Besting Dollar Cost Averaging Using A Genetic Algorithm
A Master of Science Thesis Proposal
For Applied Physics and Computer Science

By James Maxlow
Christopher Newport University
October, 2003

Approved by:

Dr. David Hibler - Thesis Advisor _____

Dr. David Game _____

Dr. Antonio Siochi _____

Abstract

This project is designed to compare stock market investment strategies derived using genetic algorithms with the classic dollar-cost averaging investment strategy. The genetic algorithm created in this effort will use the price histories of sample stocks along with fictional investment capital and transaction costs to build strategies that produce high-value portfolios, or portfolios whose values exceed analogous dollar-cost values. These strategies will be based around the core concept of making fixed-amount investment decisions (buy, sell, or hold) at regular intervals primarily using the history of price percentage changes of the stock as the determining factor. In this manner, it can be determined if such strategies offer any advantage over fixed-amount buying at regular intervals. Once the strategies are derived by the genetic algorithm, their results on a series of test data sets will be statistically compared with dollar-cost averaging results on those same sets; both types will be compared to a baseline of the optimal results that could be achieved by investing all capital at the lowest price point of the given sample.

Introduction

Problem Statement

It has long been believed that it is a virtual impossibility, if not an *actual* impossibility, to predict the value of a given stock at some point in the future - it is of course the unreachable holy grail of any pursuit into the field of devising successful investment strategies. But as any investor that is successful over the long term can tell you, having *some* strategy, even though it cannot be perfect, is key in terms of *guiding* your decisions in ever-variable conditions. Some strategies are psychological, relying on the sweeping waves of market reaction and overreaction to financial and non-financial events; some rely only on the strength and operational status of the company compared to its competition; some are purely statistical, correlating all manner of obvious and arcane averages, trends, and data points; most rely on combinations of all of the above.

One strategy is something of a *non-strategy*, in a sense: dollar-cost averaging. Under this model, an investor spends *equal amounts of money* buying shares of a company at regular intervals *regardless of the price*. This ensures that over the long term the investor will buy *more* shares at lower prices and *less* shares at higher prices. It is therefore statistical in application, though it does not address the issue of *what* company in which to buy stock - that decision is left up to the investor. It may be thought of as a non-strategy because it completely ignores any factors external to the company *and* ignores any factors internal to the company - it is a completely independent, automatic process. However, it is popular because it allows investors to "buy low" even when they don't know what prices may be considered "low", once they have decided which company in which they wish to invest. It is likewise popular with fixed-income investors and those investors that intend to hold their portfolios for long periods of time.

But can we use computers to help us derive alternative strategies that still reflect the insular nature of dollar-cost averaging yet achieve better results? Can these strategies ignore external and internal information, relying only on the history of share prices, but still outperform the inherent "buy low" nature of dollar-cost averaging? This is the area this project will investigate.

Purpose of the Study

The purpose of this project is to use genetic algorithms to derive statistical investment strategies that can outperform the use of dollar-cost averaging for a variety of stocks. If it can be found that such strategies *do* produce better results, they can be put forth as a superior alternative for those that embrace automatic, hands-off investing approaches. The composition of each of these strategies will be the assignment of one of three actions - buy, sell, or hold - for any given price percentage change of a data set. For example, one derived solution may advise the buying of shares when the price rises 10%, but may advise selling of shares when the price rises 25%. This model does *not* attempt to predict future prices or future price changes; rather, it examines the *history* of percentage changes to determine what the best (or good) action may have been for every past percentage change - and in doing so, builds a case that repeating the same action for identical *current* percentage changes can yield positive results. Testing the validity of that

claim, and hence comparing the results to dollar-cost averaging, is the focus of this project.

Research Questions

Some fundamental questions to be answered via statistical analysis are:

- Does applying the derived strategies to test data sets lead to greater portfolio values than dollar-cost averaging over the same data?
- How does the underlying trend of the training data set used to derive a strategy affect the strategy's performance on the test data set?
- Will strategies derived using the same training data share common actions at most or all price changes, or will significant differences show up, producing multiple distinct but equally effective strategies?
- If multiple different strategies can be produced using a single training data set, what can be said about the probability of any single strategy producing positive results when used on a test data set?
- Will *hold* be an action suggested by the derived strategies due to the transactional cost of buying and selling, or will the genetic algorithm route around the hold action by finding more productive buy and sell actions in its exploration of possibilities?
- Will transactional costs play a significant role in the quality of the output produced by derived strategies and dollar-cost strategies on a given data set?

Review of Related Literature

Genetic Algorithms

Genetic algorithms were originally developed by John Holland in the 1960s, although his work was inspired by earlier evolutionary computing theories put forth in the previous decade [6: 4.] His work formed the basis for modern genetic algorithms, which can be thought of as auto-intelligent search schemes. The primary application of genetic algorithms are maximizing and minimizing problems, as well as problems with ill-defined search spaces. Traditional search algorithms systematically search through a large set of possible solutions, or rely on having an ordered possible solution set. In contrast, genetic algorithms can take an arbitrary search space whose parameters are only minimally defined and can *evolve* an indefinite number of families of possible solutions, from which an acceptable, even an optimal, solution can be chosen [6: 4, 5.]

How is this evolution enacted? The initial step in using a genetic algorithm is to design a *genome* that can represent a possible solution. This genome (or chromosome) can be a simple 1-D bitstring, or something as complex as an *n*-D array of arbitrary objects. A random population of these genomes is then created, from which promising solution candidates will eventually evolve. In order to facilitate evolution, instead of random breeding, a simulated natural selection process is codified in a *fitness* function and *selection* method [6: 9.] The fitness function is used to evaluate the relative strength of possible solution (genome.) The selection method determines which genomes will

survive and reproduce for the next generation and which will be eliminated from the genome pool. Such methods give preference to stronger genomes; many offer hedges against dominating genomes by also giving credit for diversity [6: 166.]

In all, the probability that a perfect (or high-quality) solution will be evolved is high because each generation of genomes can contribute its own individual strengths to the next generation.

EO and GAlib

Both the Evolving Objects Library and GAlib are intended to be complete C++ solutions for those wishing to use genetic algorithms to solve problems. The projects stretch from simple, pre-built genetic algorithm code whose parameters (crossover rate, mutation rate, etc.) can be modified for a user's needs, to fully-customizable systems where decisions on parameters can be set at run-time and custom genome representations and operators can be created with a minimum of code writing [1][3.]

At every level, steps have been taken in both libraries to provide users with code that simplifies their work and speeds their use of the frameworks - output procedures and file-based or command-line-based input are built in and ready to use through simple function calls. These provisions are written through extensive use of template-based object-oriented implementations, which is the secret to the immense customizations levels available to users [1][3.]

There are key differences between the libraries, however. GAlib has over a dozen genome representations pre-written in the code, ranging from simple bitstrings to 2-d and 3-d arrays of objects to trees and lists. It also has genetic operators appropriate for each genome type. In contrast, EO has only a few pre-written genomes, though it places no restrictions on creating your own [1][3.]

Another difference lies in the documentation for each library. GAlib is heavily documented with class inheritance hierarchies, explanation of required member functions for custom classes, and extensive detailing of how different genetic operators, such as crossover types, perform. EO, at its current release, does not have equivalent detailed documentation; many features are left up to the user to decipher. However, what EO does feature is a very clear set of tutorials to help users take advantage of the various components of the library. GAlib lacks equivalent walkthroughs [1][3.]

Dollar Cost Averaging

There are conflicting research reports concerning whether dollar cost averaging is a more or less productive strategy than lump-sum investing, value averaging, or asset allocation - primarily because there are so many market factors that affect the researched data sets that there is no way to establish a clearly strongest strategy for the majority of cases. Certainly it was a method pushed strongly by investment companies for their clients [4][7.] Regardless of such marketing, it has been shown that dollar cost averaging allows for the accumulation of shares at a lower average cost than the average price of the shares [2: 30.]

Despite that mathematical result, some researchers have concluded that dollar cost averaging is no more successful or risk-averse than random decisions on buys and sells [5: 87.] Likewise, the alternative model they suggest, value averaging, beats dollar cost averaging, under tested data sets, by an average of 3% on investment returns [5: 94.] In that sense, it may be irrelevant if dollar cost averaging is a great strategy (likely it is not); for this project, it only matters if a strategy can be derived that nearly always bests dollar cost averaging, and hence can take its place among other legitimate investment strategies.

It is surprising to note that there seems to be little, if any, published research in the area of trying to top dollar cost averaging via genetic computing. Many artificial intelligence-based investment simulations are complicated attempts to find predictive elements out of mounds of chaotic unrelated and inter-related data - these of course all boil down to attempts to time the market. While they may or may not have any hope of success, little attention seems to have been paid to finding non-predictive yet successful strategies.

Description of the Methodology / Timeline

The initial phase of the project will be centered around information gathering in two areas. First, I will decide whether to use a pre-existing genetic algorithm library - GAlib or EO - or to write my own simple genetic algorithm implementation. The former has the advantage of saving time and allows me to rely on either of two highly robust, battle-tested libraries. Given that decision, I would review the associated help files, tutorials, and other documentation related to each so that I could choose the library that best fit my admittedly simple needs.

As an alternative, I may decide to write my own custom genetic algorithm implementation. In doing so, I could optimize the code for my personal application, leaving out the features of the pre-written libraries that I would never use. It may also make it easier for me to create a custom chromosome model which I will need. Lastly, it may provide an option to achieve better performance in using my particular algorithm. However, it seems likely that I will opt to work from one of the given libraries, so that I can spend most of the project time coding the fitness logic, putting the genetic algorithm to work, and analyzing the results. Writing my own is simply an option I am leaving myself should it become necessary.

The second phase of the information gathering will be relatively short - I will need to decide what stocks I will use to feed the genetic algorithm and to use as test data sets. These choices should reflect a variety of price histories… those that show a clearly increasing trend, those that show a clearly decreasing trend, and those that show no discernible trend. In addition, the variation of trends should be accompanied by a variation of data point distances from the trend lines, i.e., data sets whose points follow the overall trend lines closely and data sets whose points show wild and frequent deviation from the overall trend lines. Data sets stretching back a variety of years will also be selected. I can then easily retrieve their price histories through any number of

financial websites. Accounting for the magnitude of these prices, I will choose a fixed transaction dollar value for all of my simulations, in addition to deciding on a transaction fee (as a percent.) While these choices generally bear no importance in and of themselves since they will be arbitrary, it will be necessary to ensure that they are the same across all of my simulations - genetic algorithm strategies, dollar-cost averaging strategies, and the baseline model - so that the results can be legitimately compared. Furthermore, I may be investigating the influence of the transaction fee on the results, meaning I will likely vary it under certain circumstances.

The next phase of the project will be to design the chromosome and its fitness model, and then to implement them both in code. At this point, given some simple decisions as to rudimentary input and output coding, the genetic algorithm implementation will be ready for testing with a simple data set. This testing will include the variation of parameters related to crossover rate, mutation rate, population size, and selection type. The goal will be to select the parameters that push the sample derived solutions as close as possible to the sample baseline results after a reasonable number of tests. The baseline result is of course the optimum solution for any set, and so I will strive to force the genetic algorithm to come very close to this.

Assuming no problems there, I will begin using the genetic algorithm to derive strategies for all of the test sets. This will involve multiple runs for any given stock that all use a fixed number of past data points. In this manner I will produce multiple derived strategies that can be compared to each other to determine if the process produces identical or dissimilar strategies across runs.

Once the strategies from all of those test runs have been recorded, they will be applied to the same stocks using data points they have not seen - that is, the strategies will be put to work on new data sets to determine fictional portfolio values.

At that point one third of the result-set will be completed. The next task will be to write a simple function that applies a dollar-cost-averaging strategy to a given data set. This function will use the same transaction value and fee as the genetic algorithm to buy shares at every price point interval. The program will then generate portfolio value results using same data points to which the genetic algorithm was applied. Thus the second third of the result-set will be completed.

The third component of the result set will simply be a series of mathematical calculations of the optimum portfolio results; these will be found by buying shares with all available capital at the lowest price in the test set, and saving all capital afterwards. This mimics the ability of a person to guess the very lowest price a stock will ever see and buying as much as possible at that point - so while it is unrealistic from an investing sense, it will provide the ceiling portfolio values to which both the computer-derived and dollar-cost strategies can be compared.

The last phase of the project will likely be the most time-consuming. It will consist of a series of statistical comparisons between the three strategy types, and of statistical

comparisons within the set of genetic algorithm strategies to determine the consistency of using a genetic algorithm to generate result-equivalent strategies. The goal of this analysis is of course to answer the questions posed earlier, and to see if any overriding conclusions emerge by observing the qualities of the derived strategies. The analysis may support the notion of using derived strategies as alternatives to dollar-cost averaging… or it may show them to be ineffective by comparison… but it should decide the issue either way.

Proposed Timeline:
Evaluation of GAlib / EO for project suitability - 3 days
Stock and index choices, gathering of price histories, calculation of optimal cases - 2 days
Design and implementation of chromosome and fitness model - 1.5 weeks
Testing, tweaking, and refactoring code; testing with parameter variation - 2 days
Using genetic algorithm to derive strategies, applying strategies to new data sets - 2 days
Dollar-cost averaging program coding and use on data sets - 3 days
Statistical Analysis - 2 weeks
Report - 3 weeks
Total - 9 weeks

References

[1] EO documentation: https://sourceforge.net/projects/eodev/

[2] Edleson, M. E. Value Averaging: The Safe and Easy Investment Strategy. Chicago: International Publishing Corporation, 1991.

[3] GAlib documentation: http://lancet.mit.edu/ga/

[4] Liscio, J. Portfolio Discipline: The Rewards of Dollar Cost Averaging. *Barron's*, Aug. 8, 1988, pp. 57-58.

[5] Marshall, Paul S. A Statistical Comparison of Value Averaging vs. Dollar Cost Averaging and Random Investing Techniques. Journal of Financial and Strategic Decisions: Vol. 13 No. 1, Spring 2000.

[6] Mitchell, Melanie. An Introduction to Genetic Algorithms. Cambridge: The MIT Press, 2002.

[7] The Vanguard Group of Investment Companies. The Dollar Cost Averaging Advantage. Valley Forge: Brochure #0888-5, BDCA, 1988.