion can either involve finding a solution yielding a prespecified portfolio return or reaching a certain fitness level. The main advantage of terminating after running for G generation is practicality, because the optimal solution to the problem is unknown and computing resources are limited.

3.2.5.2 Result Designation

The "best-so-far" method of result designation is used for this problem. That is the best individual that ever appeared in any generation of the population as the result of a run of genetic programming.

4. Implementation

The GP system is written in Java and based upon version 1.20 of jgprog by R. Baruch (2001). It has been extensively modified and customised to suit many special requirements of the problem, particularly in the area of tree representation, crossover, fitness function and evolution selection.

The system implements a strongly typed GP. In almost all GP work the terminals and functions are chosen so that they can be used with each other without restriction (the requirement of "closure", details see the Background Chapter). If the set of terminals and functions were of such a kind that not all functions could bear all terminals as arguments, the system would become brittle. Typing is one of the ways out of this dilemma (Montana 1994), where each node carries its type as well as the types it can call, thus forcing functions calling it to cast the argument into the appropriate type.

This chapter highlights the most important aspects of the system including its class hierarchy, the system's underlying data structure, and key algorithms.

4.1 Class Hierarchy

Figure x illustrates class hierarchy used in the project. The main engine of the system is *Evolve* where the system enters for an evolutionary run. Evolve implements the abstract class *World* provided by the jgprog system. A GP *Population* contains one or more *Individuals*. An *Individual* have two *Chromosomes* which are represented as genetic trees. *Node* is used to build *Chromosomes*. Each *Chromosome* has a root *Node* and its childnodes which can be a function or terminal if it has not children.

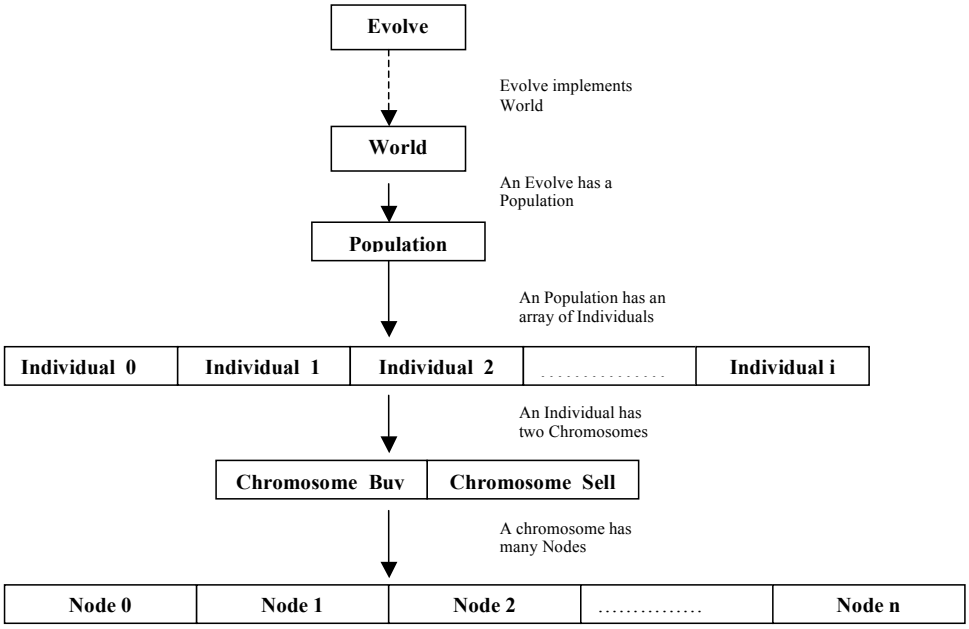


Figure 4.1: Class Hierarchy

4.2 Class Diagram

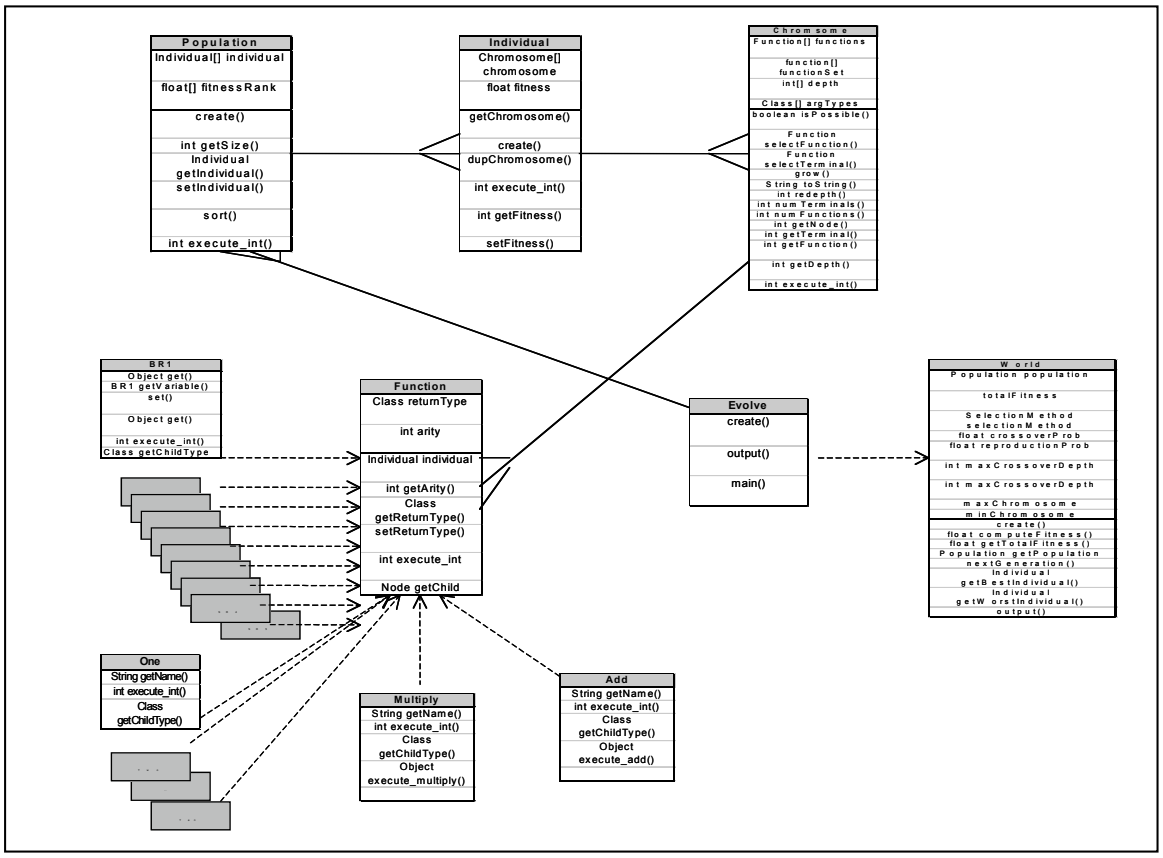


Figure 4.2: Class Diagram

- World Class is an abstract class which defines a high level process of a GP evolution run.
- Evolve Class extends World Class. It is a main engine class where a GP evolution run takes place. It contains a GP population.
- Population Class represents a population of GP individuals.
- Individual Class represents a GP individual containing one or more GP chromosomes.

- Chromosome Class represents a GP binary expression tree.
- Function Class is an abstract class representing either a GP function or a GP terminal.

a) BR1, BR2, BR3, BR4, BR5, SR1, SR2, SR3, SR4, SR5 extends Function Class.
   - BR1 represents BUY rule: current price cross above 50 days moving average.
   - BR2 represents BUY rule: current price cross above 200 days moving average.
   - BR3 represents BUY rule: 50 days moving average cross above 200 days moving average.
   - BR4 represents BUY rule: RSI <30.
   - BR5 represents BUY rule: MACD >0.
   - SR1 represents SELL rule: current price cross below 50 days moving average.
   - SR2 represents SELL rule: current price cross below 200 days moving average.
   - SR3 represents SELL rule: 50 days moving average cross below 200 days moving average.
   - SR4 represents SELL rule: RSI >70.
   - SR5 represents SELL rule: MACD <= 0.

b) One, Two, Three, Four, Five, Six, Seven, Eight, Nine, Ten extends Function Class representing weights of the rules from integer value 1 to integer value 10 inclusive.

4.3 Tree Structure

The GP system adopts the "traditional" approach (Koza (1992) and Tackett (1993) offer pointer based implementations for use in genetic programming) using binary expression tree structure to represent executable programs in the GP system.

4.3.1 Basic Data Structure

The data structure used to assemble functions and terminals into a binary expression tree is linked list. The choice of using linked list is determined by its flexibility because any information may be attached to an element. It can be particular helpful in crossover operation, because all that is required to crossover two trees is to rearrange the pointers between the nodes.

4.3.1.1 Node Representation

Nodes are the base units of a tree data structure. In this system, a node can either be a function or a terminal. In order to implement a Node of a linked list, we defined a class to encapsulate the idea of Node in a GP tree, which contains

1) a data return type that the node evaluates to. Because the system is "strongly-typed", each node must have a return type, such as integer, Boolean, float, or void.
2) arity of the node – value 0 for a terminal node and value 2 for a function node
3) a set of object references to other nodes. These object references are the edges of the tree points to either it left child node or its right child node. However, if

the arity of a node is 0 (a terminal node), the node does not contain references to other nodes.

Therefore, a basic structure of a Node in GP binary tree is defined as follows:

```
abstract class Node ()
{
        protected Class returnType;
        protected int arity;
        protected Node[] children;

        protected Node (int arity, Class type)
        {
                if (arity !=0) children = new Node[arity];
                returnType = type;
                this.arity = arity;
        }
}
```

4.3.1.2 How Each Node Is to Be Executed

Functions and terminals are executed differently. A function, either "+" or "*" in this system, takes its left and right children as its arguments and returns the result of the appropriate arithmetic operation on the two children. A terminal provides a value to the system. A terminal represents either a weight coefficient or a trading indicator. A weight terminal is defined as a numerical constant and it returns an integer value from 0 to 10 inclusive. A indicator terminal is defined as numerical variable and its return values depending on the specific properties of the rule and input data.

The execution of functions and terminals in the system requires a specific context (implemented as parameters to the execution methods) to determine where and when they are to be executed. 1) It needs to know the current chromosome where the functions and terminals executes. 2) It also needs to know the child number of the

node to execute, if it is a function. 3) Finally, it needs to know the current chromosome's argument list.

4.3.2 Tree Representation

There are a number of conventional tree building methods for the creation of the initial population of Genetic Programs in Genetic Programming literature. These include "grow method", "full method" and "ramped half-half method" which is a combination of the above (details see the Background Chapter).

However, for the purposes of this project work, it was decided to initialise each tree in the initial population as a linear combination of weighted variables (financial indicators) as illustrated in the Figure 4.3 below. Hence, the following recursive algorithm was implemented to construct such trees.

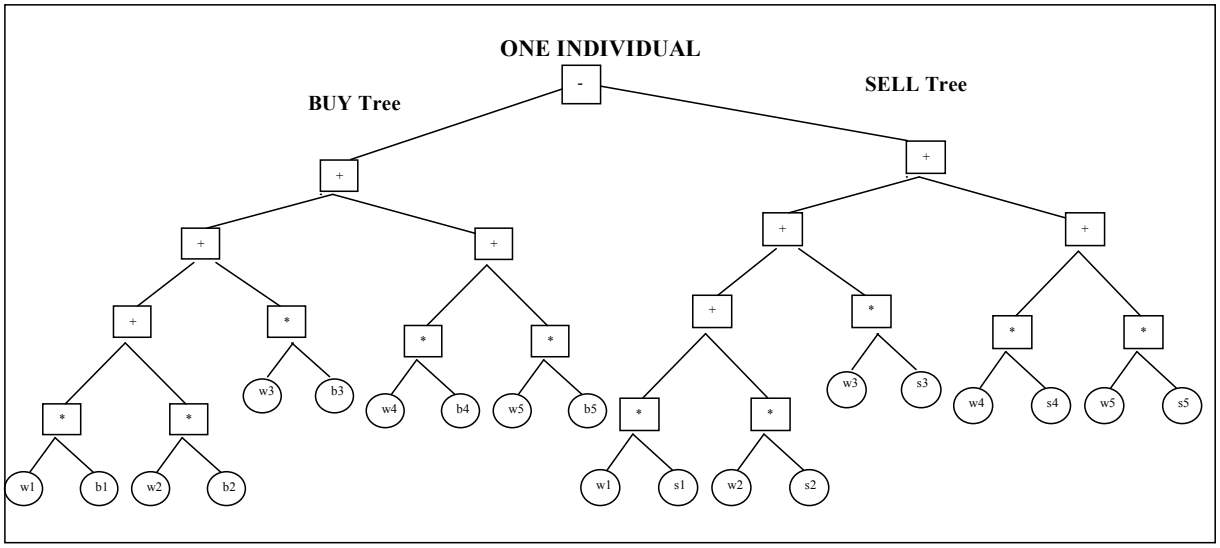


Figure 4.3: Structure of individuals in the initial population

The nodes in the tree can either be "Terminals" or "Functions". Terminals the leafs of the tree can either be B1-B5, S1-S5 according to which subtree they belong or integers 1-10 representing weights (w1-w10) as shown in the diagram 4.3. Functions on the other hand are either addition "+" or multiplication "*" with the only exception, the root of the tree which is a subtraction "-", to allow for the subtraction of the Sell subtree from the Buy subtree.

The root of both subtrees are both addition operations "+". From then on, children nodes are created according to the place we are in the tree. Two integer variables are used to denote this exact location in the tree (x,y position) with a third binary variable indicating whether we are constructing the left or right child. As can be seen from the diagram (Figure 4.3) the only nodes that are randomly selected are the leafs representing the weights which are integers from 1-10. The rest of the nodes in each tree are fixed.

4.3.2.1 Tree Evaluation

Because we use binary tree represent an expression, the levels of the nodes in the tree indicate their relative precedence of evaluation. Operations at higher levels of the tree are evaluated later than those below them. The operation at the root is always the last operation performed. The system uses postfix order for the tree evaluation (Figure 4.4). It proceeds by repeatedly evaluating the leftmost node for which all inputs are available.

It has number of advantages: 1) Parentheses are not needed. 2) Precedence is implicit. 3) The algorithm is simple and efficient. Therefore, it is easy for the system to process the algorithm.

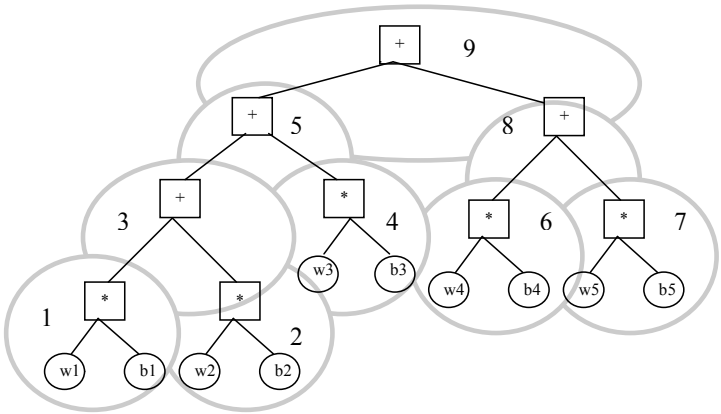
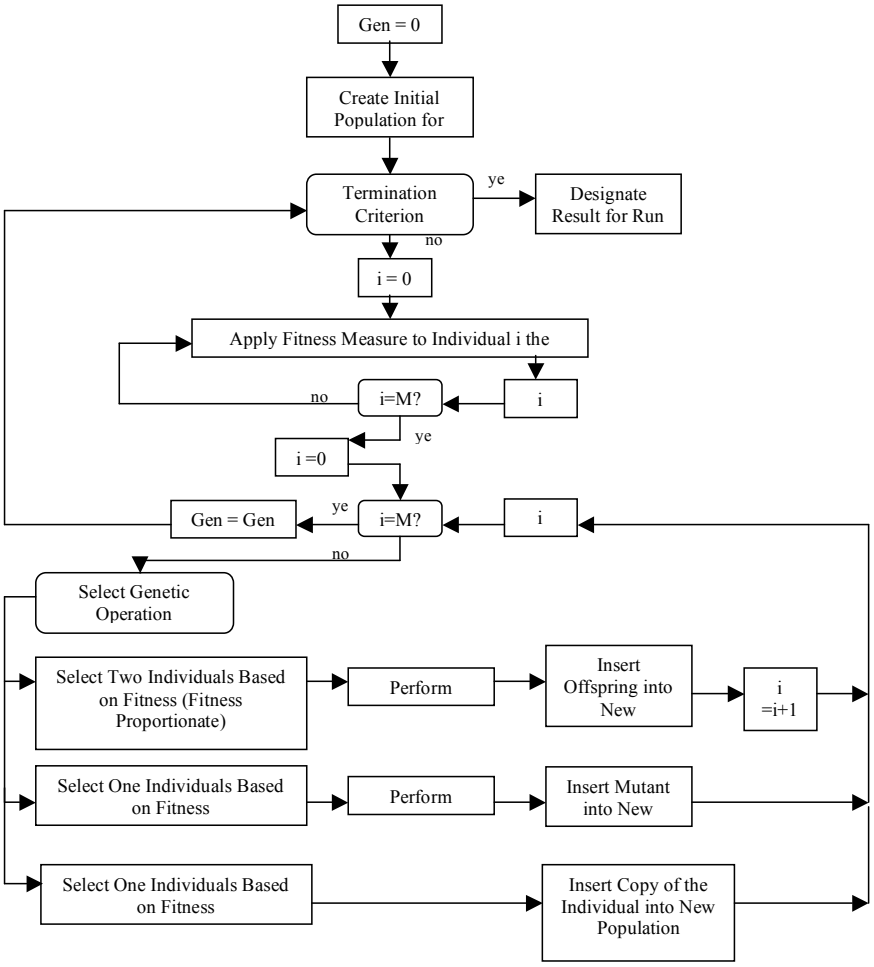


Figure 4.4 Postfix Order

4.3.3 Main Engine Class *Evolve*

Evolve, a subclass of abstract class *World*, is the main engine class where a genetic evolution run takes place. The flowchart 4.5 illustrates how *Evolve* implements the evolution process and the execution steps of a run.



Flowchart 4.5 An evolution run

4.3.4 Crossover Operation

The crossover algorithm used here takes a different approach from the conventional one-point crossover method (Koza 1992). The main objective of the algorithm is to impose constraints on how the selection of crossover points will be performed in order to preserve the required syntactic structure.

The algorithm is as follows: Initially, the algorithm randomly chooses a crossover point in the first chromosome. Then, it proceeds to choose another crossover point in the second chromosome with restrictions: the second crossover node must be of the same type as the first node. In other words, a "+" crossover with a "+" only ; a "*" crossover with a "*" only ; a rule terminal crossover with a rule terminal only and a weighting terminal crossover with a weighting terminal only. If a suitable point in the second chromosome could not be found then the chromosomes are not crossed. Although the algorithm imposes rules on crossover, it still allows the GP to crossover at any point in the tree and produce offsprings of varying length and, most importantly, it allows genetic material to move from one place to another in the genome. Therefore, the main advantage of GP crossover is not lost.

4.3.5 Ranking Selection

The selection algorithm implemented in the system is Ranking Selection (Grefenstette and Baker, 1989). The algorithm is based on the fitness order, into which the individuals can be sorted. The selection probability is assigned to individuals as a function of their rank in the population.

$$pi = 1/N * \{p_{worst} + (p_{best} - p_{worst}) * [(I-1)/(N-1)]\}$$

where p_{worst}/N is the probability of the worst individual being selected, and p_{best}/N the probability of the best individual being selected.

4.3.6 Control Parameters

The genetic programming paradigm is controlled by 13 control parameters, including two major numerical parameters, 9 minor numerical parameters (Koza 1992).

The two major numerical parameters are the population size (M) and the maximum number of generations to be run (G).

- The population size M is 500.
- The maximum number G of generations is 51 (an initial random generation, called generation 0, plus 50 subsequent generations).

The eleven minor numerical parameters used to control the process are described below:

- The probability of crossover, pc, is 0.90. That is, crossover is performed on 90% of the population for each generation. For example, if the population size is 500, then 450 individuals (225 pairs) from each generation are selected (with reselection allowed) to participate in crossover.
- The probability of reproduction, pr, is 0.10. For example, if the population size is 500, 50 individuals from each generation are selected for reproduction (with reselection allowed).
- In selection crossover points, a probability distribution is used. It allocates pp = 90% of the crossover points equally among the function points of each tree and 10% of the crossover points equally among the terminal points of each tree. This distribution promotes the recombination of larger structures.
- A maximum size (measured by depth), $D_{created}$, is established as 10 for trees created by the crossover operations.
- A maximum size (measured by depth), $D_{initial}$, is established as 4 for the random individuals generated for the initial population.
- The probability of mutation, pm, specifying the frequency of performing mutation is 0.

5. Testing

The testing takes the form of verification, checking the system has the characteristics that it should have; and validation, checking that it does the job it is supposed to do.

5.1 Verification

Apart from verifying all the classes compile cleanly and run without errors, verification also needs to test whether the system meet specific design requirements described in the Analysis and Design chapter and to test the system robustness and correctness.

5.1.1 Test the System Against Specifications

Important Specifications in Design	System Components Verified
• One GP individual has two chromosomes.	• As an individual in the system holds an array of chromosomes, the array length determines how many chromosomes the individual should hold.
• The structure of chromosomes in the initial population need to conform the structure described in x page.	• It is necessary to write a test program in the Chromosome class in this case. The program 1) constructs an empty Chromosome object; 2) calls the tree creation method to build the chromosome and 3) prints out the chromosome.

• The required tree structure is not destroyed after the crossover operation.	• Again, a testing program in the Chromosome class is required to perform the test. In fact, it is an extension of the chromosome structure test. There are two new elements in the program: 1) the method call of crossover method and 2) printing out the two offsprings of the crossover operation.

5.1.2 Robustness Test

The main objective of the robustness test is to verify whether the system is able to cope with marginal user input values. The following values are defined by users:

- Crossover Probability (P_c): $0.0 \leq P_c \leq 1.0$
- Reproduction Probability (P_r): $0.0 \leq P_r \leq 1.0$
- Maximum crossover depth (D_{max}): $D_{max} > 0$
- Maximum initial depth (D_{init}): $D_{init} > 0$
- Population size (P): $P > 0$
- Chromosome size of an Individual (C): $C>0$

The system cannot function correctly or at all if above values supplied by users are not within the valid range. In the robustness test, values (-1, 1.5) are used for P_c, P_r, values (-10, 300) for D_{max} and D_{init}, values (0, -20, 5000) for P and values (0, 3, -2) for C to explore the vulnerability of the system.

5.1.3 System Correctness

The fitness convergence test is carried out to gain insight of GP evolution. The fitness convergence test is widely used by Koza (1992) in his book.

In the test run, the initial population size is 500, the crossover rate 0.9, the reproduction rate 0.1 and the number of iterations 51. The data set is the daily closing prices of the selected 32 stocks during the period of 17, April, 2003 – 27, June, 2003 (training period). The run takes approximately 1 hour on a Laptop with specification of Pentium 4 2Ghz, 512MB RAM, 60GB HD running Windows XP Professional.

Figure 5.1 contains the *standardised fitness*[5] curves for this run. This figure shows, by generation, the progress of one run of the problem between generation 1 and 51, using three plots: the standardised fitness of the best-of-generation individual in the population, the standardised fitness of the worst-of-generation individual in the population, and the average value of standardised fitness for all the individuals in the population. As can be seen, the standardised fitness of the best-of-generation started at 24.95 in generation 1 and improved (i.e., decreased) to 20.65 in generation 51. The improvement in fitness from generation was steady, but not perfectly monotonic; there was no great leap in performance. The average standardised fitness of the population also improved between generation 1 and 51. Again, there was no great leap in performance. The standardised fitness of worst-of-generation has improved more significantly than that of average and best-of-generation. It started at 39.49 in generation 1 and improved to 25.16 in generation 51.

[5] The *raw fitness (Fr)* is the return of the portfolio. The *standardised fitness (Fs)* restates the raw fitness so that a lower numerical value is always a better value. Fs = 1/Fr

The above observation clearly shows that three fitness curves converge towards higher fitness value during the evolution process and the GP system evolves correctly.

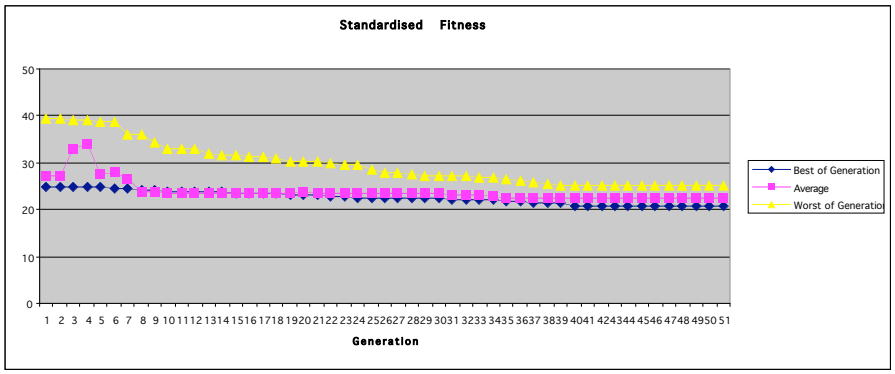


Figure 5.1: Standardised Fitness Curve

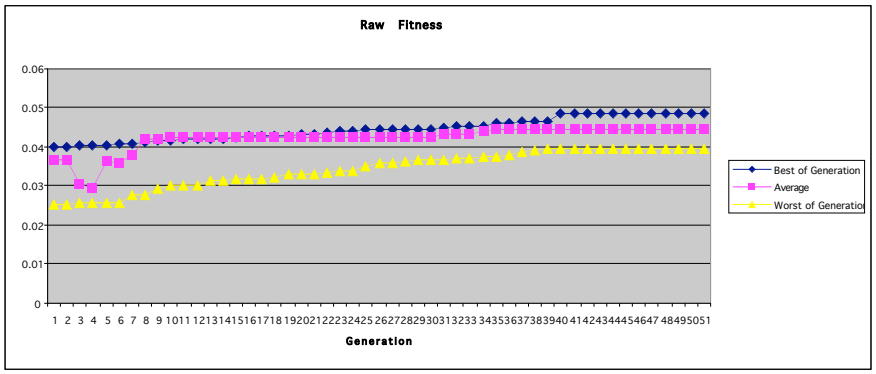


Figure 5.2: Raw Fitness Curve

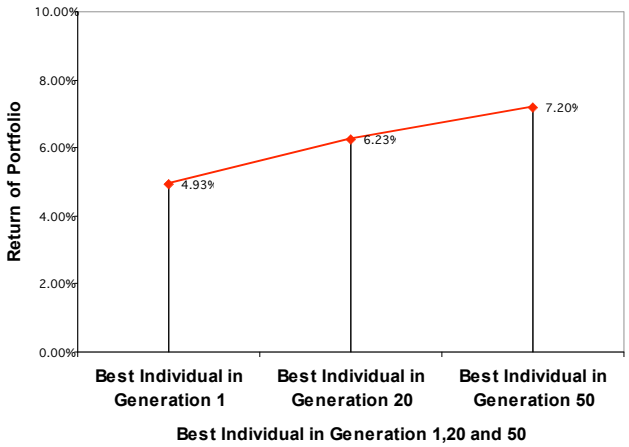


Figure 5.3: Portfolio Return and Best Individuals

Figure 5.3 depicts the relationship between portfolio return ratio and best individuals at three different stages of evolution. At the very beginning of the evolution process, the best individual in generation 1 had a return of 4.93%. When the system reached generation 20, return improved to 6.23%. The system continued to evolve, at generation 50 the return further improved to 7.2%. This confirms the result of fitness convergence test.

5.2 Validation

5.2.0 Definitions

To make the description and commentary of results easier to follow we will adopt the following definitions:

- The return achieved by the portfolio driven by the system will be called the system return.
- The return achieved by the baseline portfolio which mirrors passively the S&P
- Benchmark will be called the benchmark return.

The return here will be expressed in conventional percentage terms without reference to the transformed formula used as the measure of fitness in the code.

5.2.1 In-Sample Results

In the validation test run, the initial population size used is 500, the crossover rate 0.9, the reproduction rate 0.1 and the number of iterations 51. The training data used are the daily closing prices of a basket of 32 stocks during the period of 17, April, 2003 – 27, June, 2003.

Objective	Find a weighted linear combination of technical indicators that enhances portfolio return
Terminal Set	5 BUY indicators and 5 SELL indicators, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
Function Set	+ , *
Raw Fitness	Return on portfolio
Population	500
Termination	Terminates after generation G=51

In in-sample the system return is 7.2%. The return outperforms the benchmark return of 6.67% during the same period.

Due to time constraints no statistical test has been performed to evaluate whether this excess return is statistically significant.

However a qualitative examination of the portfolio adjustments over time reveals that the system seems well behaved:

As can be seen from Figure 5.4, the trading strategy produced by GP was able to make appropriate investment decisions when significant price movements emerged. For instance, as prices of Amgen and AOL shares increased 30% and 21% respectively, according to the signals generated by the trading strategy, the system invested in the two stocks quite significantly. While the system sold large quantity of the shares held in Johnson & Johnson during the period its price declined more than 15%.

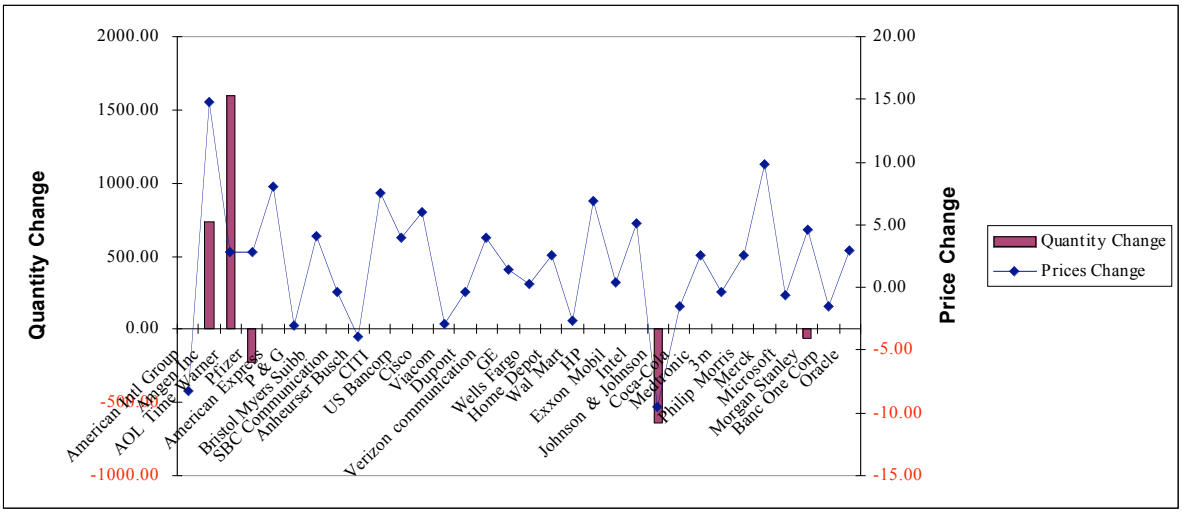


Figure 5.4: Price variation vs quantity variation in individual stock during the training period

5.2.2 Out-of-Sample Results

The best-so-far trading rule (see above) is subsequently tested out-of-sample from 28, June, 2003 to 05, August, 2003. The system has been run with two values for the R_{min} R_{max} parameters respectively 10% and 20%.

In the 10% run system return during this period is 4.59% versus 4.79% for the benchmark.

The excess return of the system disappears out of sample and this raises the usual concern that the good results over the training period are merely due to overfitting. However the qualitative look at portfolio adjustments continues to indicate that the system is well behaved.

Figure 5.5 shows the stock price variation and quantity bought or sold during the out-of-sample period. As can be seen, as in the training period, the trading strategy produced by GP was able to make appropriate investment decisions when significant price movement trend emerged. In fact, the rule performed extremely well: under its guidance, the portfolio invested in all the stocks with a favourable price movement more than 10%, and sold all the stocks with a negative price movement except for Pfizer stock.

In the run with the value of the over and under weighting parameters set at 20%
the system return is 5.622% versus the benchmark return of 4.79%.(see Figure 5.6)

Even without statistical significance tests such a margin of excess performance is remarkable. (Barely matching the benchmark after fees is relative rare for most money managers and achieving excess returns of this amplitude is considered exceptional see Vanguard Group July 2003 Newsletter).

This suggests that the disappearance of the excess return in the 10% run reflects that the threshold of 10% underweighting and overweighting of the 2 two stocks with extreme scores is set too low rather than the fact that the information conveyed in the scores is too weak.

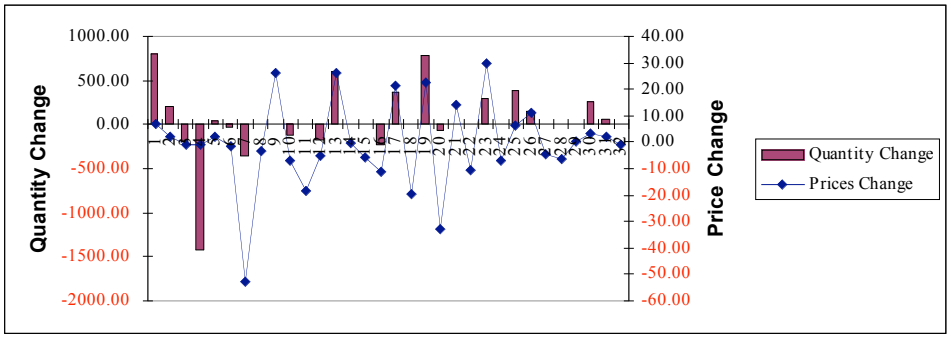
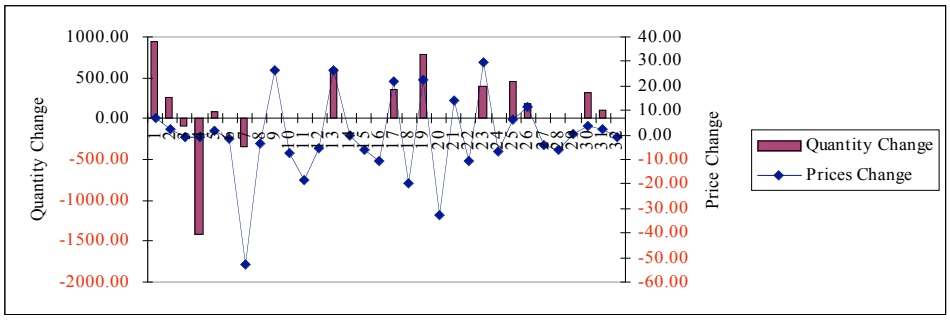


Figure 5.5: Price variation vs quantity variation in individual stock during the out-of-sample period



6. Conclusion and Further Work

This chapter will draw conclusion from the results of the validation test as well as providing suggestion as to how the work could be improved upon in the future.

Figure 5.6: Price variation vs quantity variation in individual stock during the out-of-sample period

pilot study incorporates enough of the important features of the real life conditions to provide indications on the merit of further work. And the out of sample results are indeed encouraging.

A reminder of the goal that is described in the introduction is now given:

- Deliver a pilot system illustrating the use of technical factors to make "profitable, return enhancing" portfolio adjustments.

In spite of the many limitations both in the design of the pilot system and in the scope of the data universe and the range of parameters experimented those results are enough to claim that the pilot system works correctly and technical indicators are robust enough to provide a source of enhancements for portfolio returns well worth exploring further.

This pilot is also enough to suggest the type of analytical framework that can be used to extend to multiasset portfolios the type of results obtained by successful GA financial applications for single asset trading.

The natural next progression for this work is to consider:

- **Further Validation**

Consider a broader data universe, other benchmarks and much longer time periods. This should provide the opportunity for statistical testing of the significance of the results achieved by the system as well as well as to gain insights for design improvements from the qualitative look at a much broader range of results

- **Design Improvements**

1. In terms of the portfolio adjustment one could consider many more features such as:

   -over and under weighting for more than the highest and lowest ranking stock

   -variable percentages in over and under weighting

   -additional constraints imposed on the portfolio, especially in terms of tracking error

2. In terms of the scoring system one could consider other individual technical indicators and metarules formed otherwise than by simple linear combination of individual indicators.

3. In terms of the search methodology one should evaluate the GP against other methodologies including the old multiregression technique that would also produce an estimation of the weights for each indicator.

- Last, and maybe most importantly, this pilot project suggest that the field of portfolio return enhancements may lend itself to adaptive GP systems. If the pilot suggests encouraging results incorporating such developments would be mainly a matter of additional resources rather than of the drastic modification of the analytical framework.

References

Allen, F & Karjalainen, R. (1999). "Using Genetic Algorithm to Find Technical Trading Rules", *Journal of Financial Economics 51: 245 – 271.*

Banzhaf, W *et al* (1998), *Genetic Programming: An Introduction On the Automatic Evolution of Computer Programs and Its Applications*, MK Publishers

Bauer, R. (1994), Genetic Algorithms and Management of Exchange Rate Risk, *presentation slides*

Becker, L.A. & Seshadri, M. (2003), "Comprehensibility & Overfitting Avoidance in Genetic Programming for Technical Trading Rules", *Worcester Polytechnic Institute, Computer Science Technical Report*

Clack C (1996), "Genetic Algorithms and Genetic Programming"*, presented at 1996 Emerging Technologies Workshop*, UCL, July **1996** (15 pages)

Clack C & Yu T (1997), "Performance-Enhanced Genetic Programming", *Proc. Evolutionary Programming 1997 Conference (EP'97)*, P.J. Angeline, R.G. Reynolds, J.R. McDonnell, and R. Eberhart (Eds), Springer Verlag, LNCS 1213, ISBN 3-540-62788-X, **1997**.

Chen, S (1999), "Evolutionary Computation in Finanical Engineering: A Roadmap of GAs and GP", *Financial Engineering News Vol 2 Number 4*

Cormen, T.H. *et al* (1999), *Introduction to Algorithms*, The MIT Press

Dampster MA & Jones CM (2000), "A Real-Time Adaptive Trading System Using Genetic Programming", *Quantitative Finance Vol 1 (2001) 397 – 417*, University of Cambridge

Flanagan, D. (2002), *Java In A Nutshell*, O'Reilly

Ghanea-Hercock, Robert (2003), *Applied Evolutionary Algorithms in Java*, Springer

Holland J H (1992), *Adaptation in Natural and Artificial Systems*, MIT Press

Karjalainen, R.E. (1994), "Using Genetic Algorithms to Find Technical Trading Rules in Financial Markets", *Ph.D. Thesis, University of Penneylvania*.

Koza, (1992), *Genetic Programming: One the Programming of Computer by Means of Natural Selection*, The MIT Press

Langdon W & Qureshi A (1995) "Genetic Programming – Computers Using 'Natural Selection' to Generate Programs", *Working Paper*, University of College London

Langdon, W. and Poli, R (2001), *Foundations of Genetic Programming*, Springer

Li J & Tsang E.P.K (1999) “Improving Technical Analysis Predictions: A Application of Genetic Programming”, *Proceedings, Florida Artificial Intelligence Research Symposium*, USA

Malkiel S.W. (1995) *A Random Walk down Walk Street*, W.W.Norton, New York

Markowitz H.M (1952), “Portfolio Selection”, *Journal of Finance 7*, March 1952

Neely, C., Weller, P., & Dittmar, R. (1997). “Is Technical Analysis in the Foreign Exchange Market Profitable? A Genetic Programming Approach”, *Journal of Financial and Quantitative Analysis 32: 405-26*

Neely, C., Weller P. (1999) “ Technical Trading Rules in the European Monetary System”, *Journal of International Money and Finance 32*

Kaufman.P (1995) *Kaufman on Market Analysis – Volume*, KDY Seminar in Print

Rode D. et al (1995), “An Evolutionary Approach to Technical Trading and Captial Market Efficiency”, *Working Paper*, The Wharton School, University of Pennsylvania

Viner J (1998), “The Application of Intelligent Systems to Finance and Business”, *PhD Thesis*, University College of London

Wittmar R (2000), “Can Technical Analysis Still Beat Random System?”, *Speech on the IFTA Conference 2000*

Vanguard (2003) “Source of Portfolio Performance: The Enduring Importance of Asset Allocation”, *Research Report Published by Vanguard Investment Conselling & Research* July 2003

Oussaidene, M, Chopard, B., Pictet, O & Tomassini M. (1997), “Practical Aspects and experiences – Parallel Genetic Programming and its Application to Trading Model Induction”, *Journal of Parallel Computing* Vol 23, No 9