Chapter 1

DISCOVERING FINANCIAL TECHNICAL TRADING RULES USING GENETIC PROGRAMMING WITH LAMBDA ABSTRACTION

Tina Yu
ChevronTexaco Information Technology Company
San Ramon, CA 94583 USA
gwoing_yu@yahoo.com

Shu-Heng Chen
AI-ECON Research Center, Department of Economics
National Chengchi University, Taipei, Taiwan, 11623
chchen@nccu.edu.tw

Tzu-Wen Kuo
AI-ECON Research Center, Department of Economics
National Chengchi University, Taipei, Taiwan, 11623
kuo@aiecon.org

Abstract We applied Genetic Programming with lambda abstraction module mechanism to learn technical trading rules based on S&P 500 index from 1982 to 2003. The results show strong evidence of excess returns over buy-and-hold after transaction cost. The rules can be interpreted easily; each uses a combination of one to four widely used technical indicators to make trading decisions. The consensus among Genetic Programming rules is high, with 80% of the evolved rules give the same decision for most of the testing period. The Genetic Programming rules give high transaction frequency. Regardless of market climate, they are able to identify opportunities to make profitable trades and out-perform buy-and-hold.

Keywords: modular genetic programming, lambda abstraction, higher-order-functions, financial technical trading rules, S&P 500 index, automatically defined functions, PolyGP system, stock market, constrained syntactic structure, strongly typed genetic programming.

Introduction

In this chapter Genetic Programming (GP) combined with a Lambda Abstraction module mechanism is used to find technical trading rules that make profitable trading in stock markets. The returns from GP trading rules are compared with the returns from buy-and-hold strategy. In this strategy, stocks purchased at the beginning of the term are kept until the end of the term when they are closed; no trading activity takes place during the term. It is the mostly frequently used benchmark to evaluate the profitability of any trading rules.

Finding profitable trading rules is not equivalent to the problem of forecasting stock prices, although the two are clearly linked. A profitable trading rule may forecast rather poorly most of the time, but perform well overall because it is able to position the trader on the right side of the market during large price changes. One empirical approach to predict the change of price trends is technical analysis. This approach uses historical stock prices and volume data to identify the price trends in the market. This technique was originated from the work of Charles Dow in the late 1800s, and is now widely used by investment professionals as input for trading decisions [13].

Various trading indicators have been developed based on technical analysis techniques. Examples are *moving average*, *filter* and *trading-range break*. Moving average includes a class of indicators where the trading signals are decided by comparing a short-run with a long-run moving average in the same time series, producing a "buy" signal when the short-run moving average is greater than the long-run moving average. This rule can be implemented in many different ways by specifying different short and long periods. Filter rules include a class of trading indicators where the trading signals are decided by comparing the current price with its local low or with its local high over a past period of time. It can also be implemented in different time length. Trading-range break combines multiple filter indicators to make trading decisions. Figure 1.1 gives a moving average indicator example and a trading-range break indicator example, which is a combination of 2 filter rules.

[3] reported that *moving average* and *trading-range break* give significant positive returns on Dow Jones Index from 1897 to 1986. [5] showed that *filter* strategy can out-perform buy-and-hold under relatively low

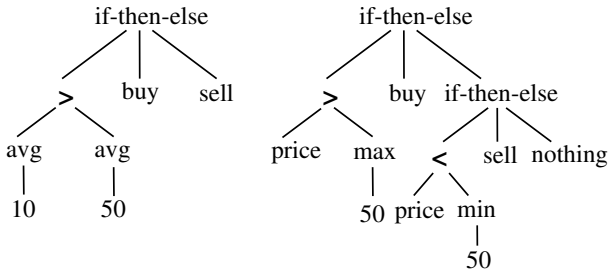

Figure 1.1. Moving average (10,50) and trading-range break rules.

transaction cost on NYSE and AMEX stocks for the 1962-1993 period. These studies are encouraging evidences indicating that it is possible to devise profitable trading rules for stock markets.

However, one concern toward these studies is that the investigated trading indicators are decided *ex post*. It is possible that the selected indicator is favored by the tested time periods. If the investor has to make a choice about what indicator or combination of indicators to use at the beginning of the sample period, the reported returns may have not occurred. In order to obtain true out-of-sample performance, Genetic Programming (GP)[7] has been used to devise the trading rules for analysis. For the two attempts made, both of them reported that GP can not find trading rules that out-perform buy-and-hold on S&P 500 index [1] [14] (see Section 1 for details). One possible reason of this outcome is that the GP systems used are not adequate for this task. This work extends GP with λ abstraction module mechanism and investigate its ability to find profitable technical trading rules based on S&P 500 index from 1982 to 2002.

This chapter is organized as follows. Section 1 reviews related work. Section 2 presents the λ abstraction module mechanism. In Section 3, the PolyGP system is described. In section 4, S&P 500 time series data are given. Section 5 explains the experimental setup while Section 6 presents the experimental results. We analyze the GP trading rules in Section 7 and 8. Finally, concluding remarks are given in Section 9.

1. Related Work

Targeted toward different financial markets, different researchers have applied GP to generate trading rules and to analyze their profitability. For example, [1] studied S&P 500 index from 1928 to 1995. They reported that the evolved GP trading rules do not earn consistent excess returns over buy-and-hold after transaction costs. In contrast, [9] re-

ported that their GP trading rules for foreign exchange markets were able to gain excess returns for six exchange rates over the period 1981-1995. [14] suggested that this conflicting result might be due to the fact that foreign exchange markets have a lower transaction cost than the stock markets have. Another reason Wang suggested is that [9] did not use the rolling forward method (explained in Section 4) to test their results for different time periods while [1] did. Finally, Wang pointed out that these two works used different benchmarks to assess their GP trading rules: [1] used the return from buy-and-hold while [9] used zero return, because there is no well-defined buy-and-hold strategy in the foreign exchange markets.

Using a similar GP setup as that of [1], [14] also investigated GP rules to trade in S&P 500 futures markets alone and to trade in both S&P 500 spot and futures markets simultaneously. He reported that GP trading rules are not able to beat buy-and-hold in both cases. Additionally, he also incorporated Automatically Defined Functions (ADFs) [8] in his GP experiments. He reported that ADFs made the representation of the trading rules simpler by avoiding duplication of the same branches. In this work, Wang did not compare the returns from GP rules with the returns from ADF-GP rules.

A different approach using GP to generate trading rules is combining predefined trading indicators [2] [10]. In these works, instead of providing functions such as *average* for GP to construct moving average indicator and *minimum* to construct filter indicator, some of the trading indicators are selected and calculated. These indicators are then used to construct the leaves of GP trees. Since there are a wide range of trading indicators, this approach has an inevitable bias; only selected indicators can be used to construct trading rules. Modular GP relieves such bias by allowing any forms of indicators to be generated as modules and then combined to make trading decisions.

In a previous work [17], we used GP to generate trading strategies that consists of rules to trade in stock markets and foreign exchange markets simultaneously. Moreover, we also included ADFs in our GP implementations. However, the results show that most ADFs are evaluated into constant value of *True* or *False*. In other words, ADFs do not fulfill the role of identifying modules in the trading rules. As a result, ADF-GP trading rules give similar returns as that of vanilla GP trading rules; both of them are not as good as the returns from buy-and-hold. This suggests that either there is no pattern in financial market trading rules or ADF is not able to find them. We find this outcome counter-intuitive since it is not uncommon for traders to combine different indicators to make trading decisions. We therefore decide to investigate a different

GP modular approach (λ abstraction) to better understand GP ability in finding profitable trading rules.

2. Modular GP through Lambda Abstraction

Lambda abstractions are expressions defined in λ calculus [4] that represent function definition (see Section 3 for the syntax). In a GP program tree, they are treated as independent modules which have unique identity and purpose. λ abstraction can take inputs and produce outputs. They are protected as units throughout the program evolution process.

One way to incorporate λ abstraction modules in GP is using higher-order functions. Higher-order functions are functions which take other functions as inputs or return functions as outputs. When a higher-order function is used to construct GP program trees, its function arguments are created as λ abstractions modules. These modules evolve in ways that are similar to the rest of the GP trees. However, they can only interact with their own kind to preserve module identities.

For example, Figure 1.2 gives two program trees, each contains two different kinds of λ abstraction modules: one is represented as a triangle and the other a cycle. Cross-over operations are only permitted between modules of the same kind.

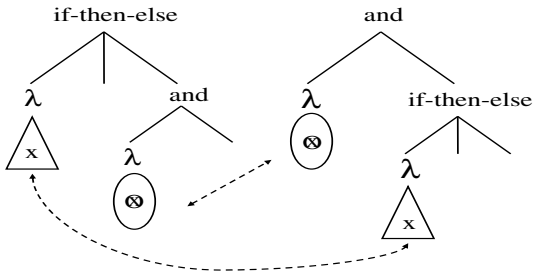

Figure 1.2. Cross-over between λ abstraction modules in two GP trees.

We use *type information* to distinguish different kind of λ abstraction modules. λ abstractions that have the same number of inputs and outputs with the same argument and return types are of the same category.

λ abstractions modules are simultaneously evolved with the main program. With protected structures, they can perform a top-down process to decompose a problem. In other words, they provide a "divide and conquer" mechanism for GP to perform problem solving. Modules can also be reused. In a previous work, λ abstractions are reused through implicit recursion to solve the even parity Boolean function problem [16]. However, in this chapter, we only investigate λ abstraction in its ability

to discover and exploit inherent patterns in technical trading rules. The reuse of λ abstraction modules in finding technical trading rules will be studied in our future work.

3. The PolyGP System

PolyGP [15] is a GP system that evolves expression-based programs (λ calculus). The programs have the following syntax:

$exp :: c$	constant
$\mid x$	identifier
$\mid f$	built-in function
$\mid exp1\ exp2$	application of one expression to another
$\mid \lambda x.exp$	lambda abstraction

Constants and identifiers are given in the terminal set while built-in functions are provided in the function set. Application of expressions and λ abstractions are constructed by the system.

Each expression also has an associated type. The types of constants and identifiers are specified with known types or type variables. For example, the stock price index has a type Double.

$index :: Double$

The argument and return types of each built-in function are also specified. For example, the function "+" takes two Double type inputs, and return a Double type output.

$+ :: Double \rightarrow Double \rightarrow Double$

For higher-order functions, their function arguments are specified using brackets. For example, the first argument of function IF-THEN-ELSE can be specified as a function that takes two argument (one of Time type and one with Double Type) and returns a Boolean value.

$IF-THEN-ELSE :: (Time \rightarrow Double \rightarrow Boolean) \rightarrow Boolean \rightarrow Boolean \rightarrow Boolean$

Using the provided type information, a type system selects type-matching functions and terminals to construct type-correct expression trees. An expression tree is grown from the top node downwards. There is a required type for the top node of the tree. The type system selects a function whose return type matches the required type. The selected

function will require arguments to be created at the next (lower) level in the tree: there will be type requirements for each of those arguments. If the argument has a function type, a λ abstraction will be created to represent it. Otherwise, the type system will randomly select a function (or a terminal) whose return type matches the new required type to construct the argument node.

λ abstractions are created using the same function set as that used to create the main program. The terminal set, however, consists both the terminal set used to create the main program and the arguments of the λ abstraction. Argument naming in λ abstractions follows a simple rule: each argument is uniquely named with a hash symbol followed by an unique integer, e.g. #1, #2. This consistent naming style allows cross-over to be easily performed between λ abstractions with the same number and type of arguments.

4. S&P 500 Index Time Series Data

We have acquired S&P 500 index time series data between January 1, 1982 and December 31, 2002 from *Datastream*. Since the original time series are non-stationary, we transform them by dividing the daily data by a 250-day moving average. This is the method used by [1], [9]. The adjusted data oscillate around 1 and make the modeling task earlier.

There is a different approach to normalize financial time series by converting the price index series into a return series. In this approach, the price difference between two consecutive days (first-order difference) are calculated to become the return series. Since our work focuses on relatively difference to [1] and [9], their method for normalization is adopted. We are also aware of the controversy about whether to use price series or return series for financial modeling [6].

Figure 1.3 gives the original and the transformed time series. There are 3 distinct phases in this time series. From 1982 to 1995, the market inclines consistently; between 1996 and 1999, the market bulls. After 2000, the market declines. With such a diversity, this data set is suitable for GP to model trading rules.

While the transformed series are used for modeling, the computation of GP trading rules returns is based on the original time series. One implication of this data transformation is that GP is searching for rules exhibited in the *change of price trend* that give profitable trading rules.

Over-fitting is an issue faced by all data modeling techniques. GP is no exception. When optimizing the trading rules, GP tends to make the rules producing maximum returns for the training period, which may contain noise that do not represent the overall series pattern. In order

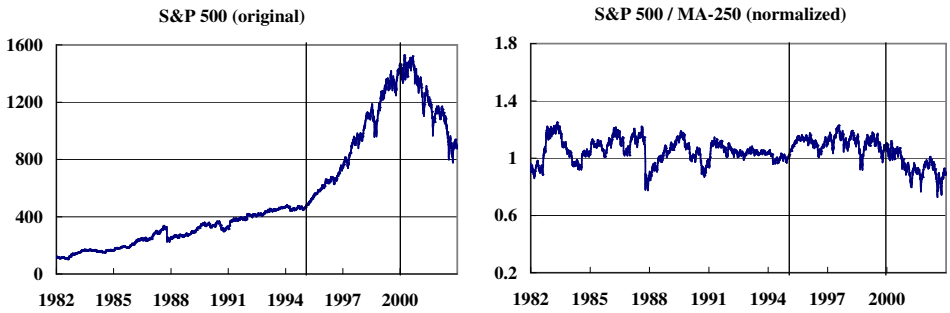

Figure 1.3. Time series data before and after normalization.

to construct trading rules that generalize beyond the training data, we split the series into training, validation and testing periods. We also adopted the rolling forward method, which was proposed by [12] and used by [1] and [14].

To start, we reserve 1982 data to be referred by time series functions such as *lag*. The remaining time series are then organized into 7 sequences, each of which will be used to make independent GP runs. In each sequence, training period is 4-year long, validation period is 2-year and testing period is 2-year. The data in one sequence may overlap with the data in other sequences. As shown in Figure 1.4, the second half of the training period and the entire validation period at the first sequence are the training period at the second sequence. The testing period at the first sequence is the validation period at the second sequence. With this setup, each testing period is two-year and covers a different time period from 1989 to 2002.

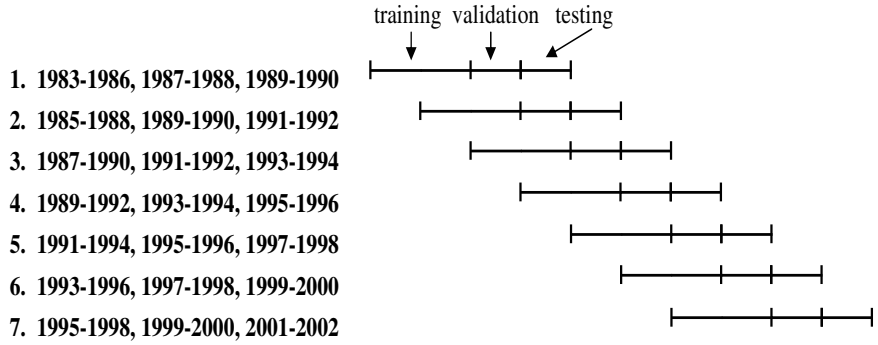

Figure 1.4. Training, validation and testing periods for 7 time sequences.

Data from the training period are used to construct/optimize GP trading rules while data from the validation period are used to select the GP trading rules, which are then applied on the testing period data to give the performance of the rule. The evaluation of the GP rules performance is based on results from the testing periods.

5. Experimental Setup

We made two sets of runs: one with λ abstraction modules and one without. The three higher-order functions defined for GP to evolve λ abstraction modules are:

$$AND :: (Time \rightarrow Boolean) \rightarrow Boolean \rightarrow Boolean$$
$$NOR :: (Time \rightarrow Boolean) \rightarrow Boolean \rightarrow Boolean$$
$$IF - THEN - ELSE :: (Time \rightarrow Double \rightarrow Boolean) \rightarrow Boolean$$
$$\rightarrow Boolean \rightarrow Boolean$$

The first argument of AND and NOR is a function with a Time type argument. As described before, this argument will be created as λ abstraction in the GP trees. Since the two λ abstractions are of the same category, the left branch of an AND node in a GP tree is allowed to cross-over with the left branch of either an AND or a NOR node in another GP tree. The first argument of IF-THEN-ELSE, however, has unique types. It's left branch is therefore only allowed to cross-over with the left branch of IF-THEN-ELSE node in another GP tree. We constrains a GP tree to have a maximum number of 4 higher-order-functions to preserve computer memory usage.

Table 1.1 and 1.2 give the functions and terminals that are used by both sets of GP runs. The function *avg* computes the moving average in a time window specified by the integer argument. For example, $avg(250)$ at time t is the arithmetic mean of $index_{t-1}, index_{t-2}, \cdots, index_{t-250}$. The function *max* returns the largest index during a time window specified by the integer argument. For example, $max(3)$ at time t is equivalent to $\max(index_{t-1}, index_{t-2}, index_{t-3})$. Similarly, the function *min* returns the smallest index value during a time window specified by the integer argument. The function *lag* returns the index value lagged by a number of days specified by the integer argument. For example, $lag(3)$ at time t is $index_{t-3}$. These functions are commonly used by financial traders to design trading indicators, hence are reasonable building blocks for GP to evolve trading rules.

Table 1.1. Functions and their types used for both sets of GP runs.

Name	Type
OR	$Boolean \rightarrow Boolean \rightarrow Boolean$
NAND	$Boolean \rightarrow Boolean \rightarrow Boolean$
$>$	$Double \rightarrow Double \rightarrow Boolean$
$<$	$Double \rightarrow Double \rightarrow Boolean$
$+$	$Double \rightarrow Double \rightarrow Double$
$-$	$Double \rightarrow Double \rightarrow Double$
$*$	$Double \rightarrow Double \rightarrow Double$
$/$	$Double \rightarrow Double \rightarrow Double$
AVG	$Integer \rightarrow Double$
MIN	$Integer \rightarrow Double$
MAX	$Integer \rightarrow Double$
LAG	$Integer \rightarrow Double$

Table 1.2. Terminals and their types used for both sets of GP runs.

Name	Type	Name	Type
INDEX	Double	RANDOM-INT	Integer
TRUE	Boolean	RANDOM-DOUBLE	Double
FALSE	Boolean	T	Time

We redefine the AND, NOR and IF-THEN-ELSE function as follow for GP runs without λ abstraction:

$$AND :: Boolean \rightarrow Boolean \rightarrow Boolean$$
$$NOR :: Boolean \rightarrow Boolean \rightarrow Boolean$$
$$IF - THEN - ELSE :: Boolean \rightarrow Boolean \rightarrow Boolean \rightarrow Boolean$$

Both sets of GP runs use the same control parameters given in Table 1.3. The GP system is generation-based, i.e. parents do not compete with offspring for selection and reproduction. We used tournament of size 2 to select winners. This means that two individuals are randomly selected and the one with a better fitness is the winner. The new population is generated with 50% of the individuals from cross-over, 40% from mutation (either point or sub-tree), and 10% from copy operator. The best individual is always copied over to the new generation. A GP run stops if no new best rule appears for 50 generation on validation data or the maximum number of generation (100) is reached.

Table 1.3. GP control parameters.

Parameter	Value	Parameter	Value
Tree Depth	4	Cross-over Rate	50
Population Size	200	Mutation Rate	40
Number of Runs	50	Copy Rate	10
Maximum Generation	100	Maximum Non-Improvement	50

Fitness Function

The fitness of an evolved GP trading rule is the *return* (R) it generates over the tested period. Initially, we are out of the market, i.e. holding no stock. Based on the trading decisions, buy and sell activities interleave throughout the time period until the end of the term when the stock will be forcibly closed. When in the market, it earns the stock market return. While out of the market, it earns a risk free interest return. The continuous compounded return over the entire period is the *return* which becomes the fitness of the GP trading rule.

There are three steps in computing the return. First, the GP rules is applied to the normalized time series to produce a sequence of trading decisions: *True* means to enter/stay in the market and *False* means to exit/stay out of the market. Second, this decision sequence is executed based on the original stock price series and the daily interest rate to calculate the compounded return. Last, each transaction (buy or sell) is charged with 0.25% fee, which is deducted from the compounded return to give the final fitness.

Let P_t be the S&P 500 index on day t, I_t be the interest rate on day t, and the return of day t is r_t:

$$r_t = \begin{cases} \log(P_t) - \log(P_{t-1}) & \text{, in the market} \\ I_t & \text{, out of the market} \end{cases}$$

Let n denotes the total number of transactions, i.e. the number of times a *True* (in the market) is followed by a *False* (out of the market) plus the number of times a *False* (out of the market) is followed by a *True* (in the market). Also, let c be the one-way transaction cost. The return over the entire period of T days is:

$$R = \sum_{t=1}^{T} r_t + n * \log \frac{1-c}{1+c}$$

In this study, the transaction fee c is 0.25% of the stock price. Compared to the transaction cost used by [1] (0.1%, 0.25% & 0.5%) and by [14] (0.12%), we have a reasonable transaction cost.

6. Results

Table 1.4 gives the returns from non-λ abstraction GP trading rules while Table 1.5 gives the returns from λ abstraction-GP trading rules. The last column of both tables give the returns from trading decisions made by the majority vote over all 50 trading rules, generated from the 50 different runs.

Table 1.4. Returns from non-λ abstraction GP trading rules on testing data.

seq	year	mean	stdev	median	max	min	majority vote
1	1989-1990	0.4910	0.2667	0.4021	1.2768	0.1681	0.5639
2	1991-1992	0.5032	0.2614	0.3640	1.0306	0.2688	0.4997
3	1993-1994	0.1776	0.1540	0.1286	0.5660	0.0477	0.1996
4	1995-1996	0.6058	0.1901	0.4964	0.9212	0.3257	0.6808
5	1997-1998	0.8678	0.4177	0.7913	1.8019	0.2392	0.9145
6	1999-2000	0.4787	0.4354	0.3667	1.7774	0.0665	0.5058
7	2001-2002	0.2608	0.5796	0.0852	1.9405	-0.4109	0.7599

Table 1.5. Returns from λ abstraction-GP trading rules on testing data.

seq	year	mean	stdev	median	max	min	majority vote
1	1989-1990	1.0353	0.2829	1.1287	1.2585	0.3081	1.1983
2	1991-1992	0.8377	0.2297	0.9507	0.9853	0.2120	0.9610
3	1993-1994	0.4479	0.1219	0.4905	0.5925	0.0477	0.5346
4	1995-1996	0.8007	0.1484	0.8537	0.9137	0.4548	0.9051
5	1997-1998	1.4917	0.4364	1.6450	1.8972	0.4976	1.8243
6	1999-2000	1.3321	0.5569	1.5488	1.9248	0.0665	1.6522
7	2001-2002	1.0167	0.7973	1.2671	1.9844	-0.1984	1.9651

Both sets of GP runs find trading rules that consistently out-perform buy-and-hold (whose returns are 0.1681, 0.2722, 0.0477, 0.4730, 0.5015, 0.0665, -0.4109 respectively for the 7 time periods). It is clear that their excess returns over buy-and-hold are statically significant. Also, λ abstraction-GP rules give higher returns than non-λ abstraction GP rules give. Moreover, trading decisions based on the majority vote over 50 rules give the best returns. These are encouraging results indicating GP is capable of finding profitable trading rules that out-perform buy-and-hold.

However, the GP rules returns are calculated without considering the following two biases: *trading cost bias* and *non-synchronous trading bias.*

Trading Cost Bias. The actual cost associated with each trading is not easy to estimate. One obvious reason is that different markets have different fees and taxes. Additionally, there are hidden costs involved in the collection and analysis of information. To work with such difficulty, break-even transaction cost (BETC) is proposed as an alternative approach to evaluate the profitability of a trading rule [11].

BETC is the level of transaction cost which offsets trading rule revenue and lead to zero profits. Once we have calculated BETC for each trading rule, it can be roughly interpreted as follows:

- large and positive: good;
- small and positive: OK;
- small and negative: bad;
- large and negative: interesting;

We will incorporate BETC to measure the profitability of the evolved GP trading rules in our future works.

Non-Synchronous Trading Bias. Non-synchronous trading is the tendency for prices recorded at the end of the day to represent the outcome of transactions that occur at different points in time for different stocks. Since the existence of thinly traded shares in the index can introduce non-synchronous trading bias, the observed returns might not be exploitable in practice. One way to test it out is by executing the trading rules based on trades occurring with a delay of one day. This could remove any first order autocorrelation bias due to non-synchronous trading [11]. This is another research topic in our future work.

Another way to evaluate the GP trading rules is applying them on a different financial index, such as NASDAQ 100. The returns might give insights about the rules and/or the stock markets themselves.

7. Analysis of GP Trading Rules

We examined all 50 rules generated from GP with λ abstraction modules on sequence 5 data and found most of them can be interpreted easily; each module is a trading indicator in various form. Depending on the number of λ abstraction modules it contains, a rule applies one to four indicators to make trading decisions (see Table 1.6). For example, $index > avg(28)$ is a *moving average* indicator which compares today's

index (divided by 250 moving average) with the average index (divided by 250 moving average) over the previous 28 days. Another example is $index > max(8)$, which is a filter rule that compares today's index (divided by 250 moving average) with the maximum index (divided by 250 moving average) of the previous 8 days.

The majority (27) of the 50 rules contain one trading indicator while the rest (23) use a combination of two to four indicators to make trading decisions. The most frequently used combinator is the AND function. This means many criteria have to be met before a stay-in-the-market decision (*True*) is issued. In other words, the GP rules evaluate the market trends using various indicators for trading decisions. Such sophisticated decision making process has led to more profitable trading.

In contrast, most (48) of the 50 rules generated from non-λ abstraction GP apply single indicator to make trading decisions. Although some of the single trading indicators can also give high returns (see Table 1.6), they are not always easy to find. Without the structure protection, forming trading indicator modules during evolution is not always easy. We have found many rules having branches under a combinator (such as AND) that are evaluated into constant value of *True* or *False*, instead of a meaningful indicator. This is very different from the results from the λ abstraction GP trading rules, where more meaningful indicators are evolved as λ abstraction modules under the branches of higher-order function combinators (AND & NOR & IF-THEN-ELSE).

Based on the analysis, we believe the λ abstraction module mechanism promotes the creation and combination of technical indicators. Such combination usage of different trading indicators give more sophisticated market evaluation and led to trades that generate higher returns.

We have also considered another possible benefit of λ abstraction module mechanism: it provides good seeding, which helps GP to find fitter trading rules. However, after examining the initial populations of all the GP runs, we find no evidence to support such hypothesis. Sometimes, λ abstraction-GP gives higher initial population fitness than the non-λ abstraction-GP does. Sometimes, it is the other way around.

8. Analysis of Transaction Frequency

As mentioned in Section 4, S&P 500 index inclines consistently between 1989 and 1995; bulls during the time period of 1996 to 1999 and declines after 2000. As expected, buy-and-hold gives the best returns during the year of 1996-1998 and the worst returns for 2001-2002 period.

Regardless of the stock markets climate, GP trading rules are able to identify opportunities to make profitable trading and out-perform buy-

Table 1.6. 50 λ abstraction GP trading rules trained by sequence 5 data.

fitness	quantity	rule
1.8972	2	$or(index > avg(2), index > lag(1))$
1.8937	1	$(index + index) > (avg(2) + avg(1))$
1.8535	1	$index > avg(1)$
1.8476	1	$if - then - else(max(1) < index, true, avg(3) < index)$
1.8059	7	$index > min(2)$
1.8034	1	$nand(if - then - else(index < avg(3), true, false),$ $if - then - else(index < min(2), true, false))$
1.7941	5	$index > avg(2)$
1.7941	1	$2 * index - avg(2) > min(21)$
1.7844	1	$and(or(avg(1) < index,$ $or(or(avg(6) < index, 1.30 < min(6)), avg(6) < index)), true)$
1.7002	1	$and(index > min(3), nand(index > avg(28), index < avg(3)))$
1.6936	1	$and(and(index < min(3), or(index > min(9),$ $index > min(11))), and(min(5) > index, true))$
1.6819	1	$or(index > 1.173, index > avg(2))$
1.6784	1	$nand(index < avg(5), index < min(3))$
1.6775	1	$nand(index < avg(4), nand(index < min(4),$ $nand(index < lag(4), nand(index < avg(13), true))))$
1.6126	1	$or(and(index > avg(5), true), and(index > min(5),$ $and(or(index > max(10), index < lag(5)), true)))$
1.5873	2	$index > min(4)$
1.5870	4	$index > avg(3)$
1.5539	1	$nand((0.00565 + index) < max(3), true)$
1.5149	1	$index > avg(4)$
1.5133	1	$and(min(5) < index,$ $nor((index + avg(175)) < (min(6) + avg(199)), false))$
1.5079	1	$and(index > min(13), and(index > min(5),$ $and(index > min(17), true)))$
1.4402	1	$and(index > min(8), or(nand(index > max(8),$ $nand(avg(165) < index, lag(45) > index)),$ $if - then - else(index > min(6), true, false)))$
1.4130	2	$index > avg(6)$
1.3283	1	$index < min(8)$
1.1427	1	$index > min(15)$
1.0650	1	$(0.01 + min(39)) < index$
0.7968	1	$2.44 * (index + index) > (avg(53) + (index * 3.86))$
0.7242	1	$index * index > avg(21)$
0.5996	1	$(index + 14.4) > (20.892/(0.28 + index))$
0.5611	1	$(index + 3.12) > (index/0.24)$
0.5611	1	$(min(84) + (8.8/index)) < ((index + 6.79) + lag(84))$
0.5015	2	$true(buy - and - hold)$
0.4976	1	$false$

and-hold. The average transaction frequency for non-λ abstraction GP rules is 22 for each testing period of 2 years. The frequency for λ ab-

straction GP rules are 3 time higher, with an average of 76 transactions in each testing period. In both cases, the higher the transaction frequency, the higher the return. This is demonstrated at the bottom half of Figure 1.5 and 1.6 where 3 cross plots from the 3 distinct time periods are given. With a reasonable transaction cost of 0.25%, GP is still able to find trading rules that explore many profitable trading opportunities. This endorses GP ability to find profitable trading rules that out-perform buy-and-hold.

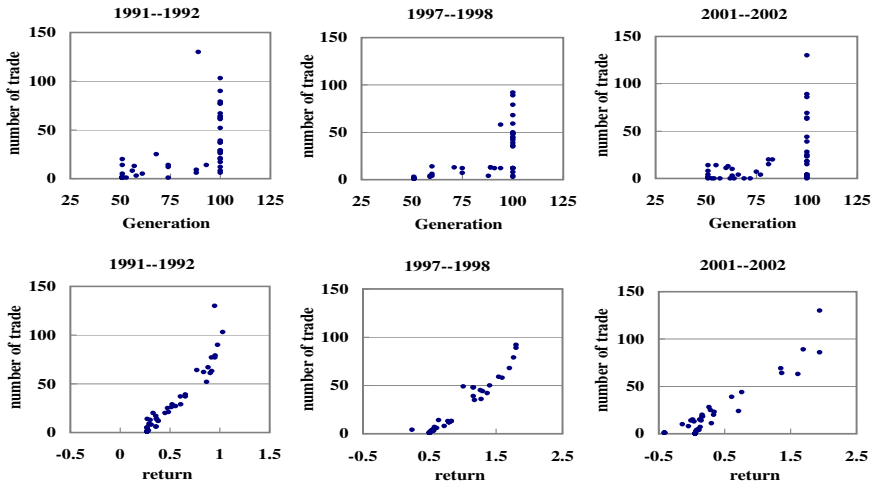

Figure 1.5. Transaction frequency vs. returns for non-λ abstraction GP rules.

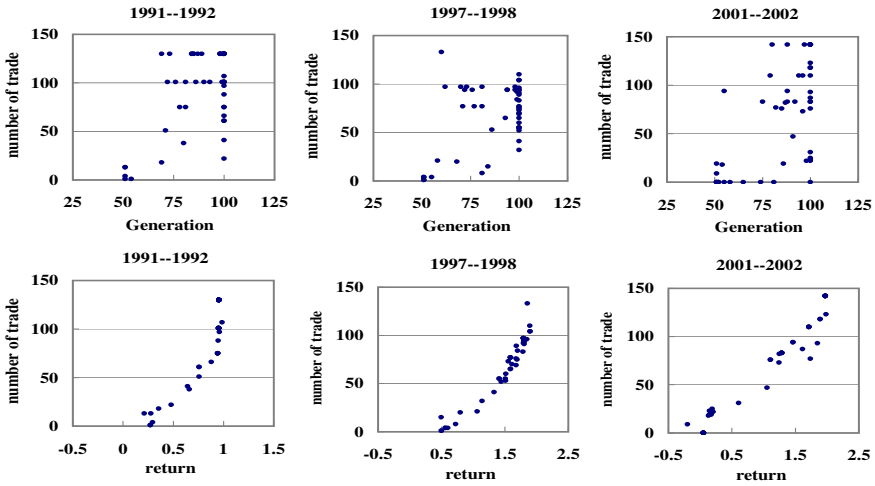

Figure 1.6. Transaction frequency vs. returns for λ abstraction GP rules.

We also compare the number of generations that each GP run lasts. As mentioned in Section 4, a GP run terminates when either no better rule on validation data is found for 50 generations or the maximum number of generation (100) is reached. This means the number of possible generations of a GP run is between 50 and 100. We have found that on average λ abstraction GP runs last 6 generations longer than non-λ abstraction GP runs. This means that λ abstraction GP is more able to continue to find fitter trading rules.

Do longer runs always generate better trading rules? The top half of Figure 1.5 shows that rules that give trading frequency large than 20 are generated by runs terminated at generation 100 (there are a couple of exceptions). In other words, longer runs generate trading rules that give higher trading frequency (> 20) and better returns. However, this pattern is not as evident in the λ abstraction GP runs (The top half of Figure 1.6). Some of the runs that terminate before generation 100 also generate trading rules that give high trading frequency (> 20) and good returns. Nevertheless, all runs that terminate at generation 100 give high trading frequency (> 20) which leads to good returns.

Figure 1.7 and 1.8 present the proportion of the 50 rules signaling a *True* (in the market) over the entire testing period. They give a visual representation of the degree of consensus among 50 rules and of the extent to which their decisions are coordinated. The λ abstraction rules have high consensus; most of the time 80% of the rules give the same decisions. In contrast, non-λ abstraction rules have a slightly lower degree of consensus: about 70% of the rules give the same decisions most of the time.

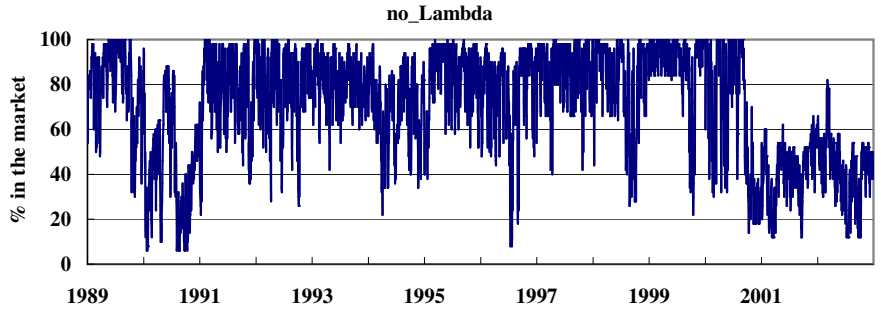

Figure 1.7. Proportion of non-λ abstraction GP rules signals"in the market".

Both sets of GP rules are able to identify market price trends. They signal mostly *True* (in the market) during the year between 1996 and 2000 when the market is up and mostly *False* (out of the market) during the year of 2001-2002 when the market is down.

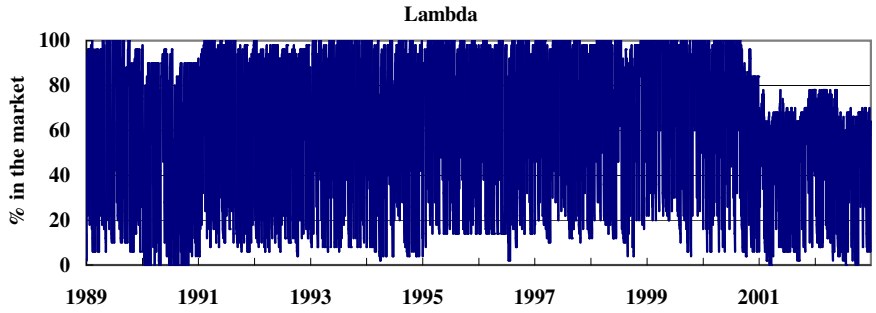

Figure 1.8. Proportion of λ abstraction GP rules signal "in the market".

9. Concluding Remarks

The application of λ abstraction GP to find technical trading rules based on S&P 500 index has generated many encouraging results:

- The GP trading rules give returns excess buy-and-hold with statistical significance.
- The GP trading rules can be interpreted easily; they use one to four widely used technical indicators to make trading decisions.
- The GP trading rules have high consensus; 80% of the rules give the same decision for most of the testing period.
- The GP trading rules are able to identify market price trends; they signal mostly *True* (in the market) during the year between 1996 and 2000 when the market is up and mostly *False* (out of the market) during the year of 2001-2002 when the market is down.
- The GP rules give high transaction frequency. Regardless of market climate, they are able to identify opportunities to make profitable trades.

These are strong evidences indicating GP is able to find profitable technical trading rules that out-perform buy-and-hold. This is the first time such positive results on GP trading rules are reported.

Various analysis show that λ abstraction module mechanism promotes the creation and combination of technical indicators during the GP evolution process. With the module structure protection, meaningful technical indicators are formed as λ abstraction modules. Such combination usage of different technical indicators give more sophisticated market evaluation and led to trades that generate higher returns.

Future Work

The evolved GP trading rules give strong evidences that there are patterns in the S&P 500 time series. These patterns are identified by GP as various forms of technical indicators, each of which is captured in a λ abstraction module. This feature is exhibited in all the rules generated from 50 GP runs.

These patterns, however, do not seem to exist in the initial population. Instead, it is through the continuous merging (cross-over) and modification (mutation) of the same kind of modules for a long time (100 generations) when meaningful technical indicators were formed.

Based on these application results, we are planning on a theoretical work to formally define the convergence process of the λ abstraction GP:

- Define each indicator in the 50 GP rules as a building block;
- Formulate the steps to find one of the 50 rules.

We are not certain if such a theory is useful, since we might not be able to generalize it beyond this particular application or data set. Nevertheless, we believe it is a research worth pursuing.

Acknowledgments

We wish to thank John Koza and Mike Caplan for their comments and suggestions to improve this work.

References

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

[2] Bhattacharyya, Siddhartha, Pictet, Olivier V., and Zumbach, Gilles (2002). Knowledge-intensive genetic discovery in foreign exchange markets. *IEEE Transactions on Evolutionary Computation*, 6(2):169–181.

[3] Brock, William, Lakonishok, Josef, and LeBaron, Blake (1992). Simple technical trading rules and the stochastic properties of stock returns. *Journal of Finance*, 47(5):1731–1764.

[4] Church, Alonzo (1941). *The Calculi of Lambda Conversion*. Princeton University Press.

[5] Cooper, Michael (1999). Filter rules based on price and volume in individual security overreaction. *The Review of Financial Studies*, 12(4):901–935.

[6] Kaboudan, M. (2002). GP Forecasts of Stock Prices for Profitable Trading. In Chen, Shu-Heng, editor, *Evolutionary Computation in Economics and Finance*, Physica-Verlag.

[7] Koza, John R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection.* MIT Press, Cambridge, MA, USA.

[8] Koza, John R. (1994). *Genetic Programming II: Automatic Discovery of Reusable Programs.* MIT Press, Cambridge Massachusetts.

[9] Neely, Christopher J., Weller, Paul A., and Dittmar, Rob (1997). Is technical analysis in the foreign exchange market profitable? A genetic programming approach. *The Journal of Financial and Quantitative Analysis*, 32(4):405–426.

[10] O'Neill, Michael, Brabazon, Anthony, and Ryan, Conor (2002). Forecasting market indices using evolutionary automatic programming. In Chen, Shu-Heng, editor, *Genetic Algoritms and Genetic Programming in Computational Finance*, chapter 8, pages 175–195. Kluwer Academic Publishers.

[11] Pereira, R. (2002). Forecasting Ability But No Profitability: An Empirical Evaluation of Genetic Algorithm-Optimised Technical Trading Rules. In Chen, Shu-Heng, editor, *Evolutionary Computation in Economics and Finance,*, Physica-Verlag.

[12] Pesaran, M. Hashem and Timmermann, Allan (1995). Predictability of stock returns: Robustness and economic significance. *Journal of Finance*, 50:1201–1228.

[13] Pring, Martin J. (1991). *Technical Analysis Explained.* McGraw-Hill Trade.

[14] Wang, Jun (2000). Trading and hedging in s&p 500 spot and futures markets using genetic programming. *The Journal of Futures Markets*, 20(10):911–942.

[15] Yu, Gwoing Tina (1999). *An Analysis of the Impact of Functional Programming Techniques on Genetic Programming.* PhD thesis, University College, London, Gower Street, London, WC1E 6BT.

[16] Yu, Tina (2001). Hierachical processing for evolving recursive and modular programs using higher order functions and lambda abstractions. *Genetic Programming and Evolvable Machines*, 2(4):345–380.

[17] Yu, Tina, Chen, Shu-Heng, and Kuo, Tzu-Wen (2004). A genetic programming approach to model international short-term capital flow. *to appear in a special issue of Advances in Econometrics.*