Analysis of Online Algorithm for Resource Allocation
Applied to the Stock Market

by

Alexander Michael St. Clair Duran

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degrees of

Master of Engineering in Electrical Engineering and Computer Science

and

Bachelor of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 1999

©Alexander Michael St. Clair Duran, MCMXCIX. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly
paper and electronic copies of this thesis document in whole or in part, and to grant
others the right to do so.

Author................................................................................................................
Department of Electrical Engineering and Computer Science
February 9, 1999

Certified by ...............................................................................................
Tom Leighton
Professor of Applied Mathematics
Thesis Supervisor

Accepted by ...............................................................................................
Arthur C. Smith
Chairman, Department Committee on Graduate Theses
Analysis of Online Algorithm for Resource Allocation Applied to the Stock Market

by

Alexander Michael St. Clair Duran

Submitted to the Department of Electrical Engineering and Computer Science on February 9, 1999, in partial fulfillment of the requirements for the degrees of Master of Engineering in Electrical Engineering and Computer Science and Bachelor of Science in Computer Science and Engineering

Abstract

In this thesis, I modified and implemented an algorithm which attempts to find and predict areas of good performance in a larger system where performance is random or adversarial. The algorithm was modified to work on a real stock market problem instead of an abstract problem. To test the algorithm, closing prices for every stock in the current S&P 500 were used, dating back to January 2, 1990.

Thesis Supervisor: Tom Leighton
Title: Professor of Applied Mathematics
Acknowledgments

I would first like to thank my thesis advisor Prof. Tom Leighton for helping me out with this thesis. Without him, this thesis wouldn't have been written. Besides providing the algorithm without which there would be no thesis to write, he helped me deal with problems that arose during the course of this research. I would also like to thank my parents, brothers, and friends who provided me with much needed moral support. Their words of encouragement and friendly derisive chiding helped me to get this done. To some of my friends I owe special thanks. Without them this thesis would have been even later and considerably more over budget. My thanks go to Charles Hope, Jennifer Cunningham, Rebecca Merz, and Geoff Reber for proofreading and offering me helpful suggestions. Without them I don't know what I would have done. Finally, I want to thank the teachers and staff of Course VI and the Institute in general. I really enjoyed my time here and they made that enjoyment possible.
## Contents

1 Introduction .......................................................... 8
   1.1 Options .................................................................. 8
      1.1.1 The Black-Scholes Model ............................ 10
   1.2 Technical Analysis ............................................. 11

2 The New Algorithm ................................................... 12
   2.1 Abstract Stock Problem ................................. 12
   2.2 The Algorithm .................................................. 13

3 Modification to Reality ........................................... 16
   3.1 Failure of the Model ...................................... 16
      3.1.1 Price Changes .................................. 17
      3.1.2 Losing Money .................................. 19
      3.1.3 Investment After the Sale ................... 19
      3.1.4 Affecting Prices .............................. 19
   3.2 Storing the Money ........................................... 20

4 Results ................................................................. 21
   4.1 Available Data ................................................. 21
   4.2 Unrealistic Assumptions ............................... 22
   4.3 Variance .......................................................... 23
   4.4 Control Group ................................................... 24
   4.5 Alternate Investment .................................. 24
   4.6 Measuring Performance .............................. 25
   4.7 Calculating $D$ ................................................. 26
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4-1</td>
<td>Expected return on a single investment under the continuous transaction assumption when $D$ was known and the goal was to make money</td>
<td>30</td>
</tr>
<tr>
<td>4-2</td>
<td>Expected return on algorithm stock selection minus expected return on random stock selection under the continuous model when $D$ was known and the goal was to make money</td>
<td>32</td>
</tr>
<tr>
<td>4-3</td>
<td>Expected return on a single investment under the once-a-day transaction assumption when $D$ was known and the goal was to make money</td>
<td>33</td>
</tr>
<tr>
<td>4-4</td>
<td>Expected return on a single investment under the continuous transaction assumption when $D$ was known and the goal was to lose money</td>
<td>37</td>
</tr>
<tr>
<td>4-5</td>
<td>Expected return on algorithm stock selection minus expected return on random stock selection under the continuous model when $D$ was known and the goal was to lose money</td>
<td>38</td>
</tr>
<tr>
<td>4-6</td>
<td>Expected return on a single investment under the once-a-day transaction assumption when $D$ was known and the goal was to lose money</td>
<td>42</td>
</tr>
</tbody>
</table>
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>An example of the abstract stock problem</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>Another example of the abstract stock problem. The adversary defeats a deterministic strategy</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>A third example of the abstract stock problem. The adversary competes against a random strategy</td>
<td>14</td>
</tr>
<tr>
<td>4.1</td>
<td>Results of attempting to make money under the continuous transaction assumption</td>
<td>29</td>
</tr>
<tr>
<td>4.2</td>
<td>Results of attempting to make money under the continuous assumption and random stock selection</td>
<td>31</td>
</tr>
<tr>
<td>4.3</td>
<td>Results of attempting to make money under the once-a-day transaction assumption</td>
<td>34</td>
</tr>
<tr>
<td>4.4</td>
<td>Results of attempting to lose money under the continuous transaction assumption</td>
<td>36</td>
</tr>
<tr>
<td>4.5</td>
<td>Results of attempting to lose money under the continuous assumption and random stock selection</td>
<td>39</td>
</tr>
<tr>
<td>4.6</td>
<td>Results of attempting to lose money under the once-a-day transaction assumption</td>
<td>41</td>
</tr>
<tr>
<td>4.7</td>
<td>Results of running the algorithm on previously untested data starting January 1, 1996.</td>
<td>44</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The problem of finding a way to reliably make money in the stock market is one that many people have attempted to solve. Stockbrokers and portfolio managers who have good track records are in high demand. A computer algorithm that is successful at picking stocks would be a very useful asset to any investor. Unfortunately, since the stock market often seems to act much like a random walk, developing such an algorithm is not an easy task. This paper analyzes one attempt at such an algorithm.

Chapter one describes some of the previous solutions to stock market prediction. Chapter two describes an abstract model of the stock market and a proposed algorithm for making effective choices on when to buy and sell stocks. Chapter three describes the modification of the real stock problem to sufficiently resemble the abstract problem so that the algorithm can be applied. Chapter four describes the results of experimentation. Chapter five discusses the conclusions to be drawn from the experiments and offers suggestions for further applications of the algorithm to the stock problem.

Investors in the past have tried numerous approaches to developing a formula for stock prediction. Unfortunately, a reliable and effective system for predicting price changes has thus far not been developed. Some methods have been developed, with mixed results. Two of these methods are options pricing and technical analysis.

1.1 Options

One way in which investors attempt to profit from correctly predicting changes in stock prices is through buying and selling options. Options guarantee the buyer either a fixed
price at which the stock can be bought or a fixed price at which the stock can be sold. There are two types of options: call options and put options.

The writer of a call option guarantees to the buyer that on the expiration date of the option, the stock for which the option was written can be bought by the buyer at the strike price of the option. The strike price is set when the option is first written, and generally matches the price of the stock at the time of writing. If, on the expiration date of the option, the market price of the stock is higher than the strike price on the option, the buyer can make money by buying the stock at the strike price and then selling it at the market price. In practice, the option buyer just sells the option back and doesn’t bother to actually exercise it. If the market price is below the strike price, however, the option is worthless.

The writer of the option makes a prediction that, on the date that the option expires, the market price of the stock will be less than the strike price of the option plus the option price, or premium. If the market price is lower than the strike price, the writer keeps the entire premium. If the market price is between the strike price and the strike price plus premium, then the buyer can recoup some of the premium by selling the option, but the writer still makes a profit. If the market price is higher than the strike price plus premium, then the buyer gets more from selling the option than he spent on the premium, and the writer loses money when forced to sell the stock to the buyer for significantly less than the market value of the stock.

A put option is the opposite of a call option. The writer of a put option guarantees that the stock can be sold for the strike price. If the market price at the expiration date is lower than the strike price, then the buyer can buy the stock at the market price and then exercise the option by selling the stock at the strike price. Again, in practice, the option is sold instead of being exercised. The writer of the option predicts that the market price will be higher than the strike price. If his prediction is correct, or nearly so, he keeps some or all of the premium.

As a writer or buyer of options, it is important to know the fair value of the option. The fair value of an option is the value at which both the writer and the buyer are neutral about trading the option. This value can, in theory, be computed by taking the expected return of the option for the buyer and subtracting a small amount for risk adjustment. For example, suppose that the buyer already has an option on stock A. The expected sale value of that option on the expiration date (a value ranging from $0 upwards) is x. After subtracting y
to compensate for the risk inherent in options, the fair value of the option is $x - y$. If this were the premium, the buyer would be neutral about buying the option[3, pp. 165-168].

The main problem in determining the fair value of an option lies in calculating the expected return. Calculating the expected return is similar to predicting what the stock price will do. It is not quite as specific, as it involves defining a probability function for the stock price, whereas specific prediction attempts to nail down exactly what the price will do, but it still requires guessing (or knowing) what the stock price is likely to do. Therefore, options pricing is tied to the problem of predicting stock prices.

1.1.1 The Black-Scholes Model

A commonly used technique for calculating the fair value of an option is the Black-Scholes model, developed by Fischer Black and Myron Scholes[3, pp. 195-199]. The Black-Scholes model simplifies the problem of determining fair option value, but makes some assumptions that adversely affect the accuracy of the model. These assumptions attempt to quantify how stock prices behave. If the stock market doesn't behave in this manner, then the validity of the model is thrown into question. Three of the more important assumptions are as follows:

1. The stock price follows a random walk in continuous time with a variance rate proportional to the square of the stock price.

2. The distribution of possible stock prices at the end of any finite interval is lognormal.

3. The variance rate of return on the stock is constant.

The model is commonly used to price options, and it appears to approximate fair value, since investors both buy and sell options. If it significantly over or undervalues options, then investors would stop buying or writing options, respectively. It’s assumptions, however, make it less useful than it would otherwise appear. Stock prices changes actually tend to deviate significantly from a lognormal distribution. Some of the other, less important, assumptions that the model makes are also incorrect[3, p.197-198]. The net result is that it tends to yield option values that are too low. In order for a model to be really effective, it needs to avoid incorrect assumptions about the market.
1.2 Technical Analysis

Technical analysis techniques attempt to guess general trends in a particular stock or index. They do not attempt to predict specific prices, but whether the market is bullish (showing an upward trend) or bearish (showing a downward trend), or due for a rally or correction. Technical analysis techniques generally look at the average over the past $n$ days of either a stock price or some derived indicator. When the indicator crosses this average, it is an indication of a stock price or market change[1].

Indicators generally attempt to track one of three things. Some try to predict what investors think of the market; since prices are based on investor actions, and these actions are based on expectations, indicators based on expectation can yield reliable results. Other indicators try to track external factors, such as interest rates, which affect stock prices. Finally, some indicators track the current momentum of the market, in the belief that trends will continue. Technical analysis is used as a technique to improve performance slightly, thus giving investors a small edge which hopefully can yield noticeable returns. Although it can give an investor a small edge in the market, it is not as effective as one might wish.

Although technical analysis is a widely used method, it is hard to quantify its effectiveness. There are a large number of such techniques in existence. If any one of them worked significantly better than the others, it would be in widespread use. Since this is not the case, there is no obvious single method to test, and testing any one of them would be unlikely to yield significant returns.
Chapter 2

The New Algorithm

2.1 Abstract Stock Problem

Suppose that there are $n$ commodities in which one can invest. On each of the next $A$ days, each of the commodities will pay either $1$ or nothing. At the beginning of every day, the results from previous days can be examined, but no information is given about future results. At the beginning of one of the days, one of these commodities should be purchased. The amount earned is the number of $1$ payoffs remaining in the $A$ days. If no commodity is purchased, no money is earned. There is no reward for not purchasing anything.

As an example, consider Table 2.1. Over a period of fifteen days each commodity pays either $1$ or nothing on each day. The buyer only buys one commodity ever. If the buyer chooses to buy commodity three at the beginning of day ten, he or she receives a total of $4$. Although commodity three had a total of $6$ in payoffs, the payoffs on days two and nine were not received because the commodity was not yet owned.

Success in this scenario can be measured as a ratio of the amount of money $m$ that the purchaser receives over the maximum amount $D$ that the purchaser could have received. $D$ is simply the maximum number of payoffs of any one commodity, and cannot exceed $A$. In Table 2.1, $D$ is equal to $11$, and $m$ is equal to $4$. The buyer achieved a ratio of $4/11$.

Success is made more difficult by the presence of an adversary. Suppose that there is an adversary who knows what strategy will be used for picking commodities, and sets up the payoffs in advance so as to minimize the ratio $m/D$. If the buyer uses a deterministic strategy, then the adversary can set up the payoffs such that whatever commodity the purchaser picks has no payoffs after the time that the purchaser picks it. Therefore, the purchaser
Table 2.1: An example of the abstract stock problem

<table>
<thead>
<tr>
<th>Commodity</th>
<th>Day 1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

makes $m = 0$ dollars, while $D$ dollars were possible. This is clearly a very poor result for the buyer.

As an example, suppose the buyer looked at the results from Table 2.1 and wanted to try to pick the best commodity. Looking at the distinguishing features of commodity one, the buyer might decide either to always buy commodity one, or to buy the first commodity that had three payoffs. Either of these strategies would purchase commodity one in the previous example. One would have a payoff of $11$, the other would have a payoff of $8$. If the adversary knows that the buyer is using either of these strategies, then he or she can set up the payoffs to look like Table 2.2. In this case, both of those deterministic strategies fail to result in any payoffs although one of the commodities pays $14$.

To avoid some of the problems that an adversary poses, a randomized strategy must be used. The most obvious choice is to pick a random commodity and purchase it on the first day. To minimize the success of the buyer, the adversary sets up the payoffs such that 1 of the $n$ stocks has payoff $D$ and the other stocks have payoff $0$. Then the buyer has an expected return of $\frac{D}{n}$. An example of such a table is Table 2.3. The buyer has an expected ratio of $\frac{1}{10}$ from optimal.

### 2.2 The Algorithm

A recently developed algorithm[2] provides a much better solution in some cases. The algorithm provides a way for achieving a profit $d$. The algorithm assumes that one of
<table>
<thead>
<tr>
<th>Commodity</th>
<th>Day 1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>8</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td>1</td>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2: Another example of the abstract stock problem. The adversary defeats a deterministic strategy.

<table>
<thead>
<tr>
<th>Commodity</th>
<th>Day 1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
<th>12</th>
<th>13</th>
<th>14</th>
<th>15</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>7</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>8</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>10</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.3: A third example of the abstract stock problem. The adversary competes against a random strategy.
the $n$ commodities has a total payoff of $D \geq 3d \log n$. In this case, $d$ will be made with probability at least $1 - O(\frac{d \log n}{D})$.

In order to select the commodity to purchase, first label the commodities as $C_1, C_2, \ldots C_n$. At the beginning of each day, for each commodity $C_i$ such that $C_i$ had a payoff the previous day, flip a coin that produces heads with probability $p$. If the coin comes up heads, purchase that commodity and cease flipping. The probability $p$ takes into account $x$, the number of payoffs that commodity $C_i$ has had in the past. Then $p$ is defined by the following formula:

$$p = \frac{1}{d} \cdot n^{((3x/D)-2)}$$

If $D$ is unknown, then there are several cases to consider. If it is essential that $d$ be made (nothing less will do), then one can guess that $D = 3d \log n$ and run the algorithm anyway. If the buyer just wants to maximize earnings from any one purchase, then the days can be partitioned as follows. The first partition lasts until some commodity has paid off $3 \log n$ times. During this time, the algorithm is run with $d = 1$. After this interval ends, the next one runs until some commodity has paid off $2 \cdot 3 \log n$ times since the beginning of the second interval. The algorithm here is run with $d = 2$. In general, the $i$-th interval ends when, since the beginning of the interval, some commodity has paid off $2^{i-1} \cdot 3 \log n$ times, and $d = 2^{i-1}$ during this time.
Chapter 3

Modification to Reality

3.1 Failure of the Model

The abstract stock problem is a far cry from the real market. Stock prices do not change by small, discrete amounts. In the abstract problem, there is either a $1 payoff or there isn't. In addition, the payoffs occur, effectively, at the end of the day. In the real market, the amount by which a stock increases varies, and its price can fluctuate throughout the day. Stocks are also worth varying amounts. When a stock valued at $100 gains a point, it is far less significant than when a stock valued at $1 gains a point.

In the abstract problem, there is a lower limit on failure. The worst case is that there are no payoffs after the day of purchase. In the real stock problem, there is no lower limit. The stock price can fall arbitrarily far, and the value of an investment can get very close to zero. An incorrect decision is not just opportunity lost, it reduces the ability to make money in the future.

In addition, in the abstract problem, there is no reason to sell the commodity. In the real stock market, once the target has been achieved, it is important to sell. If the investor doesn't sell the stock, then the price can drop suddenly and the investor will end up losing money. It is also important to determine what the investor does after selling the stock. Waiting until the end of the time period to run the algorithm again is a waste of investment opportunities.

Finally, in the abstract problem, all the payoffs are set in advance. Purchasing a stock does not affect the future payoff schedule. The purpose of a randomized algorithm is that the adversary cannot change things based on the results of the coin flips that the algorithm
performs. In the real market, extremely large purchases can affect stock prices.

3.1.1 Price Changes

Price fluctuations make the problem of the real stock market a very different one from that of the abstract stock market. For this problem, it is assumed that it is the value of the stocks purchased that is set, rather than the number of stocks purchased. If the algorithm reveals that a $10 stock should be purchased, and 100 shares are bought, then a purchase of a $5 stock would result in 200 shares being bought.

To allow changes in stock price to accurately reflect the change in value to the person using the algorithm, a gain in stock price is measured in half-percent increases. For instance, if the price of a stock rises from $5 to $6, that is a 20% increase. Since about 36 half-percent increases are needed for one 20% increase, the 20% increase is considered to be 36 payoffs or coin flips. Under this system, if two stocks both show ten payoffs, then the return on investment for purchasing either of those stocks would be the same. If the stock price increases by less than a half-percent, or drops, then that is equivalent to having no payoff in the abstract stock problem.

Stock prices fluctuate. They rise and fall on a daily basis. If every half-percent increase were considered a payoff, then those stocks which fluctuated rapidly between two values would be valued highly, since they would show many payoffs. Although valued highly, a stock which follows this wave-like price history would probably not be a good investment. If the algorithm bought it too near the top of the wave, then it might never reach its target return.

To combat this problem, the only half-percent increases that the algorithm considers are those that lead to new highs for the stock. If the previous highest value for a stock was $20, and it hits $21, then there are about ten payoffs. If the stock later drops back to $20 and then climbs to $21, there are no additional payoffs. If it climbs to $22, there are approximately nine more payoffs. In this way, stocks whose prices do nothing but alternate between two values are ignored, while those stocks which show steady gains are not.

Gains in low-valued stocks tend to represent a larger percentage of the initial price than gains in high-valued stocks. Although the algorithm considers only new highs when looking for payoffs, if the stock price takes a plunge and then recovers, it is important to let the algorithm attempt to catch that upswing. In order to provide the algorithm with that
chance, every time a stock reaches a new low price, a new stock is added to the list. That stock has no previous high price, so if it climbs above its initial value it will hit a new high.

This new stock functions almost exactly like an original stock. Its presence contributes to the count of the number of stocks being used and therefore affects payoff flips. If this new stock hits a new low, however, it doesn’t create another new stock, since, if this new stock hits a new low, then the stock that created it also hits that same new low and creates a new stock. This allows the algorithm a chance to take advantage of long recovery upswings in an attempt to make money.

For example, suppose stock $A$ hits a new low price at $10$, falling from a previous high of $12$. A second stock, $A_1$, is then created. If on the next day, stock $A$ rises to $11$, then the algorithm flips coins for stock $A_1$, because it has hit a new high from its old high of $10$, but not for stock $A$, which had a previous high of $12$. If the stock then climbs further to $13$, both $A$ and $A_1$ receive coin flips. Finally, if the stock then falls to $9$, a new stock, $A_2$, is created because $A$ has hit a new low. Although $A_1$ has also hit a new low, it does not create a new stock. The creation of a new stock happened when $A$ hit its new low.

The algorithm attempts to find, with high probability, a sequence of payoffs at least as long as the targeted sequence of payoffs. Since it makes no guarantees when looking for payoffs larger than the target, it is important to accurately track payoffs as seen by the algorithm. In order to track them accurately, payoffs are counted starting immediately after the algorithm decides to buy the stock. If it is assumed that stock can be bought and sold at any time, then the investor realizes all of the gains that the algorithm expects. On the other hand, if the investor can only buy or sell stock at the beginning of the day, then the algorithm may perceive gains which the investor does not actually receive.

For example, suppose that the stock price rises $5.2\%$ in one day. This results in ten coin flips. Suppose that the fourth flip comes up heads and the algorithm believes that the stock should be bought. Since the stock cannot be bought until the beginning of the next day, the remaining six half-percent increases are not made that day. In order for the algorithm to function properly, however, it must assume that they were made. If the target is twelve payoffs and the stock has eight more the following day, then the algorithm believes that all twelve half-percent increases were achieved, and sells. The investor, however, only receives the eight payoffs from the second day, not the twelve that the algorithm saw. In some cases, the entire target can be achieved in a single day, and the algorithm may issue a buy and
a sell instruction both on the same day, but the investor will never realize any return and won’t bother making the transaction.

3.1.2 Losing Money

The problem of losing money is serious. The algorithm tends to purchase volatile stocks. When the goal return is not met, it is likely that a significant fraction of the investment can be lost. Since the algorithm tries to catch upswings in stock prices, if the stock price drops while the algorithm is holding the stock, it is quite possible that the stock is never going to recover enough for the target to be hit.

In order to get rid of stocks whose prices are declining, a cutoff can be imposed. If, since purchase, the stock shows a net loss greater than some fraction of the targeted gain, then the stock should be sold. Of course, this means that some stocks which might have recovered and realized the desired gain will now be sold too early.

3.1.3 Investment After the Sale

After the target is achieved and the stock is sold, the investor must decide what to do next with the money. If the algorithm has a useful expected return, then it should be used again. Suppose the investor runs the algorithm over a period of six months. After two months, the target profit is achieved. If the investor waits until the end of the remaining four months, those four months are wasted as far as investment opportunities are concerned. Instead, the investor should start the algorithm over on a new six month time period. If the algorithm typically finishes buying and selling by the end of the second month, then six worthwhile investments can be made in a year instead of only two. This is clearly beneficial for the investor.

3.1.4 Affecting Prices

The problem of affecting prices by purchasing stocks is only an issue when very large sums of money are involved. Since changing the price would prevent analysis with this algorithm, it was assumed that all transactions involved insufficient amounts of money to affect the price. For almost any personal user, this is a completely valid assumption.
3.2 Storing the Money

The algorithm gives instructions on when to buy a stock and when to sell it. Often, the time between the buy date and the sell date is short, so the money necessary to own the stock is only tied up in the stock for short periods of time. An important consideration is the disposition of the funds when they are not tied up in owning the stock. If safety is desired, the money can be put in a savings account or similar risk-free short term investment plan.

A large problem with this strategy arises when the stock market is doing well. If the stock market is making high annual returns, then, although over any short time period it has low returns, over the entire year an investment will pay quite well. Although the algorithm might outperform the stock market during the time in which the algorithm actually owns the stock it picks, it spends so much of the year without any investments that it will not perform as well annually as the market. The algorithm may give fantastic returns over a period of two days, but getting those returns for two days out of six months is a smaller return than getting the stock market average return for the entire six months.

This problem can be handled by investing the money in a fund that grows at approximately the rate of the stock market, such as an index fund. Since, in theory, the algorithm gets large returns over a short period of time, taking the algorithm’s suggestions for buying and selling will increase performance above that of the index fund. Even if the stock market is performing poorly, it is still wise to have the money invested in the best alternative investment, provided the investor can enter and leave the other investment on short notice and at low cost.
Chapter 4

Results

4.1 Available Data

In an effort to have a wide range of data available for running simulations, data were obtained from January 1st, 1990, to September 18, 1998. The data consist of opening, closing, high and low prices as well as the volume traded for all of the stocks in the current S&P 500. Some of the stocks in the current S&P 500 were not trading as early as January 1st, 1990. The data for these stocks start later.

The prices were adjusted for splits and dividends. When running the algorithm, only the closing price was considered. For the purposes of testing the algorithm and deciding upon optimal parameters, only time periods starting before January 1st, 1995 were considered. After testing was completed, those configurations which performed the best on the test data were run on the untested data from 1996 onwards. This was to prevent problems with fitting the algorithm too closely to the data.

Between January 1st, 1990 and September 18, 1998, stocks in the S&P 500 fared reasonably well. Although good for investors, there were insufficient data for testing the algorithm on lean years or crashes. In order to make an estimate of how well the algorithm would fare during a crash, the program was also run with the goal of losing money.

Three other factors were adjusted. The algorithm was tested both under the assumption that it could only buy and sell at the start of the day, and under the assumption that it could buy and sell whenever it wanted. In order to minimize damage from stocks that didn’t perform well, the algorithm was tested such that it would sell a losing stock if the stock lost a certain fraction of the amount that the program was trying to make. The algorithm was
also tested without this cutoff. Finally, the algorithm was tested in the case that it knew the maximum payoff \( D \), and in the case that it did not know the value of \( D \).

### 4.2 Unrealistic Assumptions

Under the transaction model in which the investor can buy or sell stocks only at the beginning of the day, it was assumed that the closing price from the day before would be equal to the opening price the next day. In reality, this is not always the case; however, the prices are generally close enough that this assumption should not pose a significant problem.

A bigger problem arises under the continuous transaction model. In this model, the investor can buy or sell stock at any time. Unfortunately, data that fine were not available. Instead, it was assumed that stock price changes were spread out in half-percent changes evenly over the course of the day. If the stock price rose or fell from open to close, it was assumed that it changed evenly in this range. This assumption, however, is not as dangerous as it might seem. The algorithm only cares about prices that achieve new highs or lows. This helps eliminate problems caused by price fluctuations during the day.

If the stock price hit a new high or low during the day but that price was not reflected in the close, then the simulations did not take it into account. This does result in some inappropriate smoothing of the data. It is believed that this effect is small and will not significantly affect performance.

The continuous model also has another problem. All of the coin flips for a single stock during a single day were performed together. This simplified calculating immensely. In the real world, all of the stock prices would be monitored simultaneously, and a stock that is bought in the simulation might not be purchased in the real world because another stock might have been selected earlier that day. For example, if stock \( x \) would have been picked in the mid-afternoon, and stock \( y \) would have been picked in the morning of the same day, then in the real world, \( y \) would be picked every time. In the simulation, the stock that is purchased later in the day might be run first, and the earlier stock would not be checked. To stick with the example, if, in the simulation, \( x \) were checked first, it would be picked instead of \( y \). This is not a serious problem. It is unlikely that two different stocks would be likely to be picked on the same day.

Another issue that was ignored is the cost of transactions. Buying and selling stock gen-
erally involves a transaction cost or commission. There are Internet trading sites available that charge a flat fee per trade, and for sufficiently large investments this fee is relatively small. The main effect shows up when considering a one month time period versus a longer time period. During each time period in which a stock is selected, at least four transactions occur. The money is withdrawn from its previous investment, the stock is bought, the stock is sold, and the money is reinvested in its previous investment.

If the investor makes one stock selection every month, the cost of these four transactions is paid twelve times during the year. If the cost of any particular transaction were $20, then the cost of all forty-eight transactions throughout the year would be close to $1000. Unless the amount of money being invested is fairly large, this represents a significant cost. If the selections are only made every six months, then the cost is only about $160. If the amount invested at the beginning of the year were $10000, then a return of about 1% per time period on the six month investment would keep the money level constant. In order to keep the amount of money constant in the case where purchases are made every month, a return of 1% would also be needed. It is much more difficult to get a return of 1% in one month than it is in six months. When determining the optimum investment strategy for the algorithm, the cost of transactions should be considered. For the purposes of the simulation, however, the transaction cost was assumed to be $0.

4.3 Variance

Despite attempts to limit it, there can be a fair amount of variance in results from the algorithm. In an effort to reduce this variance, for each length of time, 100 time periods were randomly selected. The periods were selected by picking the starting date uniformly at random from the available start dates. For each time period, the algorithm was run 50 times.

In order to further reduce variance, every configuration for a given length of time was run on the same set of time periods. For example, all of the tests on one month time periods were run on the same randomly selected sets of one month periods. In addition, when looking at the effects of buying only at the beginning of the day versus buying at any time, the same purchase choice was always made in both cases. The returns on the once-a-day version are the same as the results if the algorithm selected the stock at the
same time as in the continuous model, but the purchase was made only at the beginning of
the next day. Finally, all cutoff levels for the same configuration made the same purchase
choice. For instance, when running the algorithm once on a four month time period in
which the optimum result is known, after the algorithm purchased a stock, the results of
having no cutoff and the results of having a cutoff equal to the target were both checked. In
other words, the effects of the choice of transaction model and cutoff level were generated
based on the same purchase choice for each run of the algorithm and targeted return.

The variance on the once-a-day transaction model is higher than that in the continuous
model. In the continuous system, whenever the algorithm correctly picks a stock, the
amount of money it makes is set, since it sells precisely at the time that its target is achieved.
The only variance here arises from whether or not the stocks are actually picked correctly.
Under the assumption that the stocks can only be bought and sold at the beginning of the
day, both the correctness of the pick and the timing of the pick affect the payoff, and so the
variance is higher.

4.4 Control Group

When the algorithm is successful, it might have been the case that it was only successful
because a large portion of the stocks were changing price in the correct direction. In this
case, picking any random stock at the same time would have an expected return similar to
that of the algorithm. In order to verify that the algorithm was being effective, the program
was run again with one small change. Instead of purchasing the stock the algorithm selected,
a random stock would be purchased. The purchase would occur at the time the algorithm
suggested for purchase, but the actual stock selection would be random.

4.5 Alternate Investment

One issue to consider is the overall return on investment. If this algorithm generates a
return of 12% annually, but picking an index fund yields a 20% annual return, it isn’t clear
from that information alone what the best strategy is. Also, the algorithm will often hold
stocks for only a very short period of time. The time during which it does not hold stocks
can be used to invest the money in other ventures. When using this algorithm, it seems
best to invest in some alternate investment that is easy to buy and sell, and withdraw funds
from that investment only during the time when they are needed to follow the advice of the algorithm.

This increases the return over that of just the investment only if the algorithm does better than the investment while the investor owns the stock picked by the algorithm. In order to test this, it was assumed that there was an alternate investment available with constant interest. When the program tried to make or lose money, the alternate investment guaranteed some specified annual return. When the program tried to make money, the alternate investment was set at 30% annual return. The return on investment for investing in a purely random stock for the entire length of time during the five years tested was approximately 17% annually. 30% was chosen as a safe upper bound. Any result the algorithm could achieve when compared to 30% would be even better when used against a lesser alternative. When attempting to lose money, the return was set at -20% annually.

4.6 Measuring Performance

Performance was measured in several different ways. Straight performance for a single run of the algorithm was computed by taking the percent change in stock price from the time the algorithm actually purchased the stock to the time the algorithm actually sold the stock. In the continuous model, these prices always exactly reflected the prices the algorithm saw. The percent change for a success was always equal to the targeted return. In the system in which the program could only purchase at the beginning of the day, the prices do not necessarily reflect those the algorithm saw. If the algorithm requested a purchase or sale midday, the actual price at which the stock was purchased or sold was the price at the close of that day.

To compute performance for a single purchase and sale for an entire group of runs for a particular configuration of the program, the arithmetic mean of the returns for single purchases and sales was taken. For example, suppose the algorithm was run twice for a particular configuration and the results were a 5% loss and a 10% gain over the time period. Then the average performance was 2.5%.

Annual performance for the algorithm alone was computed by determining what the investor would make annually if he or she immediately started the algorithm over again after the time period expired or the algorithm requested a sale-whichever came first. This
was calculated by using successive approximation. The annual performance was first guessed to be a 0% return. For each run of the algorithm, the result of running the algorithm and investing in an investment with an annual return equal to the guess was calculated. The arithmetic mean of all of these results was calculated. If this result was different from the guess by more than a very small factor (less than one-tenth of one percent), then the result was used as the next guess.

When computing performance when combined with an alternate investment, the same method was used as when computing annual performance, but during all times in which the stock chosen by the algorithm was not owned, the money was in the alternate investment.

4.7 Calculating $D$

In the ideal case, the algorithm knows the best possible return for the time period in which it is running. In the real world, this value will not be known and must be guessed. The best way to come up with an acceptable guess is to look at past performance. The values of $D$ for a large number of previous time periods of the same length could be determined. The algorithm could then guess that the best return in the current time period might be equal to the mean, median, maximum, or minimum result from the previous time periods. Since the algorithm only makes guarantees about performance when $D$ is sufficiently large relative to the targeted investment, guessing a value for $D$ that is too large will greatly reduce performance. To avoid this problem, the two guesses tested were the mean and smallest guesses.

The least result from a set of previous time periods of the same length would be unlikely to be larger than the best result from the current time period. Guessing that the best possible return would be equal to the smallest previous result is a conservative guess; guessing that the best possible return would be equal to the mean is more daring. The returns are higher, but it is likely that the guess will be higher than the actual best possible return. Both of these guesses were tested in simulation.

4.8 Ways in Which the Algorithm was Run

The algorithm was run in a variety of different situations. On some tests it was set to find stocks that increased in price, on others it was set to find stocks that decreased in price.
The algorithm's ability to accurately determine the value of $D$ (the best possible return in the time period) was altered for different simulations, varying from the case in which the algorithm knows the best return, to when it guesses the least or mean of the best returns from all other time periods of the same length. Simulations were run on time periods of three different lengths, one month, four months, and seven months. Cutoff levels were also varied, ranging from no cutoff to cutting off as soon as the price change went in the wrong direction by only a quarter of the targeted amount. Finally, some simulations picked a random stock instead of the stock suggested by the algorithm.

4.9 Price Increases

In recent years, the stock market has been doing fairly well. Stocks tend to go up rather than down and, as a result, it's easier to profit by buying stocks than selling them short. It make sense, then, to use the algorithm to predict price increases. As expected, the algorithm performed better under the continuous transaction model than it did under the system in which it could only purchase stocks at the beginning of the day. In the continuous transaction model, incorporating cutoffs increased the expected return on a single investment from the algorithm, the annual return, and the return compared to an alternate investment of 30% annual interest. When $D$ was known and there were no cutoffs, the algorithm achieved its targeted return over 90% of the time. When cutoffs were included, this declined, sometimes by as much as 20%. Although cutoffs decreased the success rate, the expected return on investment rose. In the once-a-day model, cutoffs generally decreased the success rate as well as reducing returns.

4.9.1 The Continuous Model

Under the continuous transaction model, the algorithm always made a profitable trade, and almost always performed better when combined with the alternate investment of 30% than the alternate investment alone. The expected return from a single investment varied between 0.36% to 2.71%. The longer time periods had better expected returns on a single investment than the shorter time periods. The percentage of purchases which were successes did not differ substantially between the results from various length periods.

Although the algorithm had better expected returns on a single investment when run on
larger time periods, the annual returns and returns combined with the alternate investment were higher for smaller time periods. The algorithm can be run more times per year on the shorter time periods, so although the returns on a single investment were smaller, the combined return of all the runs in a given year was larger.

Incorporating cutoffs tended to increase the expected return on a single investment as well as the annual and combined yields. The tighter cutoff levels yielded better returns than the more lax cutoff levels. Including cutoffs did cause a drop in the success rate. The tight cutoff level of .25, at which the algorithm got rid of the stock as soon as it lost a quarter of what it had been trying to make, generally had a success rate 20% lower than the algorithm did without cutoffs. The more lax cutoff levels had successes more often, but even the cutoff at twice the target return had about 10% lower success rates than the case without cutoffs.

The expected returns from running the algorithm once under the assumptions that stocks could be bought or sold at any time and that \( D \) was known are shown in Figure 4-1. It can clearly be seen that the algorithm yielded better returns when run on larger time periods and with tighter cutoffs. The complete results of running the algorithm under the continuous assumption and various guesses for \( D \) can be seen in Table 4.1.

**Guessing \( D \)**

When the optimum return, \( D \), was guessed, results were mixed. When \( D \) was guessed to be the least of the results from other time periods, success rates rose slightly relative to the results when \( D \) was known. When \( D \) was guessed to be the mean of the results from other time periods, there were fewer successes. When the algorithm was run on longer time periods, the success rate for the mean guess was closer to that of the least guess and the case when \( D \) was known.

The expected return from a single investment fell when \( D \) was guessed. The mean guess almost always had a higher return than the least guess. Although the returns from a single investment were low when \( D \) was guessed to be the least of previous results, the annual return and the return combined with an alternate investment were higher. The least guess always outperformed the mean guess and almost always outperformed the known guess. When tight cutoffs were included, the least guess did outperform the known guess in every case. The mean guess always had lower annual returns and returns combined with the 30%
<table>
<thead>
<tr>
<th>Months</th>
<th>Cutoff</th>
<th>$D$</th>
<th>Success %</th>
<th>Mean % Return</th>
<th>% Annual Return</th>
<th>% Return with 30%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0</td>
<td></td>
<td>Known</td>
<td>92.44</td>
<td>0.75</td>
<td>21.74</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>94.36</td>
<td>0.36</td>
<td>20.27</td>
<td>35.37</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>72.48</td>
<td>0.11</td>
<td>1.53</td>
<td>25.86</td>
</tr>
<tr>
<td></td>
<td>.25</td>
<td></td>
<td>Known</td>
<td>82.74</td>
<td>1.29</td>
<td>48.57</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>84.46</td>
<td>0.65</td>
<td>66.61</td>
<td>98.34</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>66.10</td>
<td>1.06</td>
<td>22.86</td>
<td>55.68</td>
</tr>
<tr>
<td></td>
<td>.5</td>
<td></td>
<td>Known</td>
<td>83.12</td>
<td>1.24</td>
<td>46.09</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>84.64</td>
<td>0.62</td>
<td>62.83</td>
<td>93.56</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>66.42</td>
<td>0.99</td>
<td>20.96</td>
<td>53.21</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>Known</td>
<td>83.80</td>
<td>1.14</td>
<td>41.56</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>85.40</td>
<td>0.58</td>
<td>55.62</td>
<td>84.69</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>67.04</td>
<td>0.85</td>
<td>17.59</td>
<td>48.72</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>Known</td>
<td>86.02</td>
<td>1.02</td>
<td>35.47</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>87.18</td>
<td>0.50</td>
<td>44.91</td>
<td>71.25</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>68.60</td>
<td>0.64</td>
<td>12.75</td>
<td>42.18</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td>Known</td>
<td>92.76</td>
<td>1.35</td>
<td>9.29</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>95.44</td>
<td>0.68</td>
<td>11.99</td>
<td>31.48</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>85.10</td>
<td>0.61</td>
<td>3.12</td>
<td>29.74</td>
</tr>
<tr>
<td></td>
<td>.25</td>
<td></td>
<td>Known</td>
<td>72.34</td>
<td>2.15</td>
<td>18.12</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>79.60</td>
<td>1.16</td>
<td>32.56</td>
<td>66.86</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>63.98</td>
<td>1.78</td>
<td>10.57</td>
<td>42.21</td>
</tr>
<tr>
<td></td>
<td>.5</td>
<td></td>
<td>Known</td>
<td>73.48</td>
<td>2.02</td>
<td>16.83</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>80.08</td>
<td>1.09</td>
<td>30.28</td>
<td>63.86</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>65.72</td>
<td>1.62</td>
<td>9.33</td>
<td>40.80</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>Known</td>
<td>75.36</td>
<td>1.79</td>
<td>14.65</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>80.90</td>
<td>0.97</td>
<td>26.13</td>
<td>58.40</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>69.50</td>
<td>1.40</td>
<td>8.16</td>
<td>38.81</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>Known</td>
<td>81.66</td>
<td>1.63</td>
<td>13.00</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>82.72</td>
<td>0.76</td>
<td>19.66</td>
<td>49.84</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>77.48</td>
<td>1.39</td>
<td>7.93</td>
<td>37.99</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td></td>
<td>Known</td>
<td>95.24</td>
<td>2.42</td>
<td>9.85</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>96.08</td>
<td>0.95</td>
<td>7.68</td>
<td>29.14</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>87.82</td>
<td>1.95</td>
<td>6.60</td>
<td>34.58</td>
</tr>
<tr>
<td></td>
<td>.25</td>
<td></td>
<td>Known</td>
<td>70.40</td>
<td>2.71</td>
<td>12.56</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>73.80</td>
<td>1.54</td>
<td>17.92</td>
<td>50.65</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>66.98</td>
<td>2.53</td>
<td>9.51</td>
<td>41.28</td>
</tr>
<tr>
<td></td>
<td>.5</td>
<td></td>
<td>Known</td>
<td>72.64</td>
<td>2.57</td>
<td>11.86</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>74.74</td>
<td>1.42</td>
<td>16.41</td>
<td>48.62</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>69.34</td>
<td>2.39</td>
<td>8.92</td>
<td>40.41</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td></td>
<td>Known</td>
<td>76.24</td>
<td>2.36</td>
<td>10.72</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>76.16</td>
<td>1.21</td>
<td>13.64</td>
<td>44.88</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>72.94</td>
<td>2.16</td>
<td>7.96</td>
<td>38.98</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td></td>
<td>Known</td>
<td>83.02</td>
<td>2.26</td>
<td>10.09</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td></td>
<td>81.12</td>
<td>1.02</td>
<td>11.14</td>
<td>41.95</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td></td>
<td>78.36</td>
<td>1.95</td>
<td>7.07</td>
<td>37.39</td>
</tr>
</tbody>
</table>

Table 4.1: Results of attempting to make money under the continuous transaction assumption
Results of the Control Group

In order to verify the effectiveness of the algorithm, the algorithm was run with random stock selection. When the purchased stock was selected randomly, the number of successes dropped significantly. Although success levels were low, the random stock selection had a positive expected return in most cases. The return when combined with an alternate investment, however, rarely exceeded the return of the alternate investment alone, and only at the tightest cutoff level. In all cases, the returns from the random stock selection were smaller than those from the algorithm’s choice.

The difference in the expected return on a single investment under the continuous model between the algorithm’s choice and the random stock selection can be seen in Figure 4-2. In every case, the algorithm’s choice outperformed the random stock. The full results of random stock selection can be seen in Table 4.2.
<table>
<thead>
<tr>
<th>Months</th>
<th>Cutoff</th>
<th>$D$</th>
<th>Success %</th>
<th>Mean % Return</th>
<th>% Annual Return</th>
<th>% Return with 30%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Known</td>
<td>67.44</td>
<td>-0.15</td>
<td>-2.31</td>
<td>10.33</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>80.08</td>
<td>-0.53</td>
<td>-12.98</td>
<td>-8.92</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>46.88</td>
<td>-0.26</td>
<td>-3.36</td>
<td>14.34</td>
<td></td>
</tr>
<tr>
<td>.25</td>
<td>Known</td>
<td>38.40</td>
<td>0.36</td>
<td>9.37</td>
<td>32.66</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>39.42</td>
<td>0.20</td>
<td>11.78</td>
<td>28.79</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>28.88</td>
<td>0.30</td>
<td>5.21</td>
<td>30.81</td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td>Known</td>
<td>43.62</td>
<td>0.26</td>
<td>6.33</td>
<td>27.87</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>44.24</td>
<td>0.13</td>
<td>6.85</td>
<td>22.19</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>33.20</td>
<td>0.21</td>
<td>3.35</td>
<td>27.78</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Known</td>
<td>51.40</td>
<td>0.15</td>
<td>3.07</td>
<td>22.28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>51.12</td>
<td>0.02</td>
<td>0.02</td>
<td>12.70</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>38.46</td>
<td>0.07</td>
<td>0.82</td>
<td>23.37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>58.74</td>
<td>-0.02</td>
<td>-0.02</td>
<td>15.83</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>60.32</td>
<td>-0.14</td>
<td>-5.72</td>
<td>2.25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>43.36</td>
<td>-0.08</td>
<td>-0.82</td>
<td>18.98</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>76.12</td>
<td>0.51</td>
<td>2.46</td>
<td>19.79</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>89.68</td>
<td>0.21</td>
<td>1.88</td>
<td>11.44</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>67.16</td>
<td>0.41</td>
<td>1.67</td>
<td>22.55</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>32.96</td>
<td>0.51</td>
<td>3.60</td>
<td>30.26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>36.40</td>
<td>0.32</td>
<td>6.85</td>
<td>31.13</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>32.20</td>
<td>0.56</td>
<td>2.97</td>
<td>30.75</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Known</td>
<td>41.06</td>
<td>0.42</td>
<td>2.78</td>
<td>27.88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>42.52</td>
<td>0.22</td>
<td>4.26</td>
<td>26.98</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>40.26</td>
<td>0.48</td>
<td>2.44</td>
<td>29.15</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>52.08</td>
<td>0.34</td>
<td>2.09</td>
<td>25.18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>51.74</td>
<td>0.07</td>
<td>0.84</td>
<td>20.99</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>50.06</td>
<td>0.37</td>
<td>1.76</td>
<td>26.88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>64.52</td>
<td>0.36</td>
<td>2.04</td>
<td>22.88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>64.32</td>
<td>-0.08</td>
<td>-1.04</td>
<td>15.17</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>60.70</td>
<td>0.40</td>
<td>1.79</td>
<td>25.07</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>79.72</td>
<td>1.17</td>
<td>3.45</td>
<td>22.52</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>91.02</td>
<td>0.65</td>
<td>3.39</td>
<td>16.74</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>72.40</td>
<td>0.53</td>
<td>1.38</td>
<td>22.01</td>
<td></td>
</tr>
<tr>
<td>.25</td>
<td>Known</td>
<td>33.16</td>
<td>0.71</td>
<td>2.91</td>
<td>30.54</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>32.86</td>
<td>0.36</td>
<td>3.54</td>
<td>30.25</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>32.94</td>
<td>0.75</td>
<td>2.55</td>
<td>30.80</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Known</td>
<td>42.92</td>
<td>0.71</td>
<td>2.81</td>
<td>29.20</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>39.26</td>
<td>0.20</td>
<td>1.81</td>
<td>27.08</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>42.16</td>
<td>0.70</td>
<td>2.30</td>
<td>29.43</td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td>Known</td>
<td>55.48</td>
<td>0.76</td>
<td>2.81</td>
<td>27.47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>50.02</td>
<td>0.01</td>
<td>0.01</td>
<td>22.96</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>54.06</td>
<td>0.67</td>
<td>2.09</td>
<td>27.67</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Known</td>
<td>68.90</td>
<td>1.01</td>
<td>3.46</td>
<td>26.11</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>66.28</td>
<td>0.01</td>
<td>0.01</td>
<td>20.31</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>64.78</td>
<td>0.65</td>
<td>1.90</td>
<td>25.63</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.2: Results of attempting to make money under the continuous assumption and random stock selection
4.9.2 The Once-a-day Model

When running under the assumption that the algorithm could only buy and sell stock at the beginning of the day, the algorithm did not always make a profitable trade. Although its success rates were the same as those of the continuous model, the fact that it could only buy and sell stock at the beginning of the day prevented it from realizing the returns that the continuous model received. Often, even when it did make a profitable trade, the trade was insufficiently profitable for the algorithm combined with the alternate investment to outperform the alternate investment.

Much like in the continuous system, the algorithm had a higher expected return on a single investment in longer time periods. Again, successes in shorter time periods did much better than those in longer time periods when comparing annual returns and returns combined with the alternate investment. Cutoffs were almost completely detrimental. Incorporating cutoffs always reduced the expected return on a single investment. Generally, the tighter cutoffs had lower expected returns than the lax cutoffs. Although the cutoffs reduced the expected return on a single investment, they did increase the annual and
Figure 4-3: Expected return on a single investment under the once-a-day transaction assumption when $D$ was known and the goal was to make money combined returns slightly.

The expected returns from running the algorithm once under the assumptions that stocks could be bought or sold only at the beginning of the day and that $D$ was known are shown in Figure 4-3. It can be seen that larger time periods and looser cutoffs had better returns. The complete results of running the algorithm under the once-a-day model and with various guesses for $D$ can be seen in Table 4.3.

Guessing $D$

When the optimum return, $D$, was guessed, results varied widely. When $D$ was guessed to be the mean of the results from other time periods and time periods were short, it consistently outperformed both the cases when $D$ was known and $D$ was guessed to be the least result from other time periods. The mean guess outperformed the other two cases in expected return from a single investment as well as annual return and combined return. For longer time periods, the reverse is true. The least guess generally outperformed the cases where $D$ was known and where $D$ was the mean guess. The returns from the least guess in long time periods, however, were not as high as those of the mean guess in short time
<table>
<thead>
<tr>
<th>Months</th>
<th>Cutoff</th>
<th>$D$</th>
<th>Success %</th>
<th>Mean % Return</th>
<th>% Annual Return</th>
<th>% Return with 30%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Known</td>
<td>92.44</td>
<td>0.17</td>
<td>3.91</td>
<td>26.30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>94.36</td>
<td>0.22</td>
<td>11.51</td>
<td>25.82</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>72.48</td>
<td>1.83</td>
<td>37.91</td>
<td>70.88</td>
<td></td>
</tr>
<tr>
<td>.25</td>
<td>Known</td>
<td>82.74</td>
<td>-0.18</td>
<td>-4.78</td>
<td>17.55</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>84.46</td>
<td>-0.10</td>
<td>-6.27</td>
<td>7.32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>66.10</td>
<td>1.69</td>
<td>38.65</td>
<td>75.58</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Known</td>
<td>93.12</td>
<td>-0.18</td>
<td>-4.84</td>
<td>17.37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>84.64</td>
<td>-0.10</td>
<td>-5.88</td>
<td>7.73</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>66.42</td>
<td>1.68</td>
<td>38.35</td>
<td>75.15</td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td>Known</td>
<td>83.80</td>
<td>-0.20</td>
<td>-5.24</td>
<td>16.86</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>85.40</td>
<td>-0.11</td>
<td>-6.80</td>
<td>6.36</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>67.04</td>
<td>1.65</td>
<td>37.45</td>
<td>73.80</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Known</td>
<td>86.02</td>
<td>-0.19</td>
<td>-4.84</td>
<td>16.92</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>87.18</td>
<td>-0.09</td>
<td>-5.28</td>
<td>7.88</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>68.60</td>
<td>1.64</td>
<td>36.54</td>
<td>72.12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Known</td>
<td>92.76</td>
<td>0.75</td>
<td>4.98</td>
<td>30.08</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>96.44</td>
<td>0.88</td>
<td>15.75</td>
<td>35.86</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>85.10</td>
<td>0.50</td>
<td>2.52</td>
<td>28.95</td>
<td></td>
</tr>
<tr>
<td>.25</td>
<td>Known</td>
<td>72.34</td>
<td>-0.17</td>
<td>-1.17</td>
<td>26.45</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>79.60</td>
<td>0.06</td>
<td>0.80</td>
<td>27.02</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>63.98</td>
<td>-0.29</td>
<td>-1.54</td>
<td>26.38</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Known</td>
<td>73.48</td>
<td>-0.20</td>
<td>-1.37</td>
<td>26.12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>80.08</td>
<td>0.03</td>
<td>0.02</td>
<td>25.89</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>65.72</td>
<td>-0.26</td>
<td>-1.36</td>
<td>26.56</td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td>Known</td>
<td>75.36</td>
<td>-0.28</td>
<td>-2.01</td>
<td>25.12</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>80.90</td>
<td>-0.03</td>
<td>-0.08</td>
<td>24.08</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>69.50</td>
<td>-0.14</td>
<td>-0.69</td>
<td>27.18</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Known</td>
<td>81.66</td>
<td>-0.09</td>
<td>-0.54</td>
<td>26.47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>82.72</td>
<td>-0.07</td>
<td>-1.13</td>
<td>22.42</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>77.48</td>
<td>0.19</td>
<td>0.98</td>
<td>29.12</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Known</td>
<td>95.24</td>
<td>1.54</td>
<td>6.18</td>
<td>33.21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>96.08</td>
<td>1.31</td>
<td>10.82</td>
<td>32.84</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>87.82</td>
<td>0.93</td>
<td>3.07</td>
<td>30.10</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Known</td>
<td>70.40</td>
<td>0.19</td>
<td>0.78</td>
<td>29.89</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>73.80</td>
<td>0.36</td>
<td>3.80</td>
<td>32.61</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>66.98</td>
<td>-0.07</td>
<td>-0.21</td>
<td>28.62</td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td>Known</td>
<td>72.64</td>
<td>0.20</td>
<td>0.83</td>
<td>29.82</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>74.74</td>
<td>0.34</td>
<td>3.46</td>
<td>32.09</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>69.34</td>
<td>-0.07</td>
<td>-0.20</td>
<td>28.55</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Known</td>
<td>76.24</td>
<td>0.24</td>
<td>0.96</td>
<td>29.81</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>76.16</td>
<td>0.24</td>
<td>2.34</td>
<td>30.47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>72.94</td>
<td>-0.10</td>
<td>-0.30</td>
<td>28.20</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Known</td>
<td>83.02</td>
<td>0.54</td>
<td>2.50</td>
<td>31.05</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>81.12</td>
<td>0.34</td>
<td>3.40</td>
<td>31.21</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>78.36</td>
<td>0.06</td>
<td>0.16</td>
<td>28.52</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.3: Results of attempting to make money under the once-a-day transaction assumption
4.10 Price Decreases

In order to simulate a crash, or a lean year, the algorithm was run with the goal of losing money. As in the case where the algorithm was trying to make money, the algorithm lost money more successfully when operating under the continuous transaction model than it did under the once-a-day model. Cutoffs decreased the success rate but enhanced performance under the continuous model. In the once-a-day model, cutoffs were generally unsuccessful and caused returns to rise.

4.10.1 The Continuous Model

When operating under the continuous transaction system, the algorithm almost always picked a stock whose price declined. It also usually made a trade that, when combined with the alternate investment of -20%, outperformed the alternate investment alone. When run on longer time periods, the algorithm had lower (and therefore, better) expected returns on a single investment than it did when run on shorter time periods. The algorithm, however, generally yielded better results on annual and combined returns when run on shorter time periods. Success rates were similar between all lengths of time periods.

Incorporating cutoffs aided performance significantly. The algorithm generally had better results for a single investment as well as for annual and combined returns. Much as in the case where the algorithm was trying to make money, the tighter cutoffs were more effective than the looser cutoffs. Success rates fell when cutoffs were incorporated, dropping between 5% and 20%.

The expected returns when trying to lose money and running the algorithm once under the continuous assumption and the assumption that $D$ was known are shown in Figure 4-4. The larger time periods and tighter cutoffs clearly had better returns. The complete results of running the algorithm under the continuous model are shown in Table 4.4.
<table>
<thead>
<tr>
<th>Months</th>
<th>Cutoff</th>
<th>$D$</th>
<th>Success %</th>
<th>Mean % Return</th>
<th>% Annual Return</th>
<th>% Return with 30%</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Known</td>
<td>94.14</td>
<td>-0.49</td>
<td>-12.26</td>
<td>-25.74</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>94.14</td>
<td>0.17</td>
<td>7.64</td>
<td>-0.54</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>82.82</td>
<td>-0.11</td>
<td>-1.81</td>
<td>-17.40</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Known</td>
<td>85.38</td>
<td>-1.29</td>
<td>-33.51</td>
<td>-45.08</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>84.68</td>
<td>-0.62</td>
<td>-37.14</td>
<td>-46.18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>74.68</td>
<td>-1.04</td>
<td>-22.47</td>
<td>-36.39</td>
<td></td>
</tr>
<tr>
<td>.25</td>
<td>Known</td>
<td>85.94</td>
<td>-1.25</td>
<td>-32.65</td>
<td>-44.38</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>84.72</td>
<td>-0.59</td>
<td>-35.71</td>
<td>-44.94</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>75.56</td>
<td>-1.00</td>
<td>-21.53</td>
<td>-35.58</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Known</td>
<td>87.24</td>
<td>-1.20</td>
<td>-31.38</td>
<td>-43.28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>84.94</td>
<td>-0.54</td>
<td>-32.82</td>
<td>-42.33</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>76.74</td>
<td>-0.90</td>
<td>-19.57</td>
<td>-33.93</td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td>Known</td>
<td>89.10</td>
<td>-1.11</td>
<td>-29.09</td>
<td>-41.26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>85.54</td>
<td>-0.44</td>
<td>-27.02</td>
<td>-37.32</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>78.92</td>
<td>-0.76</td>
<td>-16.59</td>
<td>-31.33</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Known</td>
<td>94.76</td>
<td>-0.56</td>
<td>-4.12</td>
<td>-19.37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>94.52</td>
<td>0.30</td>
<td>3.98</td>
<td>-8.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>87.34</td>
<td>-0.04</td>
<td>-0.09</td>
<td>-15.63</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Known</td>
<td>79.56</td>
<td>-1.97</td>
<td>-16.80</td>
<td>-32.69</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>80.52</td>
<td>-1.22</td>
<td>-22.82</td>
<td>-36.81</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>69.50</td>
<td>-1.67</td>
<td>-12.15</td>
<td>-28.99</td>
<td></td>
</tr>
<tr>
<td>.25</td>
<td>Known</td>
<td>80.22</td>
<td>-1.86</td>
<td>-18.58</td>
<td>-35.43</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>80.80</td>
<td>-1.15</td>
<td>-21.59</td>
<td>-35.78</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>70.22</td>
<td>-1.51</td>
<td>-10.99</td>
<td>-28.03</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Known</td>
<td>81.98</td>
<td>-1.69</td>
<td>-14.48</td>
<td>-30.74</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>82.16</td>
<td>-1.04</td>
<td>-19.53</td>
<td>-33.03</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>71.84</td>
<td>-1.22</td>
<td>-8.88</td>
<td>-26.25</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Known</td>
<td>85.38</td>
<td>-1.50</td>
<td>-12.78</td>
<td>-30.28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>84.36</td>
<td>-0.85</td>
<td>-16.03</td>
<td>-39.98</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>74.76</td>
<td>-0.77</td>
<td>-5.56</td>
<td>-23.65</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Known</td>
<td>93.22</td>
<td>3.73</td>
<td>16.86</td>
<td>-0.29</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>95.88</td>
<td>0.73</td>
<td>6.59</td>
<td>-4.77</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>88.84</td>
<td>3.37</td>
<td>13.94</td>
<td>-2.97</td>
<td></td>
</tr>
<tr>
<td>.25</td>
<td>Known</td>
<td>72.04</td>
<td>-2.17</td>
<td>-11.54</td>
<td>-26.67</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>77.18</td>
<td>-1.36</td>
<td>-18.90</td>
<td>-33.87</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>70.32</td>
<td>-2.05</td>
<td>-9.97</td>
<td>-27.45</td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td>Known</td>
<td>73.78</td>
<td>-2.01</td>
<td>-16.76</td>
<td>-28.00</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>78.04</td>
<td>-1.28</td>
<td>-17.77</td>
<td>-32.92</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>71.22</td>
<td>-1.88</td>
<td>-9.15</td>
<td>-26.77</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Known</td>
<td>76.88</td>
<td>-1.77</td>
<td>-9.47</td>
<td>-26.90</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>80.02</td>
<td>-1.14</td>
<td>-15.89</td>
<td>-31.30</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>72.12</td>
<td>-1.52</td>
<td>-7.42</td>
<td>-25.31</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Known</td>
<td>81.12</td>
<td>-1.39</td>
<td>-7.40</td>
<td>-25.02</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>83.06</td>
<td>-0.92</td>
<td>-12.82</td>
<td>-28.53</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>75.24</td>
<td>-1.00</td>
<td>-4.86</td>
<td>-23.08</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.4: Results of attempting to lose money under the continuous transaction assumption
Figure 4-4: Expected return on a single investment under the continuous transaction assumption when $D$ was known and the goal was to lose money.

**Guessing $D$**

When the algorithm was run on time periods of shorter length, the success rates when $D$ was guessed to be the least result from other time periods of the same length were slightly lower than when $D$ was known, but slightly higher than when the algorithm was run on longer time periods. When $D$ was guessed to be the mean result, success rates were consistently lower, but got closer to the other results as the time periods got longer.

In a parallel to the case in which the algorithm was trying to make money, when comparing the expected returns from a single investment, the results were best when $D$ was known, but the returns when $D$ was guessed to be the least result almost always outperformed the returns when $D$ was guessed to be the mean result from other time periods. When comparing annual and combined returns, once cutoffs were incorporated, the least guess generally outperformed the results from running the algorithm when $D$ was known, and the results from running the algorithm when $D$ was known were better than the results from running the algorithm when $D$ was guessed to be the mean result from other time periods. These results are consistent with the results of the algorithm when it was trying to make money.
Results of the Control Group

In an effort to verify that the algorithm was effective, the algorithm was run with random stock selection. Success rates dropped sharply when stocks were randomly selected. The random stock selection only rarely succeeded in losing money, and only did so when cutoff levels were tight. Except for annual return on a seven month time period with no cutoffs, the random stock selection never outperformed the algorithm’s choice, and only rarely was it even able (when combined with the alternate investment) to outperform the alternate investment alone.

The difference in expected return on a single investment under the continuous transaction assumption between the algorithm’s choice and the random stock selection can be seen in Figure 4-5. The algorithm’s choice always outperformed that of the random stock. The full results of random stock selection can be seen in Table 4.5.
<table>
<thead>
<tr>
<th>Months</th>
<th>Cutoff</th>
<th>$D$</th>
<th>Success %</th>
<th>Mean % Return</th>
<th>% Annual Return</th>
<th>% Return with 30%</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Known</td>
<td>59.50</td>
<td>1.27</td>
<td>23.10</td>
<td>12.26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>78.22</td>
<td>1.03</td>
<td>29.85</td>
<td>24.74</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>53.18</td>
<td>1.23</td>
<td>20.36</td>
<td>7.23</td>
<td></td>
</tr>
<tr>
<td>.25</td>
<td>Known</td>
<td>34.68</td>
<td>-0.26</td>
<td>-6.29</td>
<td>-20.47</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>41.74</td>
<td>-0.21</td>
<td>-10.75</td>
<td>-21.14</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>32.64</td>
<td>-0.27</td>
<td>-5.46</td>
<td>-20.88</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Known</td>
<td>40.30</td>
<td>-0.15</td>
<td>-3.27</td>
<td>-17.37</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>46.84</td>
<td>-0.15</td>
<td>-7.42</td>
<td>-17.97</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>37.14</td>
<td>-0.16</td>
<td>-2.96</td>
<td>-18.44</td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td>Known</td>
<td>47.20</td>
<td>0.04</td>
<td>0.28</td>
<td>-12.23</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>53.26</td>
<td>-0.05</td>
<td>-1.24</td>
<td>-11.79</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>42.92</td>
<td>0.04</td>
<td>0.21</td>
<td>-14.98</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Known</td>
<td>53.40</td>
<td>0.36</td>
<td>7.45</td>
<td>-4.68</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>61.82</td>
<td>0.12</td>
<td>4.60</td>
<td>-2.97</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>48.94</td>
<td>0.33</td>
<td>5.89</td>
<td>-8.02</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>Known</td>
<td>66.72</td>
<td>2.85</td>
<td>14.12</td>
<td>2.84</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>77.58</td>
<td>2.84</td>
<td>21.08</td>
<td>13.98</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>59.98</td>
<td>2.86</td>
<td>12.82</td>
<td>0.65</td>
<td></td>
</tr>
<tr>
<td>.25</td>
<td>Known</td>
<td>33.02</td>
<td>-0.42</td>
<td>-3.42</td>
<td>-20.63</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>33.92</td>
<td>-0.28</td>
<td>-4.84</td>
<td>-20.51</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>28.88</td>
<td>-0.30</td>
<td>-2.04</td>
<td>-19.81</td>
<td></td>
</tr>
<tr>
<td>.5</td>
<td>Known</td>
<td>40.18</td>
<td>-0.26</td>
<td>-2.01</td>
<td>-18.87</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>39.54</td>
<td>-0.15</td>
<td>-2.25</td>
<td>-17.81</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>35.10</td>
<td>-0.09</td>
<td>-0.50</td>
<td>-18.03</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Known</td>
<td>47.74</td>
<td>0.11</td>
<td>0.72</td>
<td>-15.28</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>48.26</td>
<td>0.07</td>
<td>0.76</td>
<td>-13.52</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>41.96</td>
<td>0.37</td>
<td>2.40</td>
<td>-14.47</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Known</td>
<td>56.78</td>
<td>0.72</td>
<td>4.96</td>
<td>-10.07</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>59.14</td>
<td>0.40</td>
<td>5.85</td>
<td>-7.52</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>50.76</td>
<td>1.06</td>
<td>6.45</td>
<td>-9.38</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>Known</td>
<td>68.26</td>
<td>4.32</td>
<td>12.55</td>
<td>1.54</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>80.66</td>
<td>3.80</td>
<td>18.25</td>
<td>11.64</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>67.70</td>
<td>3.61</td>
<td>10.33</td>
<td>-1.50</td>
<td></td>
</tr>
<tr>
<td>.25</td>
<td>Known</td>
<td>31.38</td>
<td>-0.54</td>
<td>-2.75</td>
<td>-20.66</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>34.76</td>
<td>-0.34</td>
<td>-4.53</td>
<td>-20.81</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>31.92</td>
<td>-0.50</td>
<td>-2.34</td>
<td>-20.46</td>
<td></td>
</tr>
<tr>
<td>1</td>
<td>Known</td>
<td>38.22</td>
<td>-0.32</td>
<td>-1.53</td>
<td>-19.18</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>41.20</td>
<td>-0.21</td>
<td>-2.61</td>
<td>-18.71</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>38.48</td>
<td>-0.29</td>
<td>-1.29</td>
<td>-19.15</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>Known</td>
<td>48.34</td>
<td>0.03</td>
<td>0.06</td>
<td>-16.70</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>49.72</td>
<td>0.03</td>
<td>0.12</td>
<td>-14.95</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>47.90</td>
<td>0.06</td>
<td>0.19</td>
<td>-16.93</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>Known</td>
<td>57.56</td>
<td>0.83</td>
<td>3.65</td>
<td>-12.07</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>60.48</td>
<td>0.40</td>
<td>4.40</td>
<td>-9.58</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>57.22</td>
<td>0.74</td>
<td>2.97</td>
<td>-13.05</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.5: Results of attempting to lose money under the continuous assumption and random stock selection
4.10.2 The Once-a-day Model

Unfortunately, when running the algorithm under the assumption that it could only buy and sell stocks at the beginning of the day, the expected return of the algorithm was always positive. Although the success rates were reasonable, its inability to purchase stock exactly when it wanted prevented it from losing money. Unlike previous results, the algorithm had better expected returns on a single investment at shorter time periods than it did at longer time periods. This result differs from the results of the other configurations. With the exception of an unusually high return when run on a seven month time period with no cutoffs, incorporating cutoffs did not seem to markedly affect the expected return on a single investment.

When running under the assumption that the algorithm could only buy and sell stock at the beginning of the day, the algorithm did not always make a profitable trade. Although its success rates were the same as those of the continuous model, the fact that it could only buy and sell stock at the beginning of the day prevented it from realizing the returns that the continuous model received. Often, even when it did make a profitable trade, the trade was insufficiently profitable for the algorithm combined with the alternate investment to outperform the alternate investment alone.

The expected returns under this model and assuming that \( D \) was known are shown in Figure 4-6. It can be seen that shorter time periods performed better, and except for the unusually high return when the algorithm was run on a seven month time period with no cutoffs, incorporating cutoffs did not affect the results significantly. The complete results for trying to lose money under the once-a-day model can be seen in Table 4.6.

Guessing \( D \)

When trying various guesses for \( D \), the least guess almost always outperformed the mean guess in terms of expected return on a single investment. In short time periods, the algorithm generally had better expected returns on a single investment when \( D \) was known than when \( D \) was guessed to be the least result from other time periods, but in longer time periods, the least guess outperformed the known guess, although both usually outperformed the mean guess.
<table>
<thead>
<tr>
<th>Months</th>
<th>Cutoff</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>$D$</td>
<td>Success %</td>
<td>Mean % Return</td>
<td>% Annual Return</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>94.14</td>
<td>0.14</td>
<td>3.30</td>
<td>-11.52</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>94.14</td>
<td>0.36</td>
<td>19.01</td>
<td>9.91</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>82.82</td>
<td>0.31</td>
<td>6.36</td>
<td>-9.61</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>85.38</td>
<td>0.34</td>
<td>10.73</td>
<td>-7.23</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>84.68</td>
<td>0.40</td>
<td>33.31</td>
<td>18.56</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>74.68</td>
<td>0.57</td>
<td>14.60</td>
<td>-4.92</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>85.94</td>
<td>0.36</td>
<td>10.15</td>
<td>-7.68</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>86.72</td>
<td>0.40</td>
<td>36.07</td>
<td>18.41</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>85.64</td>
<td>0.56</td>
<td>14.28</td>
<td>-5.13</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>87.24</td>
<td>0.30</td>
<td>9.07</td>
<td>-8.53</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>86.94</td>
<td>0.41</td>
<td>34.77</td>
<td>19.72</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>76.74</td>
<td>0.56</td>
<td>14.07</td>
<td>-5.21</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>89.10</td>
<td>0.26</td>
<td>7.79</td>
<td>-9.43</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>85.54</td>
<td>0.44</td>
<td>37.28</td>
<td>21.98</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>78.92</td>
<td>0.51</td>
<td>12.60</td>
<td>-6.28</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>94.76</td>
<td>0.16</td>
<td>1.13</td>
<td>-14.73</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>94.52</td>
<td>0.63</td>
<td>8.90</td>
<td>-3.74</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>87.34</td>
<td>0.55</td>
<td>3.34</td>
<td>-12.53</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>79.56</td>
<td>0.53</td>
<td>4.89</td>
<td>-14.82</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>80.52</td>
<td>0.38</td>
<td>7.89</td>
<td>-10.80</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>69.50</td>
<td>1.26</td>
<td>10.17</td>
<td>-10.68</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>80.22</td>
<td>0.55</td>
<td>5.00</td>
<td>-14.68</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>80.80</td>
<td>0.37</td>
<td>7.79</td>
<td>-10.91</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>79.22</td>
<td>1.29</td>
<td>10.40</td>
<td>-10.46</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>81.98</td>
<td>0.54</td>
<td>4.94</td>
<td>-14.68</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>82.16</td>
<td>0.41</td>
<td>8.55</td>
<td>-10.19</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>71.84</td>
<td>1.38</td>
<td>11.12</td>
<td>-9.77</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>85.38</td>
<td>0.51</td>
<td>4.61</td>
<td>-14.76</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>84.36</td>
<td>0.42</td>
<td>8.75</td>
<td>-9.77</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>74.76</td>
<td>1.54</td>
<td>12.32</td>
<td>-8.57</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>93.22</td>
<td>4.27</td>
<td>19.53</td>
<td>1.96</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>95.88</td>
<td>0.67</td>
<td>5.96</td>
<td>-5.37</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>88.84</td>
<td>4.11</td>
<td>17.25</td>
<td>-0.19</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>72.04</td>
<td>0.93</td>
<td>5.30</td>
<td>-14.91</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>77.18</td>
<td>0.50</td>
<td>7.81</td>
<td>-11.54</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>70.32</td>
<td>0.85</td>
<td>4.35</td>
<td>-15.74</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>73.78</td>
<td>0.88</td>
<td>5.00</td>
<td>-15.10</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>78.04</td>
<td>0.50</td>
<td>7.64</td>
<td>-11.64</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>71.22</td>
<td>0.92</td>
<td>4.73</td>
<td>-15.41</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>76.88</td>
<td>0.78</td>
<td>4.40</td>
<td>-15.53</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>80.02</td>
<td>0.46</td>
<td>7.01</td>
<td>-12.03</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>72.12</td>
<td>1.10</td>
<td>5.67</td>
<td>-14.59</td>
</tr>
<tr>
<td></td>
<td>Known</td>
<td>81.12</td>
<td>0.80</td>
<td>4.44</td>
<td>-15.26</td>
</tr>
<tr>
<td></td>
<td>Least</td>
<td>83.06</td>
<td>0.43</td>
<td>6.46</td>
<td>-12.22</td>
</tr>
<tr>
<td></td>
<td>Mean</td>
<td>75.24</td>
<td>1.38</td>
<td>7.12</td>
<td>-13.23</td>
</tr>
</tbody>
</table>

Table 4.6: Results of attempting to lose money under the once-a-day transaction assumption
4.11 Tests on New Data

If one throws enough darts at a dartboard, it is likely that at least one will hit the center. However, the fact that a particular dart was the one that hit the center does not mean that it will continue to hit the center. If a large number of different techniques are applied to a particular set of stock market data, at least one of them is likely to achieve good results. There is no guarantee that that technique will work on a different set of stock market data.

A similar problem might exist when looking at the results of this algorithm. Applying a large number of different configurations of the algorithm to the first five years of data might result in good returns for some configurations. If those configurations look good only because they happen to be the ones that were lucky enough to match the tested data set, then there would be no guarantee that they would work on other data sets. The fact that a large number of configurations of the algorithm achieved good returns implies that this is not the case, but the problem is still worth investigating.

In order to test this, a small set of configurations that seemed to perform well were tested on previously unused data. The configurations were chosen from both the continuous and once-a-day transaction models, and from both the attempts to make money and the
attempts to lose money. For each of these four models and goals, two setups were chosen. The first setup chosen was one that had previously yielded the optimum return when working with an alternate investment. The other setup had yielded a good return from a single investment. In all cases, the algorithm had to operate on a guess at \( D \), based on the results from the five years of previously tested data, and not on the two years of untested data. This helped to simulate the real world, where data on what the optimum returns will be are not available.

The results of the tests can be seen in Table 4.7. The algorithm did not perform as well on the new data as it had on the old. This is likely due to two factors. The first is that the choice of configurations was based on the results from the old data. The second is that the guesses at \( D \) were based on the old data, and so the tests on the new data were already likely to have lower results.

Success rates were lower for every test on the new data. This is most likely due to the fact that the guesses for \( D \) were not as good. In the continuous transaction model, although the returns were not as good as those on the previous data set, every run was successful. When the algorithm was supposed to make money, it did. When the algorithm was supposed to lose money, it did.

In the once-a-day transaction model, the algorithm performed poorly. When trying to make money, it failed when \( D \) was guessed as the mean optimum, and, though it succeeded at making money when \( D \) was guessed as the least optimum, the combined return of the algorithm and the 30% alternate investment was lower than that of the alternate investment alone. When trying to lose money, it also failed. Since the algorithm had not succeeded in losing money on the previous data set, it was expected that the algorithm would also fail to lose money on the untested data. It performed similarly to the results from the previous data. The results of the eight runs can be seen in Table 4.7.
Table 4.7: Results of running the algorithm on previously untested data starting January 1, 1996.
Chapter 5

Conclusions and Extensions

5.1 Conclusions

The choice of the best strategy depends greatly on the cost of transactions. In the simulations, these costs were assumed to be $0, but in the real world, they can be significantly higher. When the algorithm was run on shorter time periods it almost always generated lower returns from a single investment, but higher annual returns. When transaction costs are low, running the algorithm on short time periods is a better investment strategy than running the algorithm on longer time periods. When transaction costs are high, the low returns can become so low as to be insignificant, and the annual returns can drop sharply. If transaction costs are high, the algorithm appears to provide better results when run on longer time periods. Similarly, guessing $D$ conservatively tended to provide better annual returns but smaller returns on a particular investment. If transaction costs are low, a conservative guess is a better strategy. If transaction costs are high, a more daring guess is preferred.

In the real world, an investor has more opportunities to buy and sell stocks than in the once-a-day model. The investor has fewer opportunities to trade stocks, however, than assumed by the continuous transaction model. Results were not very promising for the once-a-day model, but were quite good for the continuous model. In the real world, the results are likely to fall somewhere in between.

Cutoffs failed in the once-a-day model. Investors in the real world, however, can buy and sell stocks more often than in the once-a-day model, and can even request automatic stock sales if the price passes certain thresholds. The ability to sell stock on shorter notice means
that cutoffs work better in the real world than in the once-a-day simulations. Since cutoffs worked so well in the continuous model, and allow investors to cut potentially devastating losses, it is recommended that cutoffs be used when applying this algorithm to the real world.

5.2 Suggestions for Further Research

The success of the program under various assumptions suggests that the program could actually be run as an investment tool. The program could be modified to run on current data. It could watch stock prices through a stock ticker system and run the real-life continuous transaction model. These results can then be compared to the results of running the simulation of the continuous transaction model on the closing prices for the same time period to get an idea of how misleading the assumptions are. If the assumptions are not so misleading as to prevent profit, then the program could be implemented to actually buy and sell stock, and one could attempt to make money in this manner.

It might also be advisable to extend the algorithm such that it runs concurrently on several time periods of different lengths. For instance, one might run the program on a six month time period, but while waiting for the algorithm to pick a stock in the six month period, run it on a one month time period.

The program could also be modified to work with different data sets. It might be a good predictor of options prices, commodity prices, or other markets. Though beyond the scope of this thesis to test these other data sets, it may be possible to use this algorithm to make a higher profit in some of those areas.

This algorithm could also be combined with other investment tools. For the purposes of simulation, the stocks in the current S&P 500 were used. If careful analysis found a group of several hundred stocks, all of which looked like they would perform well, then running the algorithm on those stocks might yield better returns. The research done for this thesis tests the validity of the algorithm on data close to that of the real world. It has numerous applications to various predictive problems where one good item must be picked out of a collection of items, not all of which are desirable. Applying the algorithm to any of these problems might be a worthwhile endeavor.
Appendix A

Main Program

```c
#include <time.h>
#include <stdio.h>
#include <math.h>

#define NUMBEROFDAYS 200
/*number of days for which we have to worry about stocks
200 covers up to about 9 months worth*/

#define NUMBEROFSTOCKS 600
/*number of stocks about which we have to worry*/

#define LEASTGAIN 107.0
/#have to define these for the test*/
#define MEANGAIN 307.05

#define LEASTLOSS -46.00

#define MEANLOSS -67.58

#define SEEDERVAL 0
/*if SEEDERVAL is 0 then a new seed is generated. Otherwise
the seed in SEEDERVAL is used*/

#define TRIALRUNS 50
/*the number of times the algorithm is to be run on the data set*/
```
/*these next defines allow the algorithm to run all of the tests on
the same time period in parallel. */

#define FIRSTVAL 6
#define SECONDVAL 3
#define THIRDVAL 5
/*size of the indices*/

#define GAIN 0
#define LOSE 1

#define FORWARDKNOWN 0
#define FORWARDLEAST 1
#define FORWARDMEAN 2
#define REVERSEKNOWN 3
#define REVERSELEAST 4
#define REVERSEMEAN 5
/*first indices*/

#define FINEGRAIN 0
#define COARSEGRAIN 1
#define RANDOMFINE 2
/*second indices*/

#define NOCUT 0
#define DOUBLECUT 1
#define ONECUT 2
#define HALFCUT 3
#define QUARTERCUT 4
/*third indices*/

/*GLOBAL VARIABLES*/
float targetpay[FIRSTVAL];

char longstocks[NUMBEROFSTOCKS][20];
/*holds the names of stocks*/

int spawncount[NUMBEROFSTOCKS][2];
/*tells how many stocks spawned for each base stock by
float spawnlows[NUMBEROFSTOCKS][NUMBEROFDAYS][2];
/*stock number in 1, spawn number in 2, holds low val*/

float spawnhighs[NUMBEROFSTOCKS][NUMBEROFDAYS][2];
/*for stock number in index 1, spawn number in index 2, day in index 3, holds the most recent high*/

float stockvals[NUMBEROFSTOCKS][NUMBEROFDAYS];
/*holds the close price for the stock in index 1 on the day in index 2*/

FILE *inputfile;
/*the file that holds the stock close prices*/

FILE *infofile;
/*output by reversecollate, holds the information on the best loss during the time period*/

FILE *outputfile;
/*output by this program*/

int numdays;
/*number of days in the data*/

int numcomm;
/*number of commodities in the data*/

float bestcomm[FIRSTVAL];
/*best amount you can make*/

float interest[FIRSTVAL][SECONDVAL][THIRDVAL];

float pickprob[FIRSTVAL];
/*probability that coin would come up heads when it did come up heads*/

int achievements[FIRSTVAL][SECONDVAL][THIRDVAL];
```c
float totalinterest[FIRSTVAL][SECONDVAL][THIRDVAL];

int targethit[FIRSTVAL][SECONDVAL][THIRDVAL];
/* holds the day the target was achieved if achieved, 
or the negative of the day the cutoff was triggered 
if triggered, or -1 if neither. */

int chosen[FIRSTVAL][SECONDVAL];
/* the index (into the arrays) of the stock that was chosen */

int whenchosen[FIRSTVAL];
/* the day the stock was chosen */

int spawnchosen[FIRSTVAL][SECONDVAL];
/* which spawn of the chosen stock was chosen */

float realinterest[FIRSTVAL][SECONDVAL][THIRDVAL];
/* ratio of money at end of time period to money at beginning 
of time period */

float pickval[FIRSTVAL][SECONDVAL];
/* the exact value at which stock was picked */

double log2(double x)
{
    /* Calculates the log base 2 of x */
    return(log(x)/log(2));
}

float halfpercents(float gainratio)
{
    /* returns the number of half percent increases/decreases, compounded 
each time, that have to be taken to achieve the gain ratio required */
    /* if gain ratio is 1.005, then obviously that’s exactly one half percent 
increase */
    if (gainratio >= 1)
        return((log(gainratio)) / (log(1.005)));
    if (gainratio < 1)
```
return( (log(gainratio)) / (log(0.995)) );
}

void read_stocks()
{
    /*Reads in the stock data from the output of reversecollate*/
    int i; /*index to which stock*/
    int j; /*index to which spawn*/
    int k; /*index to which day*/
    char name[20]; /*name of stock*/
    char prevname[20]; /*name of last stock*/
    long day; /*date of stock value*/
    float closeval; /*closing price of stock*/

    k = -1; /*0 is the first real index*/
    i = -1; /*0 is the first real index*/
    j = -1; /*0 is the first real index*/

    strcpy(prevname,"nothing\0"); /*clear the previous name*/
    for (;fscanf(inputfile,"Xs\t%d\t%f\n",name,&day,&closeval)!=EOF;)
    {
        if (strcmp(name,prevname)!=0) /*on a new stock now*/
        {
            i++; /*increment the stock counter*/
            k = 0; /*reset the day counter*/
            strcpy(prevname,name); /*so checking for new stock is accurate*/
            strcpy(longstocks[i],name); /*have to save the name*/
            spawncount[i][GAIN]=0; /*reset the spawncounter for new stock*/
            spawncount[i][LOSE]=0;
            spawnlows[i][0][GAIN] = closeval; /*low for non-spawn spawn is start*/
            spawnhighs[i][0][GAIN] = closeval; /*so is high*/
            spawnlows[i][0][LOSE] = closeval;
            spawnhighs[i][0][LOSE] = closeval;
        }
        else
        {
            k++; /*increment the day counter*/
        }

        stockvals[i][k] = closeval; /*store the closing val*/
    }
numdays = k+1; /* to handle the index offset(0 vs 1)*/
umcomm = i+1; /* to handle the index offset*/

/* set up all of the separate configs*/
for(i = 0;i<FIRSTVAL;i++)
    for(j = 0;j<SECONDVAL;j++)
        for(k = 0;k<THIRDVAL;k++)
            {
                targethit[i][j][k] = -1;
                chosen[i][j] = -1;
                whenchosen[i] = -1;
                spawnchosen[i][j] = -1;
                interest[i][j][k] = 1.0;
            }

printf("Number of commodities %d. Number of days %d.\n",numcomm,numdays);

void reset_stocks()
{
    {  
        int x,i,j,k;

        /* basically do what read_stocks does to reset things, but without rereading the stocks*/
        for (x = 0; x<numcomm; x++)
            {
                spawncount[x][GAIN] = 0;
                spawncount[x][LOSE] = 0;
                spawnhighs[x][0][GAIN] = spawnlows[x][0][GAIN];
                spawnlows[x][0][LOSE] = spawnhighs[x][0][LOSE];
            }

        for(i = 0;i<FIRSTVAL;i++)
            for(j = 0;j<SECONDVAL;j++)
                for(k = 0;k<THIRDVAL;k++)

```c
{ 
  targethit[i][j][k] = -1;
  chosen[i][j] = -1;
  whenc[chosen[i] = -1;
  spawnchosen[i][j] = -1;
  interest[i][j][k] = 1.0;
}
}

int genrand(int maxval)
{
  /*Returns a random integer between 0 and maxval-1, inclusive*/
  if (maxval == 0)
    return(0); /*Only one option, return 0*/
  else
    return(random()%maxval); /*Force it to be an integer between 0 and maxval-1*/
}

int choosenow(int comm, int whichday, int whichspawn, int numspawns, int firstindex, int gainloss)
{
  /*Perform the coinflips on one particular spawn of a stock on the given day*/

  int n; /*number of stocks (includes spawns)*/
  float DD; /*best value that can be obtained*/
  int randtarg; /*coin comes up heads 1 in randtarg*/
  int i; /*loop counter for each half percent*/
  float x; /*percent decrease from old low to new*/
  float y; /*new high/low value*/
  float z; /*old high/low value, then total (in/de)crease for this spawn until today*/
  double prob; /*probability coin will come up heads*/
  int steps; /*number of coin flips that need to be made*/
  int choice; /*on which step the coin came up heads, if any*/
  float d; /*the targeted (in/de)crease*/

  n = numspawns; /*number of stocks including spawns*/
  y = stockvals[comm][whichday]; /*the new high/low which triggered this*/
  if (gainloss == GAIN) /*make money*/
    
  
```
z = spawnhighs[comm][whichspawn][GAIN]; /*the old high*/
else /*lose money*/
    z = spawnlows[comm][whichspawn][LOSE]; /*the old low*/

x = y/z; /*z is ratio change*/
steps = (int)halfpercents(x); /*one for each half percent (in/de)crease (cumulative)*/
if (gainloss == GAIN) /*make money*/
    /*make z ratio change*/
    z = z/spawnlows[comm][whichspawn][GAIN];
else /*lose money*/
    /*make z ratio change*/
    z = z/spawnhighs[comm][whichspawn][LOSE];

z = (int)(halfpercents(z)); /*now z is in half percents*/

DD = bestcomm[firstindex];
d = targetpay[firstindex];

choice = 0; /*assume we don't choose anything*/

for (i=1;i<=steps;i++)
    {
        prob = DD; /*setting up probability*/
        prob = (3*(z+i))/prob;
        prob = prob - 2;
        prob = pow(((double)n),(prob));
        prob = prob/d;

        randtarg = (int)(1.0/prob); /*set up the coin*/
        if (randtarg == 0)
            randtarg = 1;
        if ((genrand(randtarg)) == 0) /*coin flip says pick it*/
            if (choice == 0)
                {
                    pickprob[firstindex] = prob;
                    choice = i;
                }
    }

return(choice); /*return on which flip it was picked, if any*/

void playmarket()
{
    int i; /*index on the commodity at which we are looking*/
    int k; /*index on the day at which we are looking*/
    int j; /*index on the spawn at which we are looking*/
    int x, y, z, gainloss;
    int choiceval; /*returned from choosenow, tells which flip was heads if any*/
    int numspawns[2]; /*total number of spawns out there*/
    float fraction; /*fraction of flips that day gone through*/
    int dirmult;
    int firsthigh;

    firsthigh = 0;

    for (k=1;k<numdays;k++) /*we never pick on day 1, we haven't made anything yet*/
    {
        numspawns[GAIN] = numcomm; /*time to calculate the number of stocks including spawns*/
        numspawns[LOSE] = numcomm;
        for (i=0;i<numcomm;i++)
        {
            numspawns[GAIN] = numspawns[GAIN] + spawncount[i][GAIN];
            numspawns[LOSE] = numspawns[LOSE] + spawncount[i][LOSE];
        }
        /*now need to create a new spawn for every stock that hit a new high/low*/
        for (i = 0; i < numcomm; i++)
        {
            if (stockvals[i][k] < spawnlows[i][spawncount[i][GAIN]][GAIN]) /*hit a new low*/
            {
                spawncount[i][GAIN] = spawncount[i][GAIN] + 1; /*increment the count*/
                spawnlows[i][spawncount[i][GAIN]][GAIN] = stockvals[i][k]; /*set the low*/
                spawnhighs[i][spawncount[i][GAIN]][GAIN] = stockvals[i][k]; /*and the high*/
            }
            if (stockvals[i][k] > spawnhighs[i][spawncount[i][LOSE]][LOSE]) /*hit a new high*/
            {
                spawncount[i][LOSE] = spawncount[i][LOSE] + 1; /*increment the count*/
                spawnlows[i][spawncount[i][LOSE]][LOSE] = stockvals[i][k]; /*set the low*/
                spawnhighs[i][spawncount[i][LOSE]][LOSE] = stockvals[i][k]; /*and the high*/
            }  
        }  
}
/*the new spawns don't contribute to the count. They can't be picked
since there won't be any change from date of creation (current day)
to current day. Also, the algorithm doesn't know which ones are
going to be created, so it shouldn't operate with this knowledge*/

for (x = 0; x < (FIRSTVAL / 2); x++)
    if (chosen[x][0] == -1)
        targetpay[x] = (bestcomm[x] / (3 * log2((float)numspawns[GAIN])));

for (x = (FIRSTVAL / 2); x < FIRSTVAL; x++)
    if (chosen[x][0] == -1)
        targetpay[x] = (bestcomm[x] / (3 * log2((float)numspawns[LOSE])));

for (i = 0; i < numcomm; i++) /*run through the stocks*/
    for (gainloss = 0; gainloss < 2; gainloss++)
        for (j = 0; j < spawncount[i][gainloss]; j++) /*run through the spawns*/
            if (((gainloss == GAIN) && (stockvals[i][k] > spawnhighs[i][j][GAIN]) ||
                ((gainloss == LOSE) && (stockvals[i][k] < spawnlows[i][j][LOSE])))
                /*hit a new low or high, depending on gainloss*/
            {
                for (x = (FIRSTVAL/2)*gainloss; x < ((FIRSTVAL/2) + (FIRSTVAL/2)*gainloss); x++)
                    if (chosen[x][0] == -1)
                        {
                        choiceval = choosenow(i, k, j, numspawns[gainloss], x, gainloss);
                        /*perform the flips*/
                        if (choiceval != 0) /*we chose it*/
                            {
                            chosen[x][FINEGRAIN] = i; /*record which stock*/
                            chosen[x][COARSEGRAIN] = i;
                            spawnchosen[x][FINEGRAIN] = j; /*which spawn*/
                            spawnchosen[x][COARSEGRAIN] = j;
                            whenchosen[x] = k; /*and when*/
                            if (gainloss == GAIN)
                                pickval[x][FINEGRAIN] =
                                    spawnhighs[chosen[x][FINEGRAIN]][spawnchosen[x][FINEGRAIN]][GAIN] *
                                    pow(1.005, choiceval); /*exact point of choice*/
                            else
                                pickval[x][FINEGRAIN] =
                            }
            }
spawnlows[chosen[x][FINEGRAIN]] = spawnchosen[x][FINEGRAIN][LOSE] * pow(0.995, choiceval); /*exact point of choice*/
pickval[x][COARSEGRAIN] = pickval[x][FINEGRAIN]; /*picking a stock at random instead*/
chosen[x][RANDOMFINE] = genrand(numcomm); /*pick a random stock*/
spawnchosen[x][RANDOMFINE] = genrand(spawncount[chosen[x][RANDOMFINE]][gainloss]); /*and a random spawn*/
/*the pickval needs to be the same fraction of the way into the day that the coin flip was in order to simulate buying the wrong stock at the right time*/
if (gainloss == GAIN)
    fraction = spawnhighs[i][j][GAIN] * pow(1.005, choiceval);
else
    fraction = spawnlows[i][j][LOSE] * pow(0.995, choiceval);
fraction = fraction - stockvals[i][k];
if (stockvals[i][k-1] == stockvals[i][k])
    fraction = 0;
else
    fraction = fraction / (stockvals[i][k-1] - stockvals[i][k]);
pickval[x][RANDOMFINE] = fraction * (stockvals[chosen[x][RANDOMFINE]][k-1] - stockvals[chosen[x][RANDOMFINE]][k]) + stockvals[chosen[x][RANDOMFINE]][k];
}
if (gainloss == GAIN)
    spawnhighs[i][j][GAIN] = stockvals[i][k]; /*since we hit a new high*/
else
    spawnlows[i][j][LOSE] = stockvals[i][k]; /*since we hit a new low*/
}
for (gainloss = 0; gainloss < 2; gainloss++)
for (x = 0 + (FIRSTVAL/2)*gainloss;
     x < (FIRSTVAL/2) + (FIRSTVAL/2)*gainloss;
     x++)
if (chosen[x][0] != -1)
for (y = 0; y < SECONDVAL; y++)
for (z = 0; z < THIRDVAL; z++)
/*handle the hitting the target or the cutoffs*/

if (targethit[x][y][z] == -1) {

    interest[x][y][z] = stockvals[chosen[x][y]][k]/pickval[x][y];
    dirmult = - 1 * (gainloss * 2 - 1);
    if ((dirmult * interest[x][y][z]) >= (dirmult * (pow((1 + dirmult * 0.005), targetpay[x]))))
        targethit[x][y][z] = k;
    if (z != NOCUT)
        if ((dirmult * interest[x][y][z]) <= (dirmult * (pow((1 - dirmult * 0.005), targetpay[x]*2.0/(pow(2,(z-1)))))
            { targethit[x][y][z] = -k;
        if (targethit[x][y][z] == -1)
            targethit[x][y][z] = -2;
    }
}

/*now calculate interest*/

for (gainloss = 0; gainloss < 2; gainloss++)
    for (x = 0 + (FIRSTVAL/2)*gainloss; x < ((FIRSTVAL/2) + (FIRSTVAL/2)*gainloss); x++)
        for (y = 0; y < SECONDVAL; y++)
            for (z = 0; z < THIRDVAL; z++)
                {
                    dirmult = -1 * (gainloss * 2 - 1);
                    if (targethit[x][y][z] == -1)
                        if (y == COARSEGRAIN)
                            interest[x][y][z] = stockvals[chosen[x][y]][numdays-1] / stockvals[chosen[x][y][whenchosen[x]]];
                        else
                            interest[x][y][z] = stockvals[chosen[x][y][numdays-1]] / pickval[x][y];
                    if (targethit[x][y][z] >= 0)
                        if (y == COARSEGRAIN)
                            interest[x][y][z] = stockvals[chosen[x][y][targethit[x][y][z]]] /
stockvals[chosen[x][y]][whenchosen[x]];
else
interest[x][y][z] = pow((1 + dirmult * 0.005),targetpay[x]);
if ((targethit[x][y][z]<=-2) && (z != NOCUT))
if (y == COARSEGRAIN)
interest[x][y][z] =
stockvals[chosen[x][y]][-1*targethit[x][y][z]] /
stockvals[chosen[x][y]][whenchosen[x]];
else
if (z != NOCUT) /*redundand*/
interest[x][y][z] =
pow((1 - dirmult * 0.005),targetpay[x]*2.0/(pow(2,z-1)));
}
for (x=0;x<FIRSTVAL;x++)
for (y=0;y<SECONDVAL;y++)
for (z=0;z<THIRDVAL;z++)
interest[x][y][z] = 100.0 *
(interest[x][y][z] - 1.0);

void printfirstinfo(int gainloss, int x, int y, int z)
{
/*print out the first stuff onto the output files*/
char whichoutfile[30];
sprintf(whichoutfile,"info.outfile%d%d%d",x,y,z);
outputfile = fopen(whichoutfile,"a");
if (chosen[x][y] != -1)
{
realinterest[x][y][z] = 1 + interest[x][y][z]/100;
if (targethit[x][y][z] >= 0)
achievements[x][y][z]++;
fprintf(outputfile,"Probability at choice was %.\n",pickprob[x]);
fprintf(outputfile,"Picked %s at ",longstocks[chosen[x][y]]);
if (y != COARSEGRAIN)
fprintf(outputfile,"%f on day %d, from ",pickval[x][y],whenchosen[x]);
else
fprintf(outputfile,"%f on day %d, from ",

stockvals[chosen[x][y]][whenchosen[x][x]]; 

if (gainloss == GAIN) 
    fprintf(outputfile,"low at Xf.\n", 
            spawnlows[chosen[x][y]][spawnchosen[x][y]][GAIN]);
else 
    fprintf(outputfile,"high at Xf.\n", 
            spawnhighs[chosen[x][y]][spawnchosen[x][y]][LOSE]);
if (targethit[x][y][z] >= 0) 
    fprintf(outputfile, "Target achieved on %d. Interest received: Xf\n", 
            targethit[x][y][z], 
            interest[x][y][z]);
else 
    fprintf(outputfile, "Target was not achieved. Interest received: Xf\n", 
            interest[x][y][z]);
if (targethit[x][y][z] <= -2) 
    fprintf(outputfile,"Cutoff triggered on day Xd.\n",(-1)*targethit[x][y][z]);
    fprintf(outputfile,"Real interest received: Xf\n",realinterest[x][y][z]);
} else 
{
    fprintf(outputfile,"No stock picked.\n");
    realinterest[x][y][z] = 1.0;
}
totalinterest[x][y][z] = totalinterest[x][y][z] + realinterest[x][y][z];
fclose(outputfile);
}

void printsecondinfo(int x, int y, int z)
{
    /*finish up the output files*/
    char whichoutfile[30];

    sprintf(whichoutfile,"info.outfile%d%d%d",x,y,z);
    outputfile = fopen(whichoutfile,"a");

    fprintf(outputfile,"Achievements: Xd.\n",achievements[x][y][z]);
    fprintf(outputfile,"Average interest: Xf\n",totalinterest[x][y][z]/TRIALRUNS);
    fprintf(outputfile,"Effective yearly interest: Xf.\n", 
            100 * (pow((totalinterest[x][y][z]/TRIALRUNS),(260.0/numdays)) - 1));

    }
fclose(outputfile);
}

void main()
{
    int i,j,k,l,zeroes,next,winners,losers;
    float interest, goodprobs, badprobs, total, putwin;
    int minloop, cutloop;
    char junkit[75];
    char commdstr[50];
    int junk1, junk2;
    float bestmin, bestcut, bestint;
    long seedused;
    float bestgaintemp,bestlosstemp;
    long junk3;
    int loopcount;
    int gainloss,x,y,z;

    printf("Starting.\n");
    system("rm info.outfile");
    system("cp info.file info.outfile");

    infofile = fopen("info.file","r");
    /*read data to get besttemp*/
    fscanf(infofile,"Seed used was %d.\n",&seedused);
    fscanf(infofile,"Starting date is %d.\n",&junk3);
    fscanf(infofile,"Ending date is Xd.\n",&junk3);
    fscanf(infofile,"Best gain was %f\n",&bestgaintemp);
    fscanf(infofile,"Best loss was %f\n",&bestlosstemp);
    bestcomm[FORWARDKNOWN] = halfpercents(1.0 + bestgaintemp/100.0);
    bestcomm[FORWARDLEAST] = halfpercents(1.0 + LEASTGAIN/100.0);
    bestcomm[FORWARDMEAN] = halfpercents(1.0 + MEANGAIN/100.0);
    bestcomm[REVERSEKNOWN] = halfpercents(1.0 + bestlosstemp/100.0);
    bestcomm[REVERSELEAST] = halfpercents(1.0 + LEASTLOSS/100.0);
    bestcomm[REVERSEMEAN] = halfpercents(1.0 + MEANLOSS/100.0);

    for (x = 0;x<FIRSTVAL;x++)
printf("Bestcomm %d is %f\n",x,bestcomm[x]);
   fclose(infofile); /*done with input*/
   printf("Info read.\n");
   if (SEEDERVAL == 0) /*new seed or old?*/
   {
       seedused = (int)time(0); /*generate the seed*/
   }
   else
   {
       seedused = SEEDERVAL; /*use the given seed*/
   }
   srand(seedused); /*seed the pseudo-random number generator*/
   outputfile = fopen("info.outfile","a"); /*open output file*/
   inputfile = fopen("tmp.data","r"); /*open data file*/
   read_stocks(); /*read the stocks*/
   fclose(inputfile); /*close data file*/
   fprintf(outputfile,"Seed used was %d.\n",seedused);
   /*so results can be duplicated*/
   fprintf(outputfile,"Number of days is %d.\n",numdays);
   /*record the number of days*/
   fclose(outputfile);
   /*start setting up outputfiles*/
   for (x = 0;x<FIRSTVAL;x++)
   {
       for (y = 0; y<SECONDVAL;y++)
       {
           for (z = 0; z<THIRDVAL;z++)
           {
               sprintf(commdstr,"cp info.outfile info.outfile%d%d%d",x,y,z);
               system(commdstr);
           }
           /*done setting up outputfiles*/
   }
   for (x=0;x<FIRSTVAL;x++)
for (y=0; y<SECONDVAL; y++)
    for (z=0; z<THIRDVAL; z++)
        {
            achievements[x][y][z] = 0;
            totalinterest[x][y][z] = 0;
        }

for (loopcount = 1; loopcount<=TRIALRUNS; loopcount++)
    {
        printf("%d-",loopcount); /*which iteration?*/

        inputfile = fopen("tmp.data","r"); /*extraneous*/
        reset_stocks(); /*reset the stocks, reset the spawns etc*/
        fclose(inputfile); /*you too*/

        playmarket(); /*run the algorithm*/

        printf("%d-Playing done. \n",loopcount);

        /*records status onto output file*/
        for (gainloss = 0; gainloss < 2; gainloss++)
            for (x = 0 + (FIRSTVAL/2)*gainloss;
                 x <= ((FIRSTVAL/2) + (FIRSTVAL/2)*gainloss);
                 x++)
                for (y = 0; y < SECONDVAL; y++)
                    for (z = 0; z < THIRDVAL; z++)
                        printfirstinfo(gainloss,x,y,z);

        for (x=0; x<FIRSTVAL; x++)
            for (y=0; y<SECONDVAL; y++)
                for (z=0; z<THIRDVAL; z++)
                    printsecondinfo(x,y,z);
    }
Appendix B

Data Preparing Program

```c
#include <stdio.h>
#include <string.h>
#include <math.h>
#include <time.h>

#define TIMEPERIOD 7
/*TIMEPERIOD defines the number of months of data over
 which the algorithm will be run*/

#define STARTINGYEAR 1990
/*the algorithm will generate time periods starting no earlier than
 January first of STARTINGYEAR*/

#define NUMBEROFYEARS 5
/*the number of years under consideration as start dates for time
 periods, including STARTINGYEAR*/

#define SEEDERVAL 0
/*if SEEDERVAL is 0, generate a new seed for the pseudo random number
 generator. Otherwise, use the value in SEEDERVAL*/

FILE *inputfile;
/*the input data file for each particular stock*/

FILE *outputfile;
/*the merged output data file*/
```
FILE *infofile;
/*the information file output by the program*/

FILE *datefile;
/*a file that lists the first and last date for each stock for which data exists. Some stocks do not have data starting at the beginning of 1990*/

/* 19900102 to 19980918 */

int genrand(int maxval)
{
    /*Returns a random integer between 0 and maxval-1, inclusive*/
    if (maxval == 0)
        return(0); /*Only one option, return 0*/
    else
        return(random()%maxval); /*Force it to be an integer in the range*/
}

void main()
{
    int seed;
    int intjunk;
    char stringjunk[10];
    char dirname[10];
    char filename[10];
    char commandstring[50];
    char filepath[20];
    char newpath[25];
    char junk;
    long firstdate;
    long lastdate;
    long curdate;
    long startdate;
    long enddate;
    float highval,lowval;
    float lossval,gainval;
    long lossdate,gaindate;
long endlossdate,endgaindate;
float bestloss,bestgain;
long bestlossdate,bestgaindate;
long endbestlossdate,endbestgaindate;
char bestlossfile[20];
char bestgainfile[20];
long highdate,lowdate;
int year;
int month;
int mdate;
float closeval;

system("rm info.file");
infofile = fopen("info.file","w");
/*set up the information file*/

if (SEEDERVAL == 0)
    /*generate a seed for the pseudorandom generator*/
    seed = (int)(time(0));
else
    seed = SEEDERVAL;

srandom(seed);

fprintf(infofile,"Seed used was \d.\n",seed);
/*record the seed*/
printf("Seed written.\n");

startdate = 19900101;
for (;startdate==19900101; /*last possible non valid starting date*/)
{
    /*randomly pick a starting date*/
    year = genrand(NUMBEROFYEARS) + STARTINGYEAR;
    printf("Chosen year for start is \d.\n",year);
    month = genrand(12) + 1;
    printf("Chosen month for start is \d.\n",month);
    mdate = genrand(31) + 1;
    printf("Chosen day for start is \d.\n",mdate);
    startdate = (year * 10000) + (month * 100) + (mdate);
/*pick the full time period*/
month = month + TIMEPERIOD;
if (month > 12)
{
    month = month - 12;
    year = year + 1;
}
enddate = (year * 10000) + (month * 100) + (mdate);
fprintf(infofile,"Starting date is %d.\n",startdate);
fprintf(infofile,"Ending date is %d.\n",enddate);

datefile = fopen("datefile","r");
system("rm tmp.data");
outputfile = fopen("tmp.data","w");
/*prepare the output data file*/
bestloss = 0.0;
bestgain = 0.0;
bestlossdate = 19760518; /*needed old date, just used my birthdate*/
bestgaindate = 19760518;
endbestlossdate = 19760518;
endbestgaindate = 19760518;
/*set up variables for calculation of best gain/loss*/
for (;;scanf(datefile,"%s %d %d",&filepath,&firstdate,&lastdate)!=EOF;)
{
    if ((firstdate <= startdate) && (lastdate >= enddate))
    /*if the stock has data for the entire time period*/
    {
        lossdate = 19760518;
        gaindate = 19760518;
        highdate = 19760518;
        lowdate = 19760518;
        lossval = 0.0;
        gainval = 0.0;
        highval = 0.0;
        lowval = 10000.0; /*sufficiently large*/
        inputfile = fopen(filepath,"r");
printf("Using file %s
",filepath);
for (; fscanf(inputfile,"%d  %fc
",&curdate,&closeval,&junk)!=EOF;)
    if ((curdate >= startdate) && (curdate <= enddate))
        /*if curdate is within the time period, then that line
         of the stock file is important*/
        {
            fprintf(outputfile,"%s  %d  %f
",filepath,curdate,closeval);
            if (((closeval-highval) * (100/highval)) < lossval)
                /*have a new, better loss for this stock*/
                {
                    lossval = ((closeval-highval) * (100/highval));
                    lossdate = highdate;
                    endlossdate = curdate;
                }

            if (((closeval-lowval) * (100/lowval)) > gainval)
                /*have a new, better gain for this stock*/
                {
                    gainval = ((closeval-lowval) * (100/lowval));
                    gaindate = lowdate;
                    endgaindate = curdate;
                }

            if (closeval > highval)
                /*have a new high*/
                {
                    highval = closeval;
                    highdate = curdate;
                }

            if (closeval < lowval)
                /*have a new low*/
                {
                    lowval = closeval;
                    lowdate = curdate;
                }
        }

    if (lossval < bestloss)
/ *have a new best loss overall*/
{
    bestloss = lossval;
    bestlossdate = lossdate;
    endbestlossdate = endlossdate;
    strcpy(bestlossfile,filepath);
}
if (gainval > bestgain)
/*have a new best gain overall*/
{
    bestgain = gainval;
    bestgaindate = gaindate;
    endbestgaindate = endgaindate;
    strcpy(bestgainfile,filepath);
}
close(inputfile);
}
/*print out the info*/
fprintf(infofile,"Best gain was %f\n",1.0 * (int)bestgain);
fprintf(infofile,"Best loss was %f\n",1.0 * (int)bestloss);
fprintf(infofile,"Gain occurred starting at %d in %s\n",bestgaindate,bestgainfile);
fprintf(infofile,"Gain ended on %d.\n",endbestgaindate);
fprintf(infofile,"Loss occurred starting at %d in %s\n",bestlossdate,bestlossfile);
fprintf(infofile,"Loss ended on %d.\n",endbestlossdate);
close(infofile);
close(outputfile);
close(datefile);
#include <stdio.h>
#include <string.h>
#include <math.h>

void main(int argc, char** argv)
{
    FILE *cutfile; /* file where cutoff info is stored */
    FILE *targetfile; /* file where info on achievements is stored */
    FILE *outfiles; /* list of files output from the algorithm */
    FILE *inputfile; /* current input file */
    FILE *outputfile; /* results output file */
    char currentfile[100];
    int cutdate; /* date cutoff triggered */
    int picked;
    int numdays;
    int buyday;
    int sellday;
    int nopicks;
    int curcutdate;
    int curcutline;
    int curline;
    int achieved;
    int progress;
    float interest;
char stringjunk[80];
char notword[10];
char maybe[10];
char subdir[30]; /*subdirectory for output files*/
char syscommd[100]; /*for preparing shell commands*/
int months; /*number of months on which the program was run*/
float localinterest;

if (argc != 2) /*just to remind me of number of arguments*/
    printf("Usage: compressresults <subdir>\n");

if (argc == 2)
{
    sprintf(subdir,"%s",argv[1]);
    sprintf(syscommd,"gunzip newautos/%s/*\n",subdir);
    /*gunzip any that happen to be compressed*/
    system(syscommd);
    system("rm outfile.list");
    sprintf(syscommd,"ls newautos/%s/info.outfile[0-9][0-9][0-9].[0-9] > outfile.list", subdir); /*generate the list of files*/
    system(syscommd);

    progress = 0;

    outfiles = fopen("outfile.list","r"); /*if there's nothing to do, don't remove compression.output*/
    if (fscanf(outfiles,"%s",stringjunk)!EOF)
    {
        system("rm compression.output");
        outputfile = fopen("compression.output","w");
        progress = 1;
    }
}

fclose(outfiles);

outfiles = fopen("outfile.list","r");

for (;fscanf(outfiles,"%s",currentfile)!EOF;)
{
    /*loop through each of the algorithm's output files*/

    /*Code continues here...*/
/ *prepare all of the little input files*/

```c
system("rm daynum.list");
sprintf(syscommd,"grep 'Number of days' %s > daynum.list",currentfile);
system(syscommd);

system("rm picked.list");
sprintf(syscommd,"grep Picked %s > picked.list",currentfile);
system(syscommd);

system("rm target.list");
sprintf(syscommd,"grep -n Target %s > target.list",currentfile);
system(syscommd);

system("rm cutoff.list");
sprintf(syscommd,"grep -n Cutoff %s > cutoff.list",currentfile);
system(syscommd);

system("rm nopick.list");
sprintf(syscommd,"grep 'No stock' %s > nopick.list",currentfile);
system(syscommd);

/*get the number of days*/

inputfile = fopen("daynum.list","r");
fsnscanf(inputfile,"%s %s %s %s %d%d", stringjunk,stringjunk,stringjunk,stringjunk,&numdays,stringjunk);
fclose(inputfile);

nopicks = 0;
/*figure out how many times nothing was picked*/

inputfile = fopen("nopick.list","r");
for(;fscanf(inputfile,"%s %s %s",stringjunk,stringjunk,stringjunk)!=EOF;)
nopicks++;
fclose(inputfile);

printf("%d %d\n",numdays,nopicks);

/*get the first cutoff date*/

curcutline = 0;
cutfile = fopen("cutoff.list","r");
if (fscanf(cutfile,"%d%10s %s %d%10d", &curcutline,stringjunk,stringjunk,stringjunk,
          stringjunk,&curcutdate,stringjunk)! = EOF)
    curcutline = curcutline - 1;
```
targetfile = fopen("target.list","r");
inputfile = fopen("picked.list","r");

/*start reading through and calculate when each stock was picked and when it was sold or time expired*/
for (;fscanf(targetfile,"Xd's Xs Xs Xs Xs Xf",
   &curline,stringjunk,stringjunk,notword,maybeday,
   stringjunk,stringjunk,&localinterest)!=EOF;)
{
    achieved = 0;
    fscanf(inputfile,"X/s Xs Xs Xs Xs Xs Xd/s Xs
   stringjunk,stringjunk,stringjunk,stringjunk,stringjunk,stringjunk,
   &buyday,stringjunk,stringjunk,stringjunk,stringjunk,stringjunk,stringjunk); 
    if (strcmp(notword,"not")!=0) /*it was sold via achievement*/
    {
        sscanf(maybeday,"Xd. ",&sellday);
        achieved = 1;
    }
    else if (curline == curcutline) /*it was sold because it cut off*/
    {
        sellday = curcutdate;
        if (fscanf(cutfile,"Xd's Xs Xs Xs Xd's",
           &curcutline,stringjunk,stringjunk,stringjunk,
           stringjunk,&curcutdate,stringjunk)!=EOF)
            curcutline = curcutline - 1;
    }
    else /*it didn't get sold until the last day*/
    sellday = numdays;
           numdays,buyday,sellday);
    fprintf(outputfile,"Interest: %4d Achieved: %4d\n",
           localinterest,achieved);
    /*record the info*/
}

for(;nopicks>0;nopicks--)
{
fprintf(outputfile,"Picked: 0 Days: %d Buyday: 0 Sellday: 0 ",numdays);
fprintf(outputfile,"Interest: 0.0 Achieved: 0\n");/*include the no picks*/
}

close(targetfile);
close(inputfile);
close(outfile);

sprintf(syscommd,"gzip %s",currentfile);
/*compress the current file to save space*/
system(syscommd);
}
close(outfiles);
if (progress == 1)
{
    close(outputfile);
    /*put compression.output in the right place*/
    sprintf(syscommd,"cp compression.output newautos/%s",subdir);
    system(syscommd);
}
}
Appendix D

Result Calculation Program 1

```c
#include <stdio.h>
#include <string.h>
#include <math.h>

void main(int argc, char** argv)
{
    FILE *inputfile; /*the compression.output file*/
    FILE *outputfile; /*the list of results*/
    char inputfilepath[80]; /*the full path of the compression.output file*/

    char stringjunk[80]; /*for sscanf*/
    char subdir[30]; /*subdirectory for output files*/
    char syscommd[100]; /*for preparing shell commands*/

    int pickedval; /*these are just read through fscanf*/
    int numdaysval; /*correspond to stuff output in */
    int buydayval; /*compression.output*/
    int selldayval;
    int achievedval;
    float interestval;

    int picked[10000]; /*hold the information from the various*/
    int numdays[10000]; /*lines of compression.output*/
    int buyday[10000];
    int sellday[10000];
```
int achieved[10000];
int indexcount; /*need to keep track of how many lines there were*/
int indexloop;
int achievesum;

float interest[10000];
float calculatedinterest; /*for successive approximation*/
float previnterest;

float averageinterest;
float averagedays;

float achieveperc;
float calcperc;
float strictperc;
float annualperc;

if (argc != 2)
    printf("Usage: averageautos <subdir>\n");

if (argc == 2)
{
    sprintf(subdir,"%s",argv[1]);
    sprintf(inputfilepath,"newautos/%s/compression.output",subdir);
    printf("Using %s\n",inputfilepath);
    /*figured out filepath*/

    inputfile = fopen(inputfilepath,"r");
    /*open the big input file*/

    outputfile = fopen("averages.results","a");
    /*prepare to tack another line onto the results*/

    indexcount = 0;
    achievesum = 0;

    for (; fscanf(inputfile,"%s %s %s %d %s %d %s %d %s %f %s %d", stringjunk,&pickedval,stringjunk,&numdaysval,stringjunk,&buydayval,
        stringjunk,&selldayval,stringjunk,&interestval,stringjunk,&achievedval)!=EOF;)

/*just a simple fscanf to read compression.output*/

if (pickedval == 0)
{
    interestval = 0.0;
    buydayval = numdaysval;
    selldayval = numdaysval;
}
picked[indexcount] = pickedval;
numdays[indexcount] = numdaysval;
buyday[indexcount] = buydayval;
seleday[indexcount] = selldayval;
interest[indexcount] = 1.0 + interestval/100.0;
achieved[indexcount] = achievedval;
achievesum = achievesum + achievedval;
indexcount++;
}

fclose(inputfile);

printf("%d\n",indexcount);

calculatedinterest = 1.0;
previnterest = 0.0;

printf("%lf\n",calculatedinterest);

/*now we loop through and until calculatedinterest is very close to previnterest which means we've got the right number*/
/*calculated interest is our guess at what the daily interest is while repeatedly running this algorithm*/
for (;;(((calculatedinterest - previnterest) >= 0.000001) ||
        ((calculatedinterest - previnterest) <= -0.000001));)
{
    previnterest = calculatedinterest;
    calculatedinterest = 0.0;
for (indexloop = 0; indexloop < indexcount; indexloop++)
    /*if I run the algorithm, I earn it's interest, plus
     the expected daily interest for the remaining days*/
calculatedinterest = calculatedinterest +
    interest[indexloop] * pow(previnterest,(260.0-sellday[indexloop]));

printf("%f\t",calculatedinterest);
calculatedinterest = calculatedinterest/indexcount;
printf("%f\t",calculatedinterest);
calculatedinterest = pow(calculatedinterest,(1.0/260.0));
printf("%f\n",calculatedinterest);
}
/*now I've calculated the average daily interest*/

averageinterest = 0.0;
averagedays = 0.0;

/*just calculate the average return*/
for (indexloop = 0; indexloop < indexcount; indexloop++)
{
    averageinterest = averageinterest + interest[indexloop];
    averagedays = averagedays + numdays[indexloop];
}

if (indexcount !=0)
    /*do some scalings up to a year*/
    {
    averageinterest = averageinterest/indexcount;
    averagedays = averagedays/indexcount;
    strictperc = 100.0 * (averageinterest - 1.0);
    annualperc = pow(averageinterest,260.0/averagedays);
    annualperc = 100.0 * (annualperc - 1.0);
    achieveperc = (100.0 * achievesum)/indexcount;
    }
else
    {
    averageinterest = 1.0;
    averagedays = 0.0;
    strictperc = 0.0;
annualperc = 0.0;
achieveperc = 0.0;
}

/*print it all*/
printf("Strict Average, Time period: %f\n",strictperc);
printf("Scaled to a year it's roughly: %f\n",annualperc);
calculatedinterest = pow(calculatedinterest,260.0);
printf("Full Interest Yearly: %f\n",calculatedinterest);
calcperc = 100.0 * (calculatedinterest - 1.0);
printf("As Interest: %f\n",calcperc);

printf("Achievement percentage: %f\n",achieveperc);
printf("Total number %d\n",indexcount);

fprintf(outputfile,"\tAchievements:%.2f Strict:%.2f Annual:%.2f Full:%.2f ",
    subdir,achieveperc,strictperc,annualperc,calcperc);

fprintf(outputfile,"Number: %d days: %.1f\n",indexcount,averagedays);
fclose(outputfile);
Appendix E

Result Calculation Program 2
(Alternate Investments)

```c
#include <stdio.h>
#include <string.h>
#include <math.h>

void main(int argc, char** argv)
{
    FILE *inputfile; /*the compression.output file*/
    FILE *outputfile;/ *the list of results*/
    char inputfilepath[80]; /*the full path of the compression.output file*/
    char stringjunk[80]; /*for fscanf*/
    char subdir[80]; /*subdir for output files*/

    int pickedval; /*these are read through fscanf from compression.output*/
    int numdaysval; /*and stored in the arrays*/
    int numdays[10000];
    int buydayval;
    int buyday[10000];
    int selldayval;
    int sellday[10000];
    int achievedval;
    int indexcount;
    int indexloop;
```
float previnterest; /*for successive approximation*/
float otherinterest;
float calculatedinterest;
float calcpere;
float interestval;
float interest[10000];

if (argc != 3)
    /*<interest> is the alternate interest, 30 or -20*/
    printf("Usage: compete <interest> <subdir>\n");

if (argc == 3)
{
    sprintf(subdir,"%s",argv[2]);
    sprintf(inputfilepath,"newautos/%s/compression.output",subdir);
    printf("Using %s\n",inputfilepath);

    inputfile = fopen(inputfilepath,"r");
    outputfile = fopen("compare.results","a");

    sscanf(argv[1],"Xf ",&otherinterest);
    otherinterest = 1.0 + otherinterest/100;
    otherinterest = pow(otherinterest,(1.0/260.0));
    /*now we've read the arguments and prepared the input/output
     files*/

    indexcount = 0;

    /*read the necessary information from compression.output*/
    for (;fscanf(inputfile,"%s %d %s %d %s %d %s %f %s %d",
               stringjunk,&pickedval,stringjunk,&numdaysval,stringjunk,&buydayval,
               stringjunk,&selldayval,stringjunk,&interestval,stringjunk,&achievedval)!=EOF;)
    {
        if (pickedval == 0)
        {
            interestval = 0.0;
            buydayval = numdaysval;
            selldayval = numdaysval;
        }
numdays[indexcount] = numdaysval;
buyday[indexcount] = buydayval;
sellday[indexcount] = selldayval;
interest[indexcount] = 1.0 + interestval/100.0;
indexcount++;
}

fclose(inputfile);
calculatedinterest = otherinterest;
previnterest = 0.0;

/* go until calculated interest is very close to previnterest
 calculated interest is supposed to be the average daily
 return on investment when the alternate and algorithm
 combine */
for (; (((calculatedinterest - previnterest) >= 0.0000001) ||
        ((calculatedinterest - previnterest) <= -0.0000001));)
{
    previnterest = calculatedinterest;
    calculatedinterest = 0.0;

    /* we earn alternate until we buy, then the stock interest,
 then, once we've sold, we earn calculated interest for
 the rest of the year */
    for (indexloop = 0; indexloop < indexcount; indexloop++)
        calculatedinterest = calculatedinterest +
            pow(otherinterest,(buyday[indexloop]) - 1) *
            interest[indexloop] *
            pow(previnterest,(260.0 - sellday[indexloop]));

    /* we take the arithmetic mean here */
    calculatedinterest = calculatedinterest / indexcount;
    calculatedinterest = pow(calculatedinterest,(1.0/260.0));
}

otherinterest = calculatedinterest;
/* output it all */
otherinterest = pow(otherinterest,260.0);
calcperc = 100.0 * (otherinterest - 1.0);
printf("Yearly percentage: \%.2f\n", calcperc);
fprintf(outputfile, "%s \tAlternate: %s Full: %.2f\n", subdir, argv[1], calcperc);
}
Bibliography

