SANDIA REPORT

SAND2004-4774
Unlimited Release
Printed October 2004

Successful Technical Trading Agents Using Genetic Programming

Grant V. Farnsworth
John A. Kelly
Andrew S. Othling
Richard J. Pryor

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550

Sandia is a multiprogram laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of Energy's
National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Issued by Sandia National Laboratories, operated for the United States Department of Energy by Sandia Corporation.

Available to DOE and DOE contractors from
U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@adonis.osti.gov
Online ordering: http://www.doe.gov/bridge

Available to the public from
U.S. Department of Commerce
National Technical Information Service
5285 Port Royal Rd
Springfield, VA 22161

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.fedworld.gov
Online ordering: http://www.ntis.gov/ordering.htm

SAND2004-4774
Unlimited Release
Printed October 2004

Successful Technical Trading Agents Using Genetic Programming

Grant V. Farnsworth, John A. Kelly,
Andrew S. Othling, and Richard J. Pryor
Evolutionary Computing and Agent Based Modeling
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1109

Abstract

Genetic programming (GP) has proved to be a highly versatile and useful tool for identifying relationships in data for which a more precise theoretical construct is unavailable. In this project, we use a GP search to develop trading strategies for agent based economic models. These strategies use stock prices and technical indicators, such as the moving average convergence/divergence and various exponentially weighted moving averages, to generate buy and sell signals. We analyze the effect of complexity constraints on the strategies as well as the relative performance of various indicators. We also present innovations in the classical genetic programming algorithm that appear to improve convergence for this problem. Technical strategies developed by our GP algorithm can be used to control the behavior of agents in economic simulation packages, such as ASPEN-D, adding variety to the current market fundamentals approach. The exploitation of arbitrage opportunities by technical analysts may help increase the efficiency of the simulated stock market, as it does in the real world. By improving the behavior of simulated stock markets, we can better estimate the effects of shocks to the economy due to terrorism or natural disasters.

Acknowledgments

This work was funded by the United States Department of Homeland Security. We would like to thank members of the Evolutionary Computing Department at Sandia National Laboratories for their constructive comments and insight.

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

Contents

Figures

Successful Technical Trading Agents Using Genetic Programming

Introduction

Agent based economic simulations depend on the aggregation of individual decisions by the agents to model larger scale phenomena. In order to achieve accurate macro predictions it is not strictly necessary that the agent decisions accurately model the complexity and variety of individual human choices. It is often, however, necessary that the major groups of individuals (and their respective strategies) are represented. In financial markets there are two classes of strategies: fundamental and technical trading. Fundamental trading estimates the value of an asset based on information about current and future cash flows and tries to hold assets whose net present value exceeds their current price. Because so many agents constantly estimate the worth of financial assets, there is little opportunity to make excess returns (above the risk premia for that asset) of stocks based on publicly available information. Some traders believe that predictable patterns emerge in prices over time. For example, if the market adjusts slowly to positive news about a particular asset, a trader noticing the beginning of an upward trend can buy while the asset is still undervalued. Trading based on patterns in the asset price, volume traded, and other market data, rather than news about the asset's expected returns, is called technical trading. Functions of this data that are used in technical trading are called technical indicators.

The economic simulation package ASPEN-D simulates the behavior of firms and household agents to model US financial markets. We are especially interested in equity markets. In this model, households use only fundamental strategies to make stock purchases. While the majority of stock trades in the real world are likely based on market fundamentals, there are probably sufficient technical traders to have a significant affect on the behavior of the market as a whole. In order to incorporate traders who mimic this effect we need to create successful technical trading rules.

Historically, the consensus in the academic literature has been that equity markets are efficient, meaning there is no systematic way for discovering successful trading rules based on purely technical indicators. Advancements in computing technology and data mining techniques have, on the whole, confirmed this hypothesis. Nevertheless, there is a growing interest in the use of genetic algorithms as a tool to discover patterns in the noise found in financial data. Genetic programming (GP), in particular, has the ability to find highly nonlinear and nonintuitive relationships using only moderate computing resources. Several attempts have been made in recent years to use this powerful algorithm to discover technical trading rules, but the most prominent studies have failed to consistently produce trading rules that outperformed

buy-and-hold strategies in either a profit maximizing or risk adjusted sense[1][5].

Genetic programming methods often need extensive customization in order to effectively solve a particular class of problem. The solution space to be searched, the specifics of the genetic operations, and the GP parameters can have a significant impact on the success of the search. Recent papers use the same data as previous attempts and present evidence that past failure to identify successful trading rules was a result of GP implementation specifics, rather than the impossibility of the problem[2][3].

We incorporate techniques from past research as well innovations to the GP algorithm in order to develop technical trading strategies based on real stock index data and evaluate the efficacy of various indicators and parameters in generating successful trading rules. The resulting GP engine can then be used on simulated data to generate rules to drive the trading strategy of a class of technical traders in ASPEN-D. This will add variety and, hopefully, efficiency to the model, paving the way for experiments to determine how financial markets would react to various negative shocks, such as terrorist attacks.

Overview of Genetic Programming

Genetic Programming is an evolutionary optimization algorithm, which means it solves problems by mimicking the process of evolution[4]. A population of candidate solutions is maintained and modified in such a way that the weak solutions are removed and replaced by variants of the strongest until little or no progress can be made over the best solution in the population. When this occurs we say that the problem has converged to a feasible solution.

Population Construction

The first step in this process of evolution is the creation of a population of randomly generated candidate solutions. Each candidate is evaluated according to its ability to solve the problem and assigned a fitness value. The population is sorted according to fitness so that fitter solutions are put at the top of the population. The individuals at the top of the population are reproduced with greater probability than those at the bottom. The population is then modified via mutation and crossover, processes which mimic the effects of sexual reproduction and adaptation in nature. Each iteration of this process is called a generation. After many generations, the population will tend toward high fitness.

Each of the solutions in the population can be described in the form of a tree. A tree is a software construction comprised of mathematical operators and data linked

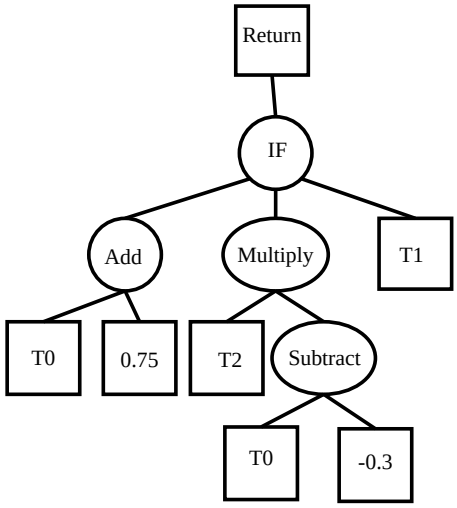


Figure 1. An example GP tree.

together (see Figure 1). The bottom of the tree consists of terminal nodes. These nodes reference elements of data, such as asset price values or technical indicators such as an exponential moving average. Alternately, they can contain a random constant.

In each generation, the entire population of trees is evaluated, substituting the corresponding data and indicator values into the terminals. The terminals pass their data up to the operator nodes, which perform mathematical operations on them and pass the result to the operator nodes above them. Finally, a resulting value is passed to the return node at the top of each tree, serving as the value of the tree at that time index. For each tree, the return values for all times are used by the fitness function. The tree may also be divided into subtrees, each of which consists of all operators and terminals below a given node.

Simulated Evolution

Crossover is generally considered the most important genetic operation and is analogous to breeding in nature. Two random trees are chosen from the population and random subtrees are selected from each of those trees. The subtrees are then switched, creating two new individuals. Alternately, a subtree may be simply copied from one tree into the second, creating one new individual. Since subtrees are selected randomly, this operation can potentially increase the depth of a tree tremendously. For this reason a tree depth limitation may be imposed. If a tree is crossed over and exceeds the tree depth, the bottom levels may be cut off or alternately the crossover may be disallowed.

The second genetic operation is mutation. We consider two varieties: node mutation and tree mutation. In node mutation, the program chooses a few random nodes in the tree and changes them, with some probability, to nodes that take the same number of arguments. This retains the shape of the tree. Operator nodes change operations and terminals may change which indicator they refer to or which constant they produce. Instead of regenerating constants from the uniform distribution, as in the creation of the tree, node mutation changes the value of the constant by a small, random amount.

Tree mutation selects a random subtree, deletes the subtree, and recreates a new subtree for that location. Since this does not preserve the shape of the tree, this operation can result in very large trees. For this reason, a restriction on the number of nodes or levels in the resulting trees is usually applied to tree mutation operations. If the resulting tree exceeds one of these limitations, it may be truncated, or the operation may be rejected. In our experiments, the latter option generally produced fitter trees.

Our method of controlling the population of candidates from one generation to another differs somewhat from the classical GP setup, and most innovations were made in the interest of minimizing memory and computational requirements. First, we generate a random population. In general, the larger the population the better. In fact, performance of GP algorithms is usually more closely linked to population size than number of generations[4]. The population is then sorted according to the fitness criterion, which in our case requires evaluation of the tree at each S&P500 closing price and calculating the performance of the resulting strategy. Some percentage of the population is then killed (deleted), starting at the bottom. The deleted poor performers are then replaced with relatively high performing trees from the population. This kill-replace operation is performed before other genetic operations in each generation and generally increased the convergence of our experiments. We typically killed about a fourth of the population. After this is done, we traverse the population, performing genetic operations probabilistically on each tree. Typical probabilities are 50% crossover, 20% node mutation, 10% tree mutation, 10% kill and regenerate, 10% leave as is. The top few performers are left alone to ensure that the top fitness in the population never decreases. If crossover is chosen, the partner tree is selected from a probability distribution that heavily favors trees on the high performing end. In this way high performing genes (in the form of subtrees) propagate more quickly through the population. After the genetic operations are completed, the population is re-sorted and we enter the next generation.

After a few hundred generations, the highest fitness tree serves as a feasible solution and the population may lack sufficient genetic diversity to improve on it significantly. This is generally a good stopping point, but if we want to continue searching for even better trees, we can kill all the trees in the population except for the top few, regenerate, and continue to the next generation. Although the resulting population will have a low average fitness, the wiping clean operation, which we call "the plague"

serves to jump-start the diversity of the gene pool and helped combat genetic stagnation in our experiments. Ultimately, we found that overfitting was a more serious problem than genetic stagnation, so our use of the plague was limited. Problems requiring more complex strategies may have more need for the genetic stagnation avoidance.

Our Implementation

There are two main components in any genetic programming search. These are the node types that make up the individual trees and the fitness function that determines how these trees are evaluated.

After describing our implementation of these two components, we focus on our resolution to the common pitfalls encountered by genetic programming such as overfitting the data and a lack of genetic diversity. By tailoring our GP search to the needs of this problem, we can produce more successful trading strategies than a more general purpose search would produce.

Tree Structure

Each tree consists of two node types: operator nodes and terminal nodes. Operator nodes take anywhere from two to four arguments and perform an operation on those arguments. Previous attempts at this problem used more complicated operator nodes to implement a combination of real valued and Boolean operators. We decided to use three mathematical operators: addition, subtraction, and multiplication, and two types of if constructs. We chose not to use a division operator because of the likelihood of generating numbers that are too far from the scale of our data. Each of the mathematical operators work as one would expect, accepting two arguments (which may be subtrees or terminal nodes) and performing the given operation on the values returned by these arguments. The first if construct, simply referred to as "IF", accepts three arguments. If the first argument returns a value greater than zero, "IF" returns the value of the second argument, otherwise it returns the value of the third argument. The other if construct, "IFGT" (if greater than), is similar to "IF" except that it takes four arguments. This node compares the values of its first two arguments. If the value of the first node is greater than that of the second, the value of the third node is returned, otherwise the value of the fourth is returned.

The other node type is the terminal node which makes up the bottom of each branch of the tree. These nodes access data from the technical indicators or return constants. Some previous attempts at this problem used only untransformed S&P500 prices as an indicator, a simplification that we believe may have contributed to their failure to construct successful trading strategies. In our experiments, we used not only

S&P500 prices but several high level technical indicators. The data used represents daily S&P500 closing prices during the period from April 4, 1983 to June 18, 2004.

Technical Indicators

One class of technical indicators that we used was the exponential moving average (ema). The ema is a weighted average of the S&P500 prices from the current day to the beginning of the data. The length parameter, k, controls how responsive to recent trends the ema is. At time i, the k day ema is

$$ema_i = ema_{i-1} + \left(\frac{2}{k+1}\right)(p_i - ema_{i-1}) \tag{1}$$

where p_i is the asset price at time i and ema_{i-1} is the previous day's ema with length parameter k.

The advantage exponential moving averages have over simple moving averages is they weight recent data more heavily than distant data. This makes the ema relatively more responsive to short term trends. We used exponential moving averages with lengths 20, 50, and 200, although these lengths are not necessarily optimal. A technical trader could compare a short term ema (20 day or so) to a long term ema (200 day or so) and buy when the short term ema crossed above the long term ema. Another technical indicator we used was the moving average convergence/divergence (macd). The macd is found by taking the difference of a 12 day and a 26 day ema of the closing prices. A macd trigger line is then formed using a 9 day ema of the macd line. A technical trader could use the macd components in a way similar to the ema. The macd indicates the general behavior of a market trend. If the macd and its trigger are diverging, a trend is accelerating, and the opposite is true if they are converging. At the beginning of our analysis, we hand formulated strategies based on the conventional usage of these indicators and compared them to a buy-and-hold strategy. We found that both ema and macd based strategies compare unfavorably with buy-and-hold over our data.

Another branch of technical indicators we used involved the number of advancing/declining issues (stocks) rather than the index closing price. The McClellan Oscillator is often thought of as a market momentum indicator and can be used to generate trading strategies. It is found by taking the difference of a 19 day and 39 day ema of the difference between advancing and declining issues. The usual associated trading strategy involves "overbought" (above +100) and "oversold" (below -100) ranges. When oscillator moves from oversold to positive, a buy signal is generated, and a sell signal is generated when the oscillator moves from overbought to negative. We also used the Arms Index, which uses not only issues but also volume data. However, this indicator did not prove useful in any of our experiments, so we omit further discussion except to mention that its lack of success may be due to a recent change in

the market from a quarter based pricing system to one based on decimals. The trees were also able to use lagged S&P data as well as lagged technical indicator data, but we found that lags of greater than one day were not generally useful. The lag feature allows the GP to generate approximate time derivatives by taking the difference of an indicator and its value for the previous day.

Since the technical indicators generate data that may be on highly incompatible scales, we decided to normalize all of our indicator data to the interval [-1,1]. In addition to the scaling problem, some indicators have a much greater time dependence than others. For example, S&P closing prices exhibit much higher volatility during recent years than during the first years in our sample. To compensate for this problem we converted our closing prices to a moving z-score by taking the difference of the current price and average price (over the past 200 days) and then dividing by the sample standard deviation over that period.

As is common in many genetic programs, we also used a terminal node that did not reference indicator data, returning instead a random constant. This constant was generated using the uniform probability distribution between -1 and 1 so as to be on the same scale as our normalized data.

Fitness Criterion

The other important component of a genetic program is a fitness function that measures the success of a solution strategy. Since the goal of our agents is to maximize the return on their investment, it is natural to use a fitness function that computes the return that a one dollar investment would generate using the tree's strategy. For our fitness function, each tree is evaluated at each time step. If the tree returns a value greater than 0, a buy signal is generated, otherwise a sell signal is generated. If the tree was "in the market" at the previous time step and a buy signal is generated no action is taken; similarly a sell signal has no effect if the tree is "out of the market." Each tree begins with a dollar at the beginning of the training period. When it sells, the initial investment is multiplied by the ratio of the sell price to the buy price. After the run through all the times, the original dollar is subtracted off so that the fitness most nearly corresponds to the trading profit. It should also be noted that "in the market" implies the investment of all the trader's money and "out of the market" implies a withdrawal of all the trader's money. There were no partial investments.

The addition of the sell penalty and the variation in the range over which we tested the trees prevented the fitness from reflecting the actual profits the strategy would yield, but our fitness is a monotonic transformation of that profit function. In other words, trees with higher fitness values earn greater returns than trees with lower fitness values.

Genetic programming algorithms are very effective at generating solutions that

maximize the fitness function. In our case, a solution is generated that buys and sells at very prudent times over the training data. Unfortunately, fitting past data is not necessarily indicative of having predictive power over future data. For this reason we reserved a portion of our data to serve as a validation set, so we could evaluate the performance of solutions against data that had not been used in training. Initially we trained over the first 4750 observations and then used the last 500 days as validation data. In order to correct for the time dependence in the behavior of this financial market, we also performed experiments using groups of data 100 observations long distributed evenly throughout the data as our validation data (each was separated from the next by 300 days of training data). In the end we returned to the 4750/500 strategy because it more closely mimics the way a real trader would learn.

If our hypothesis of trend predictability based on technical indicators is correct and the GP is behaving well, solutions that preform well over the training set should also generate returns that significantly outperform buy-and-hold strategies when evaluated over validation data. Unfortunately, genetic programming searches are susceptible to the problem of overfitting. This occurs when a tree matches the specifics of the training data extremely well but has no predictive power over unseen data. This situation is similar to the problem of using an interpolated polynomial to approximate a small data set. As the order of the polynomial increases, more data points are intersected, but the behavior of the polynomial between points and outside the data range becomes highly oscillatory and useless as a predictive tool.

As with polynomial interpolation, the main cause of overfitting is too many degrees of freedom. In order to mitigate the overfitting problem, we implemented several restrictions into our GP. We limited the number of levels each tree can have (excluding the return node level) to four. We also limited the total number of allowed nodes in a tree, usually to 20 or 24. Perhaps most importantly, we limited the number of times the GP bought and sold by including a fitness penalty proportional to the number of transactions the tree made over the training data. At first we tried to mimic the transaction costs faced by real world traders, but because of innovations in financial markets over the years, those costs were too small to make a noticeable difference in the resulting strategies. In order to restrict the freedom of the GP, we included a fairly substantial transaction penalty in the fitness function, although we did not penalize transactions while validating data because we use validation fitness to compare against buy-and-hold.

Finally, we restricted the number of generations the GP was allowed to complete to a few hundred because longer runs tended to increase training fitness without improving the validation fitness. In other words, the GP search found the fundamental trends relatively quickly, so long runs were not necessary.

Results

We ran a series of numerical experiments to determine which set of indicators and parameters was most effective at generating effective trading strategies. Any combination of parameters and indicators may generate strategies that obtain a high fitness over the training range, but we consider a setup effective if, on average, the solution trees that it finds outperform buy-and-hold strategies over validation data. It must be remembered that GP search is a highly stochastic process. Profit earned over the validation range can vary from negative to four times the buy-and-hold result. For this reason, optimization of the parameters is difficult at best.

We found several ingredients requisite to effective tree generation. The first is judicious choice of indicators. We were unable to find any high performing trees without the inclusion of the z-scored S&P500 data values, the McClellan indicator, and either several ema's of different lengths or the macd and its trigger line. Since we could not determine a significant difference in the efficacy of the macd and ema inclusions (or both, in fact), it appears that these indicators span essentially the same information. We tried many different lengths of ema without noticing a significant difference in the result. For the rest of the experiments we present here, we used the z-score of the S&P500 prices, the McClellan oscillator, and ema's of length 20, 50, and 200 or the macd and its trigger. All of this data was normalized to [-1,1] before performing the search. For most runs, we implemented a 0.2% penalty for every time the strategy sells.

Our experiments indicate that there was a large range of acceptable values for GP parameters. We typically used a tree depth of four, maximum number of nodes around 20 or 24, and maximum number of generations in the 300-500 range. Typical population size was 2,500 and 750 were killed off per generation. A large population size was key to search success; populations below about 500 were generally ineffective. In general, runs that utilized a training range broken into 300 day periods separated by 100 day validation ranges outperformed runs that divided the data into one contiguous training set and a single validation set. Despite this result, we chose to run most of our experiments using a single validation set in order to better simulate the problem a trader faces.

A noteworthy trading strategy from this set had a training fitness of 11.46 (compared to 5.31 for buy-and-hold) and a validation data fitness of 0.397 (compared to 0.1098). Strategies generated using this sell penalty traded less frequently and were more consistently profitable than those generated with no sell penalty was implemented. Figures 2 and 3 illustrate the tree and its trading strategy.

Hand picking trees that perform well over the validation range violates the principle of blind testing. To avoid this source of bias we ran the same problem 206 times and saved the training and validation finesses of each tree. By evaluating the average predictive values of these solutions, we test against the hypothesis that the GP

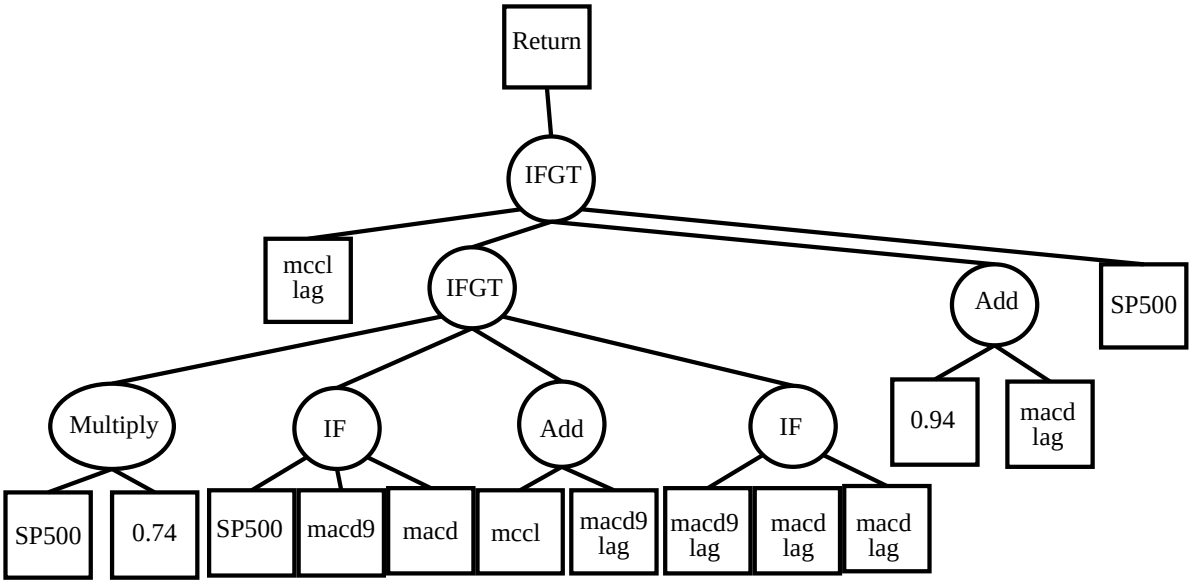


Figure 2. A tree that performs well over training and validation data

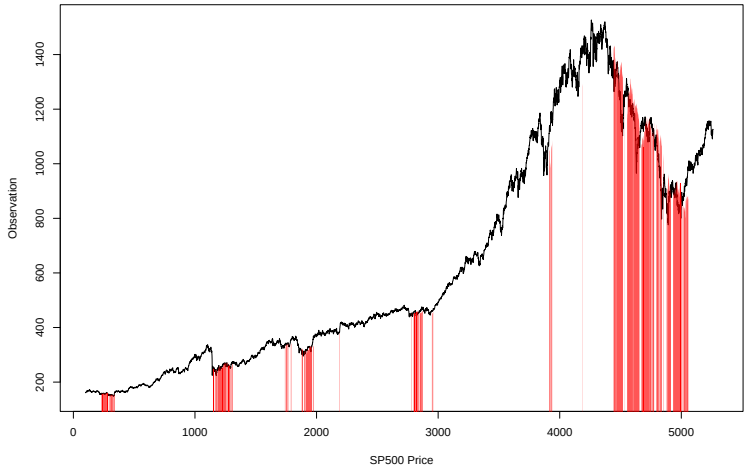
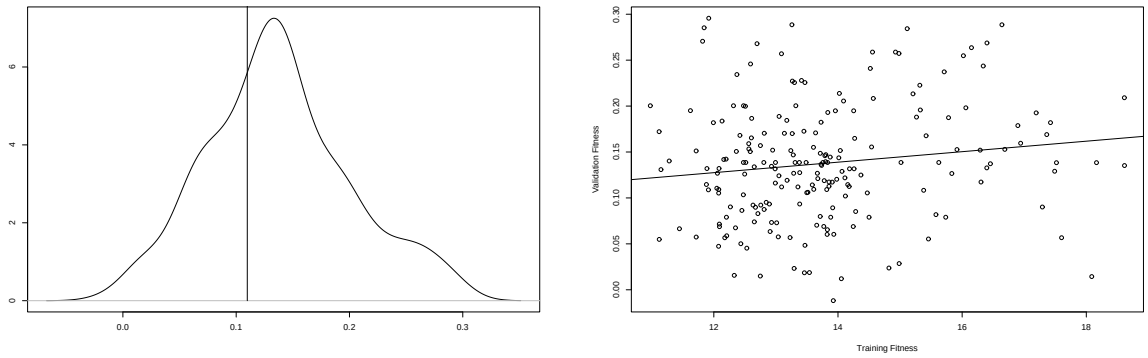


Figure 3. An effective strategy (shaded areas indicate out of the market)



(a) Empirical distribution of validation returns. The vertical line indicates the return to buy-and-hold

(b) Relationship between training and validation fitness

Figure 4. Results from a 206 independent run experiment

cannot generate trees that perform better than random strategies. The mean validation fitness of these trees is .13760, compared to .1098 for buy-and-hold. The sample standard deviation of these returns was 0.06339. Figure 4(a) shows the kernel density estimator (a numerical approximation of the probability distribution function) for these returns.

A correlation between the training fitness and validation performance of solution trees generated using the same parameter set would give reason to believe that the GP searches either did not converge or lacked sufficient diversity to find a global max. A least squares regression (see Figure 4(b)) on the results of the multi-run experiment shows a slight positive correlation between these two quantities. A hypothesis test for statistical significance of the slope parameter indicates that this positive correlation is statistically significant at the 5% level. This could be an indicator that an increased population size or an increase in the number of allowable generations (above 300).

Conclusion

Our numerical experiments provide strong evidence supporting a degree of technical predictability in the daily returns to holding S&P500 index funds. This has implications for the efficient market hypothesis, especially since the S&P500 is one of the most studied indices in the world.

Although the returns to following any particular solution strategy generated by

our GP search engine may not outperform a buy and hold strategy, on average they do. This means, for example, that a strategy of investing a percentage of one's funds in the S&P500 proportional to the number of solution strategies that recommend being in the market that day would likely outperform an index fund and potentially incur even less risk. It should also be remembered that in our simulations, the return to being "out of the market" is exactly zero. In real world applications, money that is withdrawn from an index fund generally earns an immediate return at the risk free rate. With this addition, our results would look even more impressive.

In short, we find that genetic programming can be used to identify predictable patterns in financial asset prices. This completes the first phase in the construction of technical traders. Now we can generate technical strategies based on simulated stock data from ASPEN-D and put these strategies to work in the code. We expect that technical trading agents will help the model simulate real world financial markets. The model can then be used to estimate the economic effects of shocks such as terrorist attacks and natural disasters. This information can be useful for developing strategies to mitigate these negative effects. Effective implementation may take a few iterations of generating and testing technical traders because their inclusion in the simulation may change the behavior of the simulated stock prices.

References

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

[2] Lee A. Becker and Mukund Seshadri. Comprehensibility and overfitting avoidance in genetic programming for technical trading rules. Technical report, Worcester Polytechnic Institute, May 2003.

[3] Lee A. Becker and Mukund Seshadri. GP-evolved technical trading rules can outperform buy and hold. In *Proceedings of the Sixth International Conference on Computational Intelligence and Natural Computing*, Embassy Suites Hotel and Conference Center, Cary, North Carolina USA, September 26-30 2003.

[4] John R. Koza. *Genetic programming: on the programming of computers by means of natural selection*. MIT Press, Cambridge, MA, 1996.

[5] Christopher J. Neely. Risk-adjusted, ex ante, optimal technical trading rules in equity markets. Working Papers 99-015D, Federal Reserve Bank of St. Louis, 2001. available at http://ideas.repec.org/p/fip/fedlwp/99-015.html.

DISTRIBUTION:

1 MS 0321
W. J. Camp , 9200

1 MS 0318
D. C. Barton , 9216

1 MS 0318
M. B. Boslough , 9216

1 MS 0196
B. N. Chenoweth , 9216

1 MS 0318
C. R. Jorgensen , 9216

1 MS 0196
M. D. Peters , 9216

1 MS 0318
D. A. Schoenwald , 9216

1 MS 0196
J. A. Sprigg , 9216

1 MS 0318
M. A. Taylor , 9216

1 MS 0318
R. M. Zalesak , 9216

1 MS 0310
P. Yarrington , 9230

1 MS 1110
D. Womble , 9214

10 MS 0318
G. V. Farnsworth , 9216

10 MS 0318
J. A. Kelly , 9216

10 MS 0318
A. S. Othling , 9216

10 MS 1109
R. J. Pryor , 9216

20 MS 0318
J. E. Nelson , 9216

1 MS 0318
S. G. Wagner , 9209

1 MS 0310
A. Slepoy , 9235

1 MS 0820
A. Farnsworth , 9232

1 MS 0771
J. E. Kelly , 6870

1 MS 0817
S. M. Kelly , 9224

1 MS 0819
D. I. Ketcheson , 9231

3 MS 9018
Central Technical Files, 8945-1

2 MS 0899
Technical Library, 9616

1 MS 0612
Review & Approval Desk, 9612