Chapter 2

DISCOVERING FINANCIAL TECHNICAL TRADING RULES USING GENETIC PROGRAMMING WITH LAMBDA ABSTRACTION

Tina Yu[1], Shu-Heng Chen[2] and Tzu-Wen Kuo[2]
[1]*ChevronTexaco Information Technology Company;* [2]*National Chengchi University, Taiwan*

Abstract We applied genetic programming with a lambda abstraction module mechanism to learn technical trading rules based on S&P 500 index from 1982 to 2002. The results show strong evidence of excess returns over buy-and-hold after transaction cost. The discovered trading rules can be interpreted easily; each rule uses a combination of one to four widely used technical indicators to make trading decisions. The consensus among these trading rules is high. For the majority of the testing period, 80% of the trading rules give the same decision. These rules also give high transaction frequency. Regardless of the stock market climate, they are able to identify opportunities to make profitable trades and out-perform buy-and-hold.

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

1. Introduction

In this chapter genetic programming (GP) (Koza, 1992) combined with a lambda abstraction module mechanism is used to find profitable trading rules in the stock market. 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 price

change is technical analysis. This approach uses historical stock prices and volume data to identify the price trend in the market. Originated from the work of Charles Dow in the late 1800s, technical analysis is now widely used by investment professionals to make trading decisions (Pring, 1991).

Various trading indicators have been developed based on technical analysis. Examples are *moving average*, *filter* and *trading-range break*. For the moving average class of indicators, 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 indicator can be implemented in many different ways by specifying different short and long periods. For example, on the left side of Figure 2-1 is a moving average with a short of 10 days and a long of 50 days. For the filter indicators, 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. Similar to the moving average, it can be implemented with different time length. When multiple filter indicators are combined together similar to the one on the right side of Figure 2-1, it is called a *trading-range break indicator*.

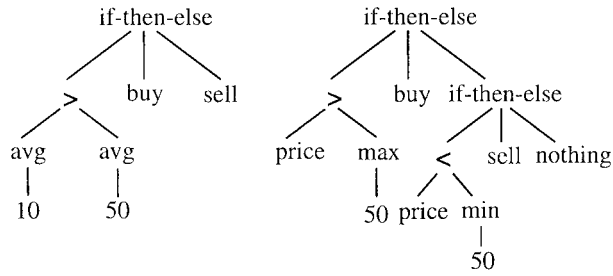

Figure 2-1. A moving average (10,50) and a trading-range break indicator.

Previously, (Brock et al., 1992) reported that *moving average* and *trading-range break* give significant positive returns on Dow Jones index from 1897 to 1986. Similarly, (Cooper, 1999) showed that *filter* strategy can out-perform buy-and-hold under relatively low transaction cost on NYSE and AMEX stocks for the 1962-1993 period. These studies are encouraging evidence indicating that it is possible to devise profitable trading rules for stock markets.

However, one concern regarding 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 period. If the investor had 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, GP 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 (see Section

2 for details). One possible reason for this outcome is that the GP systems used are not adequate for this task. The work described in this chapter extends GP with a λ abstraction module mechanism and investigates 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 2 reviews related work. Section 3 presents the λ abstraction module mechanism. In Section 4, the PolyGP system is described. In section 5, S&P 500 time series data are given. Section 6 explains the experimental setup while Section 7 presents the experimental results. We analyze the GP trading rules in Section 8 and 9. Finally, concluding remarks are given in Section 10.

2. Related Work

Targeted toward different financial markets, different researchers have applied GP to generate trading rules and analyzed their profitability. For example, (Allen and Karjalainen, 1999) studied S&P 500 index from 1928 to 1995. They reported that the evolved GP trading rules did not earn consistent excess returns over after transaction costs. In contrast, (Neely et al., 1997) reported that their GP trading rules for foreign exchange markets were able to gain excess returns for six exchange rates over the period 1981-1995. (Wang, 2000) 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 (Neely et al., 1997) did not use the rolling forward method (explained in Section 5) to test their results for different time periods while (Allen and Karjalainen, 1999) did. Finally, Wang pointed out that these two works used different benchmarks to assess their GP trading rules: (Allen and Karjalainen, 1999) used the return from buy-and-hold while (Neely et al., 1997) 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 (Allen and Karjalainen, 1999), Wang 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 were not able to beat buy-and-hold in both cases. Additionally, he also incorporated Automatically Defined Functions (ADFs) (Koza, 1994) in his GP experiments. He reported that ADFs made the representation of the trading rules simpler by avoiding duplication of the same branches. However, no comparison was made between the returns from GP rules and the returns from ADF-GP rules.

Another approach using GP to generate trading rules is by combining predefined trading indicators (Bhattacharyya et al., 2002, O'Neill et al., 2002). In these works, instead of providing functions such as *average* for GP to construct a moving average indicator and *minimum* to construct filter indicators, 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, which are then combined to make trading decisions.

Our first attempt using modular GP to evolve financial trading rules was based on ADF-GP (Yu et al., 2004). There, the evolved rules trade in both stock markets and foreign exchange markets simultaneously. However, our study results showed that most ADF modules were evaluated into constant value of *True* or *False*. In other words, ADFs did not fulfill the role of identifying modules in the trading rules. Consequently, ADF-GP trading rules gave similar returns to those from vanilla GP trading rules; both of them were not as good as the returns from buy-and-hold. This suggests either that 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 technical indicators to make trading decisions. We therefore decide to investigate a different modular approach (λ abstraction) to better understand GP's ability in finding profitable trading rules.

3. Modular GP through Lambda Abstraction

Lambda abstractions are expressions defined in λ calculus (Church, 1941) that represent function definition (see Section 4 for the syntax). Similar to a function definition in other programming languages such as C, a λ abstraction can take inputs and produce outputs. In a GP program tree, each λ abstraction is treated as an independent module, with a unique identity and purpose. It is protected as one unit throughout the program evolution process.

One way to incorporate λ abstraction modules in GP is using higher-order functions, i.e., 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 2-2 gives two program trees. Each contains two different kinds of λ abstraction modules: one is represented as a triangle and the other as a cycle. Cross-over operations are only permitted between modules of the same kind.

We use *type information* to distinguish different kind of λ abstraction modules. Two λ abstractions are of the same kind if they have the same number of inputs and outputs, with the same input and output types. For example, in this

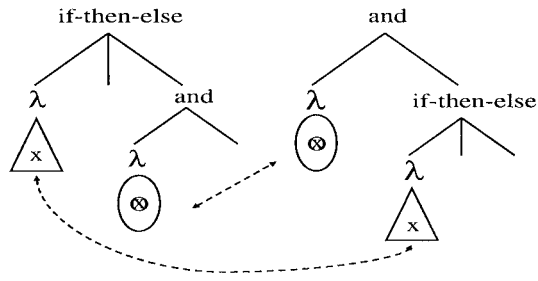

Figure 2-2. Cross-over between λ abstraction modules in two GP trees.

work we define a λ abstraction with type information $Time \rightarrow Boolean$: it takes one input with type *Time* and returns a *Boolean* output.

Unlike an ADF, whose position in a program tree is determined by evolution, a λ abstraction module is hard-wired to sit underneath a specified function node. Therefore, this module mechanism can be use to incorporate domain knowledge to design GP tree structure. In this work, we want GP to combine multiple technical indicators. To achieve that, we first add Boolean function combinators AND, OR, NAND, NOR to the function set. Additionally, we specify some of the combinators as higher-order functions. In this way, technical indicators can be evolved inside λ modules, which are then integrated together by the higher-order function combinators.

Incorporating domain knowledge to design can speed up the GP evolution process, and leads to faster discovery of meaningful solutions. In a previous work, a similar concept was used to design recursive program structure for the general even parity problem. With a very suitable design, the population program structures were quickly converged (in the first generation) and most GP evolution effort went to find the correct program contents (Yu, 2001).

4. The PolyGP System

PolyGP (Yu, 1999) 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 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 returns 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 with type *Time* and the other 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 program trees. A program 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 tree will be created. 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. This process is repeated many times until the permitted tree depth is reached.

λ abstraction trees are created using a similar procedure. The only difference is that their terminal set consists not only of the terminal set used to create the main program, but also the input variables to the λ abstraction. Input variable naming in λ abstractions follows a simple rule: each input variable 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 abstraction trees with the same number and the same type of inputs and outputs.

5. S&P 500 Index Time Series Data

From *Datastream*, we acquired the S&P 500 index time series data between January 1, 1982 and December 31, 2002. Since the original time series is non-stationary, we transformed it by dividing the daily data by its 250-day moving average. This is the same method used by (Allen and Karjalainen, 1999) and (Neely et al., 1997). The adjusted data oscillate around 1 and make the modeling task easier.

A different approach to normalize financial time series is converting the price series into a return series . This is done by calculating the price difference between two consecutive days (first-order difference) in the original price series. Whether financial modeling should be based on price series or return series is still a subject under much debate (Kaboudan, 2002). We adopt the approach used by previous GP works on modeling technical trading rules so that we can make sensible performance comparisons.

Figure 2-3 gives the original and the transformed time series. There are three distinct phases in this time series. From 1982 to 1995, the market grew consistently; between 1996 and 1999, the market bulled; after 2000, the market declined. With such diversity, this data set is suitable for GP to model trading rules.

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

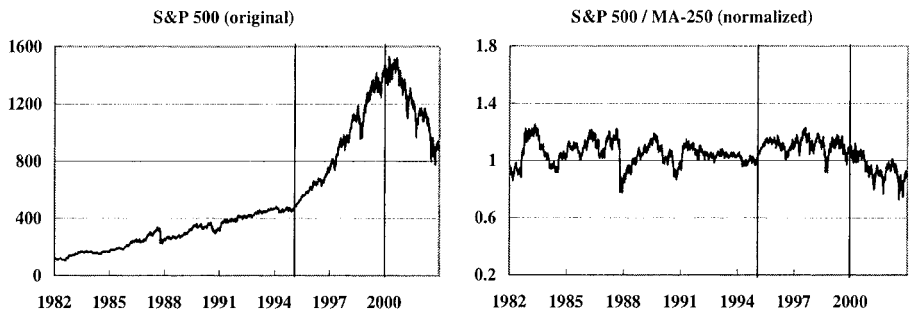

Figure 2-3. Time series data before and after normalization.

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 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 (Pesaran and Timmermann, 1995) and used by (Allen and Karjalainen, 1999) and (Wang, 2000).

To start, we reserved 1982 data to be referred to by time series functions such as *lag*. The remaining time series were then organized into 7 sequences, each of which was used to make an independent GP run. In each sequence, the training period is 4 years long, validation period is 2 years and testing period is 2 years. The data in one sequence may overlap the data in another sequence. As shown in Figure 2-4, the second half of the training period and the entire validation period of the first sequence are the training period of 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 2 years, and covers a different time period from 1989 to 2002.

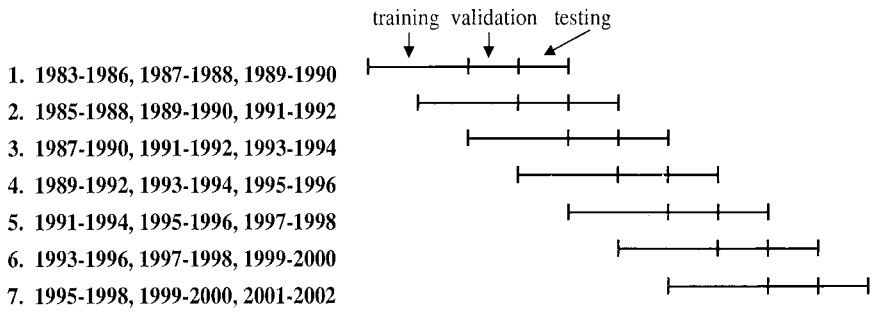

Figure 2-4. Training, validation and testing periods for 7 time sequences.

For each data series, 50 GP runs were made. The three data periods are used in the following manner:

1 The best trading rule against the training period at the initial population is selected and evaluated against the validation period. This is the initial "best rule".

2 A new generation of trading rules are created by recombining/modifying parts of relatively fit rules in the previous generation.

3 The best trading rule against the training period at the new population is selected and evaluated against the validation period;

4 If this rule has a better validation fitness than the previous "best rule", this is the new "best rule".

5 Goto step 2 until the maximum number of generation is reached or there is no fitter rule found after a certain number of generations (50 in this study).

6 The last "best rule" is evaluated against the testing period. This is what we use to evaluate the performance of the GP trading rule.

In summary, 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 to the testing period data to give the performance of the rule. The evaluation of performance of the GP trading rules is based on the results from testing periods.

6. 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 takes one input with type *Time* and returns a *Boolean* output. As described before, this function argument will be created as a 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, is a function with a different type. Its left branch is therefore only allowed to cross-over with the left branch of an IF-THEN-ELSE node in another GP tree. We constrain a GP tree to have a maximum of 4 higher-order functions to preserve computer memory usage.

Tables 2-1 and 2-2 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(t,250)* 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(t,3)* 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(t,3)* 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. Also, the ranges for integer values are 0 and 250 while the ranges for double values are 0 and 1.

Table 2-1. 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	$Time \rightarrow Integer \rightarrow Double$
MIN	$Time \rightarrow Integer \rightarrow Double$
MAX	$Time \rightarrow Integer \rightarrow Double$
LAG	$Time \rightarrow Integer \rightarrow Double$

Table 2-2. Terminals and their types used for both sets of GP runs but rlr added some text to make this table caption go more than one line to see if that is also centered.

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

For GP runs without λ abstractions, we redefine the AND, NOR and IF-THEN-ELSE functions as follows:

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

Both sets of GP runs used the same control parameters given in Table 2-3. The GP system is generation-based, *i.e.* parents do not compete with offspring for selection and reproduction. We used a tournament of size 2 to select winners. This means that two individuals were randomly selected and the one with a better fitness was the winner. The new population was generated with 50% of the individuals from cross-over, 40% from mutation (either point or sub-tree), and 10% from the copy operator. The best individual was always copied over to the new generation. A GP run stopped if no new best rule appeared for 50 generation on validation data, or the maximum number of generations (100) was reached.

Table 2-3. 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* (R) which becomes the fitness of the GP trading rule.

There are three steps in computing the return of a GP trading rule. First, the GP rule is applied to the normalized time series to produce a sequence of trading decisions: *True* directs the trader 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 a 0.25% fee, which is deducted from the compounded return to give the final fitness.

Let P_t be the S&P 500 index price 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 denote 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 (Allen and Karjalainen, 1999) (0.1%, 0.25% & 0.5%) and (Wang, 2000) (0.12%), we have a reasonable transaction cost.

7. Results

Table 2-4 gives the returns from non-λ abstraction GP trading rules while Table 2-5 gives the returns from λ abstraction-GP trading rules. The last column in both tables gives the returns from trading decisions made by the majority vote over the 50 trading rules, generated from 50 different GP runs.

Table 2-4. 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 2-5. 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[1]. 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. Moreover, trading decisions based on the majority vote by 50 rules give the highest returns. These are encouraging results indicating that GP is capable of finding profitable trading rules that out-perform buy-and-hold.

[1]With buy-and-hold , 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. This is the most frequently used benchmark to evaluate the profitability of a financial trading rule. Buy-and-hold returns for the 7 testing periods are 0.1681, 0.2722, 0.0477, 0.4730, 0.5015, 0.0665, -0.4109 respectively

However, the GP rules returns may have two possible biases, from *trading costs* and *non-synchronous trading*.

Trading Cost Bias. The actual cost associated with each trade 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) has been proposed as an alternative approach to evaluate the profitability of a trading rule (Kaboudan, 2002).

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 work.

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. The existence of thinly traded shares in the index can introduce non-synchronous trading bias. As a result, the observed returns might not be exploitable in practice. One way to test this is to execute 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 (Pereira, 2002). This is a research topic in our future work.

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

8. 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 of some form. Depending on the number of λ abstraction modules it contains, a rule applies one to four indicators to make trading decisions (see Table 2-6). For example, $index > avg(28)$ is a *moving*

Table 2-6. 50 λ abstraction GP trading rules trained by sequence 5 data.

fitness	quantity	rules (after minor editing of non-executing code)
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	$false$

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

Among the 50 λ abstraction GP trading rules, 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 use various indicators to evaluate the market trend and to make trading decisions. Such a 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 a single indicator to make trading decisions. Although some of the single trading indicators can also give high returns (see Table 2-6), they are not always easy to find. Without the structure protection, forming meaningful trading indicators 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 λ abstraction GP trading rules, where more meaningful indicators were 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 combined usage of different trading indicators gives a more, and leads to trades that generate higher returns.

We have also considered another possible benefit of the 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 a hypothesis. Sometimes, λ abstraction-GP gives higher initial population fitness than the non-λ abstraction-GP does. Sometimes it is the other way around.

9. Analysis of Transaction Frequency

As mentioned in Section 5, the S&P 500 index grew consistently between 1989 and 1995, bulled from 1996 to 1999 and declined after 2000. As expected, buy-and-hold gives the best return during the years 1996-1998 and the worst returns for the 2001-2002 period.

Regardless of the stock market's climate, GP trading rules were able to identify opportunities to make profitable trading. The average transaction frequency for non-λ abstraction GP rules is 22 for each testing period of 2 years: about

one transaction every 33 days. The frequency for λ abstraction GP rules is 3 times 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 2-5 and 2-6 where 3 cross plots from the 3 distinct time periods are given.

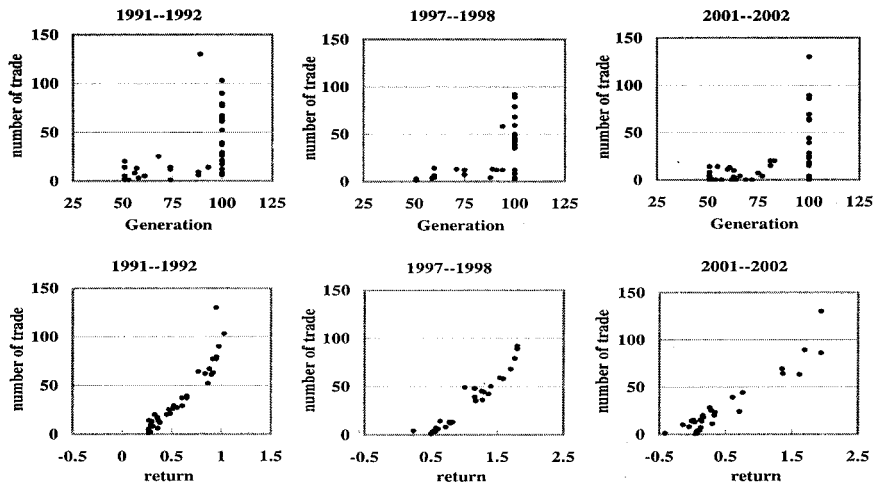

Figure 2-5. Transaction frequency *vs.* returns for non-λ abstraction GP rules.

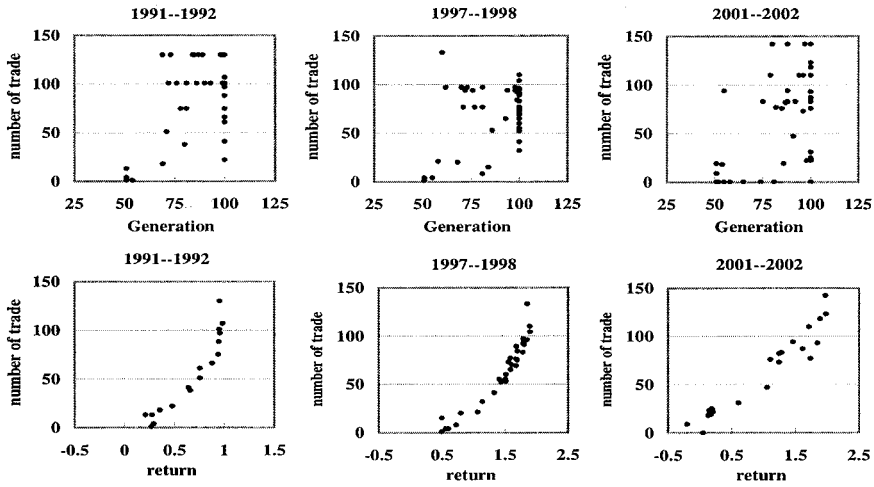

Figure 2-6. Transaction frequency *vs.* returns for λ abstraction GP rules.

We also compare the number of generations that each GP run lasted. As mentioned in Section 5, a GP run terminated when either no better rule on vali-

dation data was found for 50 generations or the maximum number of generation (100) has reached. This means that the number of possible generations of a GP run is between 50 and 100. We have found that on average λ abstraction GP runs lasted 6 generations longer than non-λ abstraction GP runs. This indicates that λ abstraction GP is better able to continue to find fitter trading rules.

Do longer runs always generate better trading rules? The top half of Figure 2-5 shows that non-λ abstraction GP rules which give higher than 20 were generated by runs terminated at generation 100 (there are a couple of exceptions). In other words, longer runs generated trading rules that gave 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 2-6). Some of the runs that terminated before generation 100 also generated trading rules that gave high trading frequency (> 20) and good returns. Nevertheless, all runs that terminated at generation 100 gave high trading frequency (> 20) which led to good returns.

Figure 2-7 and 2-8 present the proportion of the 50 trading 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; during most of the testing period, 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 over the majority of the testing period.

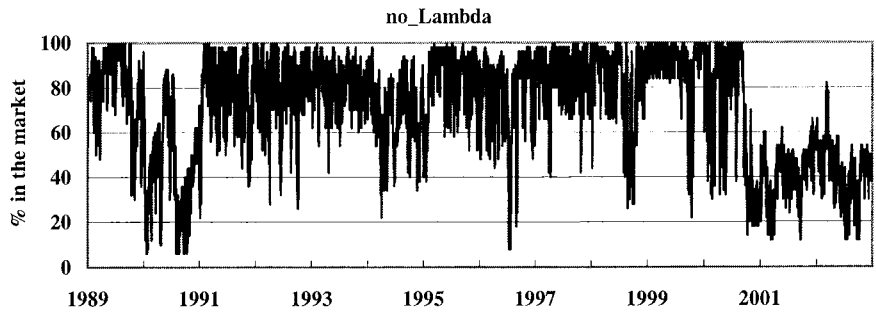

Figure 2-7. Proportion of non-λ abstraction GP rules signals"in the market".

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

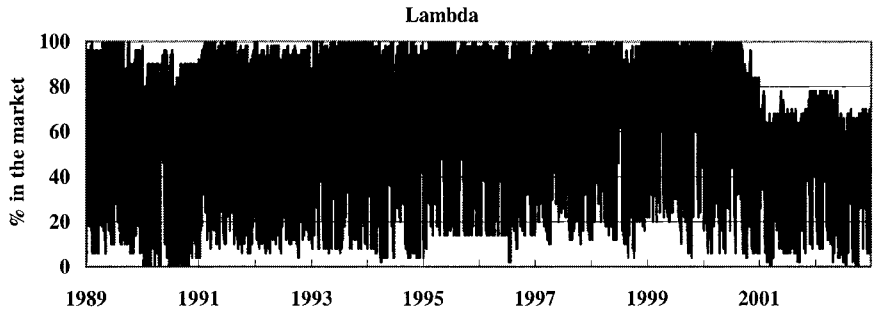

Figure 2-8. Proportion of λ abstraction GP rules signal "in the market".

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

These are strong evidence 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 indicates that the λ abstraction module mechanism promotes the creation and combination of technical indicators in the GP trading rules. Such combination of different trading indicators gives more sophisticated market evaluation and leads to trades that generate higher returns.

Lambda abstraction is a module mechanism that can incorporate domain knowledge to design program structures. When properly used, it leads to the

discovery of good and meaningful solutions. This chapter gives one such example, in addition to the example of even parity problem reported in (Yu, 2001). We anticipate there are more such domain-knowledge-rich problems that the λ abstraction module mechanism can help GP to solve.

Future Work

The evolved GP trading rules give strong evidence 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.

References

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

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.

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.

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

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

Kaboudan, Mak (2002). Gp forecasts of stock prices for profitable trading. In *Evolutionary Computation in Economics and Finance*, pages 359–382. Physica-Verlag.

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

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

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.

O'Neill, Michael, Brabazon, Anthony, and Ryan, Conor (2002). Forecasting market indices using evolutionary automatic programming. In *Genetic Algoritms and Genetic Programming in Computational Finance*, pages 175–195. Kluwer Academic Publishers.

Pereira, Robert (2002). Forecasting ability but no profitability: An empirical evaluation of genetic algorithm-optimised technical trading rules. In *Evolutionary Computation in Economics and Finance*, pages 275–295. Physica-Verlag.

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

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

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.

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.

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.

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.*

http://www.springer.com/978-0-387-23253-9

Genetic Programming Theory and Practice II
O'Reilly, U.-M.; Yu, T.; Riolo, R.; Worzel, B. (Eds.)
2005, XVI, 320 p., Hardcover
ISBN: 978-0-387-23253-9