Exploration vs. Exploitation: A Study in Optimization

Contents

1. Abstract

This paper examines a real-world optimization problem. Simple Moving Average Crossovers represent the the basis of technical analysis. Finding the optimal crossover would allow an investor to maximize return while minimizing the time required to achieve that return. Those interested in computational optimization may find this paper interesting for its technical background, while individual investors might gain insight as to how to increase their trading efficiency.

2. Purpose

The fundamental issue in optimization is the tradeoff between optimality and processing time. Given an infinite amount of time, even the simplest optimization algorithm can locate extrema in any function. Industry, however, requires definitive results given finite resources. Hence, one often sacrifices optimality for speed. One can either meticulously explore a data set, or he can exploit one or several local extrema deemed "good enough." This project will investigate different optimization methods in order to minimize the loss in optimality involved in seeking faster computation time. The data set chosen for this project comes from the field of finance.

The purpose of this project is to apply optimization algorithms to financial data in order to find two Simple Moving Average time frames whose crossover allows investors to quickly predict movements in stock price with a high level of confidence.

3. Introduction

One of computer science's greatest boons to science in general has been optimization. The ability to analyze vast sets of data for patterns, trends, and significance has engendered monumental strides in every field from medicine to mathematics. One such area of application that has benefitted from the plethora of optimization strategies is economics/finance. Financial analysis methods have become increasingly mathematically rigorous/complex.

There are many factors that affect stock price. Company performance, economic health, and extraneous incidents all drive changes in share value. The investor seeks to predict which way a stock will move in order to be able to buy or sell appropriately in advance. Whereas natural disasters or political changes cannot be adequately predicted for the purposes of trading, there are mathematical indicators that can be.

The most basic such indicator is the Simple Moving Average (SMA). Using SMAs, the investor can predict whether a stock will rise or fall in the short term. The *x*-day SMA is the average price of a stock over the previous x days. SMAs can be calculated for any length of time, and can be plotted on a price vs. time chart.

Below, Figure 1 illustrates the ten and twenty-day SMAs of a hypothetical stock. Points 1 and 2 are SMA "crossovers" because one SMA crosses over the other. A principle of financial analysis is that when a short term SMA (e.g. ten day) crosses above a long term SMA (e.g. twenty day), as happens at point 1, the stock price should continue to rise in the short term. The stock price should fall if a short term SMA crosses below a long term SMA, as happens at point 2. These two trading rules constitute the predictive value of SMAs. Thus, if one bought this stock at point 1, he should be able to expect to make a profit in the short term.

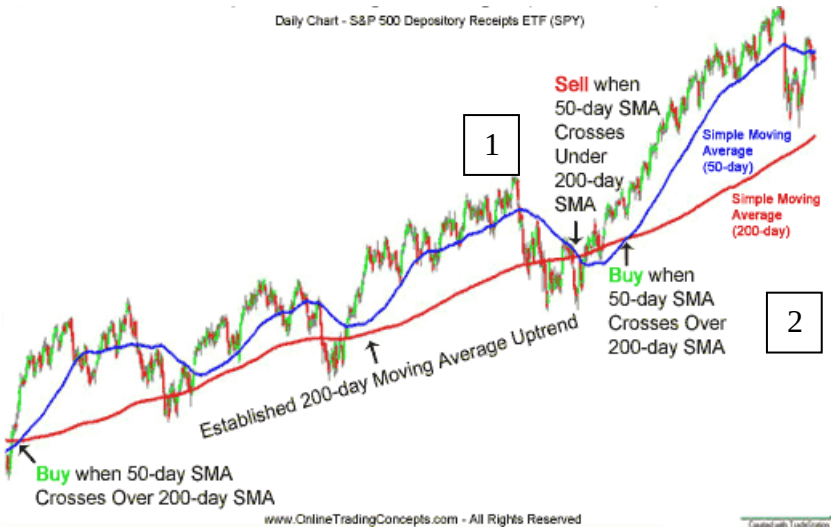

Fig 1 Crossover of Simple Moving Averages

In theory, crossover rules should work for any short term and any long term SMA, whether it be a two-day SMA crossing over a three-day SMA, or a fifty-day SMA crossing over a two hundred-day SMA. However, crossovers don't always make accurate predictions, especially when shorter time periods are used. If a fifty-day SMA crosses above a two hundred-day SMA, one can be fairly certain that this crossover bodes well for the stock. However, if a two-day SMA crosses over a three-day SMA, the crossover may just be the result of market fluctuations and may have no significance for future pricing. Thus, crossovers based on longer time periods will more consistently result in future price increases. The trade off, of course, is that one may lose an opportunity if he waits for a long term crossover. Waiting for a fifty-over-two hundred crossover to occur will cost the trader all of the money he could have made if he had bought the stock earlier.

Thus, one wants to find a crossover that will simultaneously maximize consistency and minimize waiting time. This relationship can be expressed in the following equation

$$f(short\ length, long\ length) = \frac{average\ \%\ change}{time}$$

Time is some expression based on the short and long lengths, possibly the average of the short and long lengths.

$$\text{time} = \frac{shortlength + longlength}{2}$$

A hypothetical example of function *f* is

$$f(5,10) = \frac{5\%}{\frac{5+10}{2}} = 7.5\frac{\%}{days}$$

So a 5-over-10 crossover yields a value of 7.5 %*days^{-1}. Maximizing function *f* will yield a short and long length that maximize consistency and minimize waiting time.

Since the function is not continuous, calculus cannot be used for optimization; other methods must be explored. This project would investigate different optimization methods to find the global maximum of *f* on the graph below (Figure 2). These different methods are compared below based on their applicability to this problem.

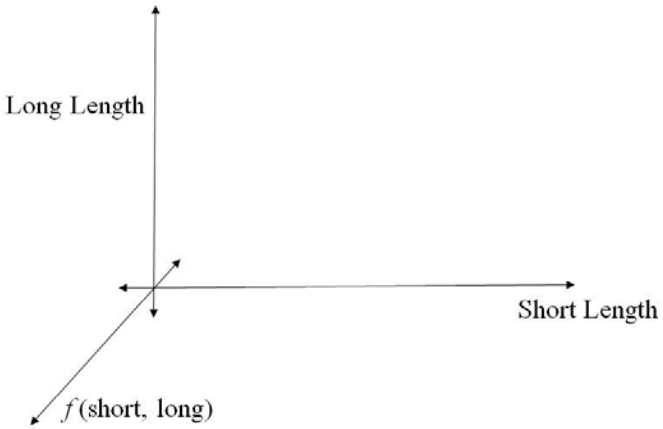

Figure 2 Axes of Function *f*

4. Technical Approach

Mathematics and computer science contain many optimization methods. However, there are unique constraints of this problem that preclude many of these techniques:

1. The function to be maximized is not continuous, and therefore not differentiable, which automatically eliminates the more obvious optimization strategies involving calculus (Hessians, gradients) such as Newton's Method and Gradient Descent.
2. The state space for this project is assumed to be fairly uneven and contain many local extrema. Thus, local search techniques (e.g. Tabu Search) would be misapplied here. (Note, Hill Climbing is used as an academic example of what will probably not work well).
3. Special purpose algorithms (i.e. shortest path algorithms such as Dijkstra) are not applicable here.

Based on the above constraints, the following optimization methods were chosen for this project.

4.1 Methodology

Some of the following optimization techniques are specifically designed for minimization. Since this problem requires maximization, the function *f* will be multiplied by negative one for the minimization methods. Algorithms for which this technicality must be handled are indicated by the (-) sign.

4.1.1 Particle Swarm Optimization (PSO)

PSO, originally proposed by Kennedy and Eberhart in 1995, is based on the natural behavior of birds trying to find food. Initially, birds scatter across the search space. Those who are close to a food source transmit the information to the rest of the flock (or "swarm"). The other birds (or "particles") then move toward the bird(s) that are close to food, possibly encountering different food sources in the process. Eventually, the flock converges on a food source (Eberhart R, Kennedy J, 1995)[1].

A population of "particles" is instantiated distributed uniformly across the state space. Each particle knows its best previous position, as well as the global best position discovered so far. In each iteration, every particle moves according to the following equation (De Oca 2007)[2]:

$$x^{t+1} = x^t + v^{t+1}$$

$x^{t+1} = new\ position$
$x^t = current\ position$
$v^{t+1} = velocity = wv^t + a_1 b_1 (M_P - x^t) + a_2 b_2 (M_G - x^t)$
$w = inertia\ weight$
$v^t = inertia = previous\ velocity$
$a_1, a_2 = constant\ weight\ factors$
$b_1, b_2 = random\ weight\ factors\ on\ interval\ (0,1)$
$M_P = personal\ maximum$
$M_G = global\ maximum$

Thus, each iteration, every particle moves toward both its personal best location, and the global best location so far discovered. Eventually, the particles should converge to the global maximum.

4.1.2 Nelder-Mead (-)

The Nelder-Mead Simplex Algorithm, first published in 1965 (Nelder and Mead, 1965), is a popular direct search method for multidimensional unconstrained minimization. Nelder-Mead is a simple optimization algorithm that manipulates simplexes (triangles here because this is a two-dimensional problem) to surround an extrema, and then converge on it. The algorithm evaluates a function at the vertices of the simplex, and accordingly decides how to shift to approach the extrema.

---

[1] http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=488968&tag=1
http://www.swarmintelligence.org/tutorials.php
[2] http://iridia.ulb.ac.be/~mmontes/slidesCIL/slides.pdf

Three random vectors, **A**, **B**, and **C**, that serve as the vertices of a triangle are created. A vector **D** is created by the reflection of the minimum vertex, **A** here, across the midpoint of the segment connecting the other two vertices, **B** and **C**, as shown by the vector formula

$$\vec{D} = \vec{B} + \vec{C} - \vec{A}$$

If the value of **D** is less than that of **A**, then the algorithm has moved in the correct direction, and should take another step of the same magnitude in that direction to create vertex **E**

$$\vec{E} = \frac{3(\vec{B} + \vec{C})}{2 - 2\vec{A}}$$

If the value of **D** is equal to that of **A**, then the simplex is contracted by forming two new vertices, **F** and **G**

$$\vec{F} = \frac{2\vec{A} + \vec{B} + \vec{C}}{4}$$

$$\vec{G} = \frac{3(\vec{B} + \vec{C}) - 2\vec{A}}{4}$$

The minimum of **F** and **G** is kept.

If neither **F** nor **G** is smaller than **A**, the simplex shrinks by moving the side that connects **A** and **C** toward **B**, creating the new vertices **I**, on **CB**, and **H**, on **AB**:

$$\vec{H} = \frac{\vec{A} + \vec{B}}{2}$$

$$\vec{I} = \frac{\vec{B} + \vec{C}}{2}$$

This process is repeated until the simple surrounds the minimum and converges on it.

Nelder-Mead will probably get stuck at a local minimum, but we'll see.

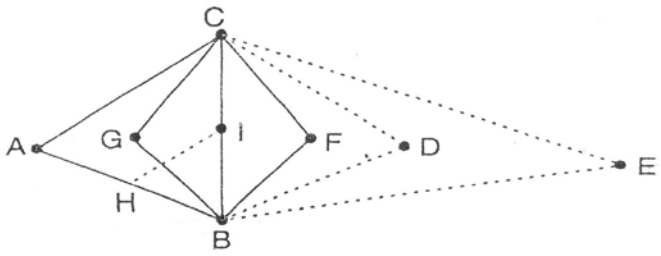

Figure 3 Nelder Mead Schematic

4.1.3 Genetic Algorithms (GAs)

GAs (Holland, 1975) are based on the principle of "survival of the fittest"[3]. First a population of "chromosomes," which are lists of parameters that serve as inputs for the fitness function, is randomly instantiated. Next, certain chromosomes are selected for breeding. Each pair of two selected chromosomes ("parents") produces two children. Children are either selected or discarded, as are their parents, based on their fitness values. After multiple iterations ("generations"), the algorithm should converge to the best chromosome[4].

GAs have three steps: Initialization, Selection, Breeding, and Replacement.

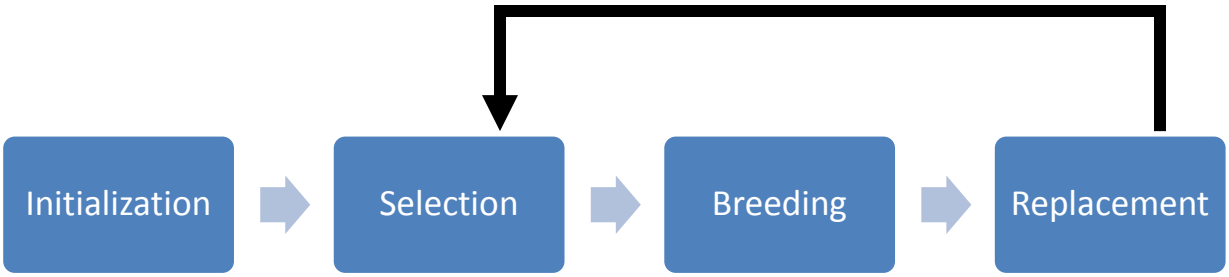

Figure 4 Genetic Algorithms Process

Initialization:
A population of chromosomes is randomly instantiated. Each chromosome has multiple genes. Chromosomes of genes can be represented as lists of parameters:

$$\text{chromosome } 1 = [\text{parameter } 1, \text{parameter } 2 \ldots]$$

Selection:
Next, certain chromosomes are selected to breed. Although there are multiple methods for selection, in order to avoid local extrema (since fitter in the immediate sense will not necessarily lead to the global optimum, given that there will probably be many local extrema), the simple method of breeding the top half (arranged by fitness) of the population with the bottom half will be used here.

Breeding:
After selection, the population is bred by random crossovers; a chromosome is split at a random index and swaps one segment with another chromosome as shown below:

$$Before\ breeding:$$

$$\boldsymbol{Chromosome\ 1} = [\mathbf{1}, \mathbf{2}, \mathbf{3}, \mathbf{4}]$$

[3] http://mathworld.wolfram.com/GeneticAlgorithm.html
[4] http://www.mathworks.com/discovery/genetic-algorithm.html

$$\boldsymbol{Chromosome\ 2} = [\mathbf{5}, \mathbf{6}, \mathbf{7}, \mathbf{8}]$$

$$After\ Breeding:$$

$$\boldsymbol{Chromosome\ 1} = [\mathbf{1}, \mathbf{2}, \mathbf{7}, \mathbf{8}]$$
$$\boldsymbol{Chromosome\ 2} = [\mathbf{5}, \mathbf{6}, \mathbf{3}, \mathbf{4}]$$

Replacement:
After breeding, certain chromosomes are replaced in the population. Again, there are several methods for replacement, but in order to avoid local extrema and ensure that the algorithm does not lose diversity and converge too quickly, the entire population will be replaced after each round of breeding.

Selection, Breeding, and Replacement continue until the end condition is reached (the chromosomes converge to the same value).

4.1.4 Differential Evolution (DE)

DE, introduced by Storn and Price (1997), can be considered a subset of genetic algorithms. The defining characteristic of DE is relies on mutation rather than breeding to engender new chromosomes. Although there is apparently no proof of convergence for DE, it has been shown to be more accurate than GAs and other forms of evolutionary programming[5]. DE's most appealing element for this project is the mutation process, which could potentially help the algorithm avoid local extrema that GAs might get stuck in.

DE has four stages: Initialization, Mutation, Recombination, and Selection.

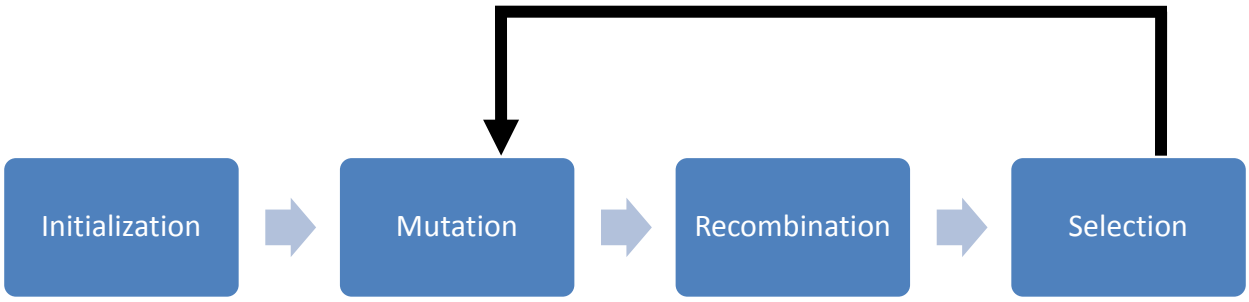

Figure 5 Differential Evolution Process

[5] http://www.maths.uq.edu.au/MASCOS/Multi-Agent04/Fleetwood.pdf

Initialization:
A random population of chromosomes is randomly instantiated in the form:

$$x_{i,G} = [x_{1,i,G}, x_{2,i,G}, \ldots x_{D,i,G}]$$

i ϵ [1, N], where i is the identifier of each chromosome
N is the number of chromosomes in the population.
G is the generation number.
D is the number of parameters in each chromosome.

Mutation:
For each "parent vector" $x_{i,G}$, random indices r_1, r_2, r_3 ϵ (1, N) are chosen such that i, r_1, r_2, and r_3 are distinct.

For each index i, a "donor vector" $v_{i,G+1}$ is created by adding the weighted difference of two of the randomly chosen vectors to the third for each parameter:

$$v_{i,G+1} = x_{r1,G} + F(x_{r2,G} - x_{r3,G})$$

The "mutation factor" F is a constant in [0, 2].

Recombination:
Next, recombination proceeds. A for each index i, a "trial vector" $u_{i,G+1}$ is created from elements of the parent vector and the donor vector:

$$u_{j,i,G+1} \begin{cases} v_{j,i,G+1} & if\ rand_{j,i} < CR\ or\ j = I_{rand} \\ x_{j,i,G+1} & if\ rand_{j,i} > CR\ and\ j \neq I_{rand} \end{cases}$$

j ϵ [1, D]
$rand_{j,i}$ is a random number from (0,1)
CR is a constant proabability set by the user.
I_{rand} is a random integer from (0,1) that ensures that at least one element in $u_{i,G+1}$ comes from the donr vector.

Selection:
After recombination is selection. Each trial vector is compared to its parent. The vector with the higher fitness value is admitted to the next generation:

$$x_{i,G+1} \begin{cases} u_{i,G+1}\ if\ f(u_{i,G+1}) > f(x_{i,G}) \\ x_{i,G}\ otherwise \end{cases}$$

Mutation, Recombination, and Selection continue until the end condition is reached (the chromosomes converge to the same value)[6].

[6] www.maths.uq.edu.au/MASCOS/Multi-Agent04/Fleetwood.pdf

4.1.5 Simulated Annealing (SA) (-)

Simulated Annealing, proposed by Kirkpatrick, Gelett and Vecchi (1983) and Cerny (1985), emulates the physical process whereby a metal is heated and allowed to re-cool so that weaknesses in its structure disappear and the lowest energy (i.e. the most stable) crystal lattice structure is formed.

Simulated Annealing begins with an initial position instantiated somewhere in the state space. A neighboring position is picked at random and is compared to the current position. If the position is better than the current position, it is accepted. If the position is worse, it is accepted with probability[7]:

$$e^{\frac{-(E-E_0)}{T}}$$

$$E = new\ position$$
$$E_0 = current\ position$$
$$T = temperature$$

Accepting some worse values helps the algorithm avoid getting stuck in local extrema like Hill Climbing. Over an infinite time, SA can be expected to find the global optimum, not that that is much help here because the whole point of this project is to find a time-efficient method.

4.1.6 Hill Climbing

Hill Climbing operates based on the analogy of a climber ascending a mountain in the dark. The algorithm starts at a random position, and checks that position's neighbors. The algorithm then assumes the position of the best neighbor. If no neighbors are better than the current position, the algorithm has finished. Evidently, Hill Climbing often gets stuck at local extrema. Hill Climbing is used here purely as an academic example. Although Hill-Climbing is not expected to work well on this data set, it should provide a good basis line, a control sample, against which to compare the results of the other methods.

The algorithm starts at a random position, and checks all neighbors of that position. The algorithm then assumes the position of the best neighbor. If no neighbors are better than the current position, the algorithm has finished. Evidently, Hill Climbing often gets stuck at local extrema. Hill Climbing is used here purely as an academic example.

4.2 Data

Each optimization algorithm was implemented and tested on historical Dow Jones data. How well an algorithm optimized the fitness function was determined by its consistency in finding the global maximum. For example, Hill Climbing proved a poor fit for this fitness function because each time it was executed it would identify a different point as the global

[7] http://www.sph.umich.edu/csg/abecasis/class/2008/615.18.pdf

maximum. Based on this procedure, PSO was determined to optimize the fitness function the best, as it fairly consistently identifies the global maximum of the fitness function.[8]

4.3 User Interface

In the GUI below, the user inputs the optimization method he wants to use, the stock or index he wants the algorithm to act upon, and the date range he wants the algorithm to consider.

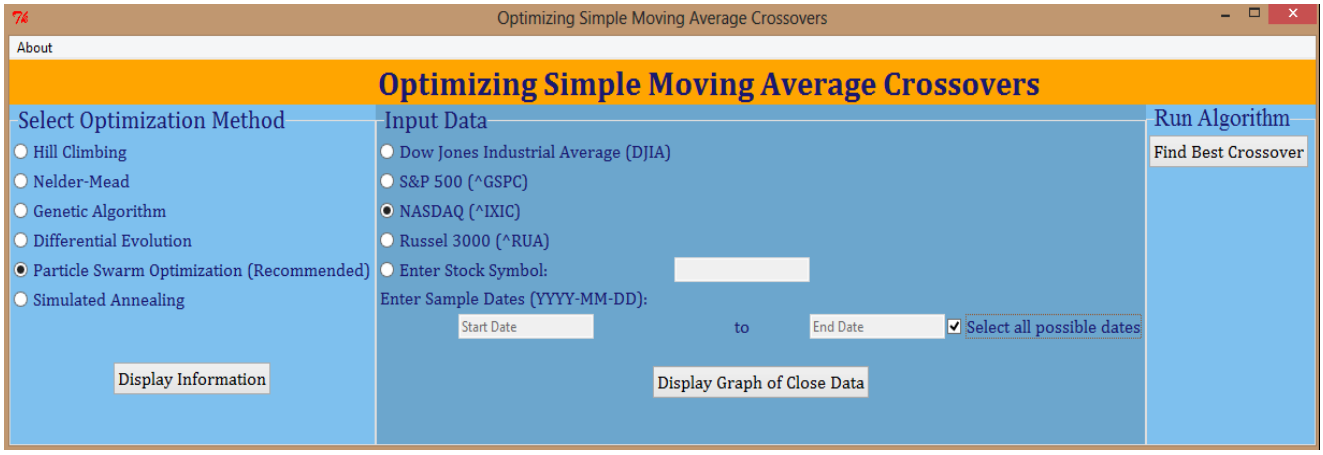

Figure 6 Model of GUI

From the GUI, the user can output the daily close data of the stock or index, shown below.

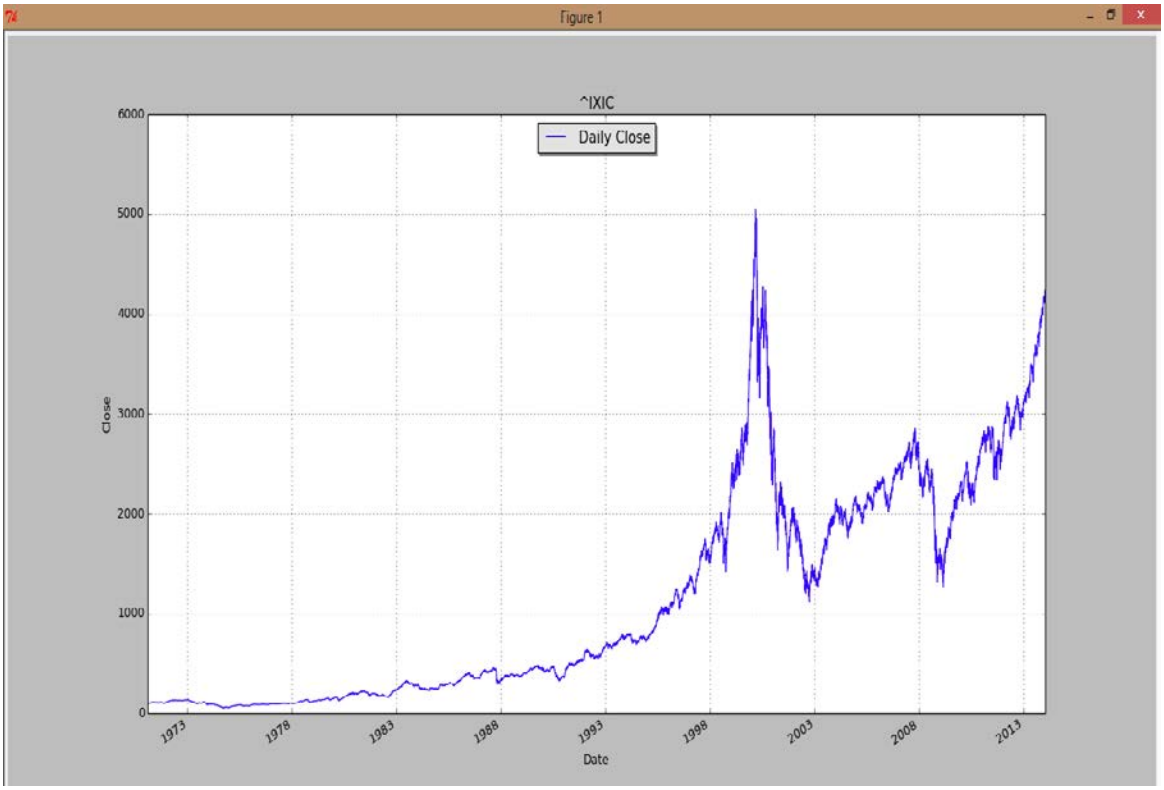

Figure 7 Daily Close Output

[8] https://research.stlouisfed.org/fred2/series/DJIA/downloaddata?cid=32255/

Once the user executes the algorithm, the GUI will output the best crossover for the given stock or index, for the given time period. In this example, the best crossover for the NASDAQ since its creation in 1971 has been a 2-day SMA crossing over a 6-day SMA.

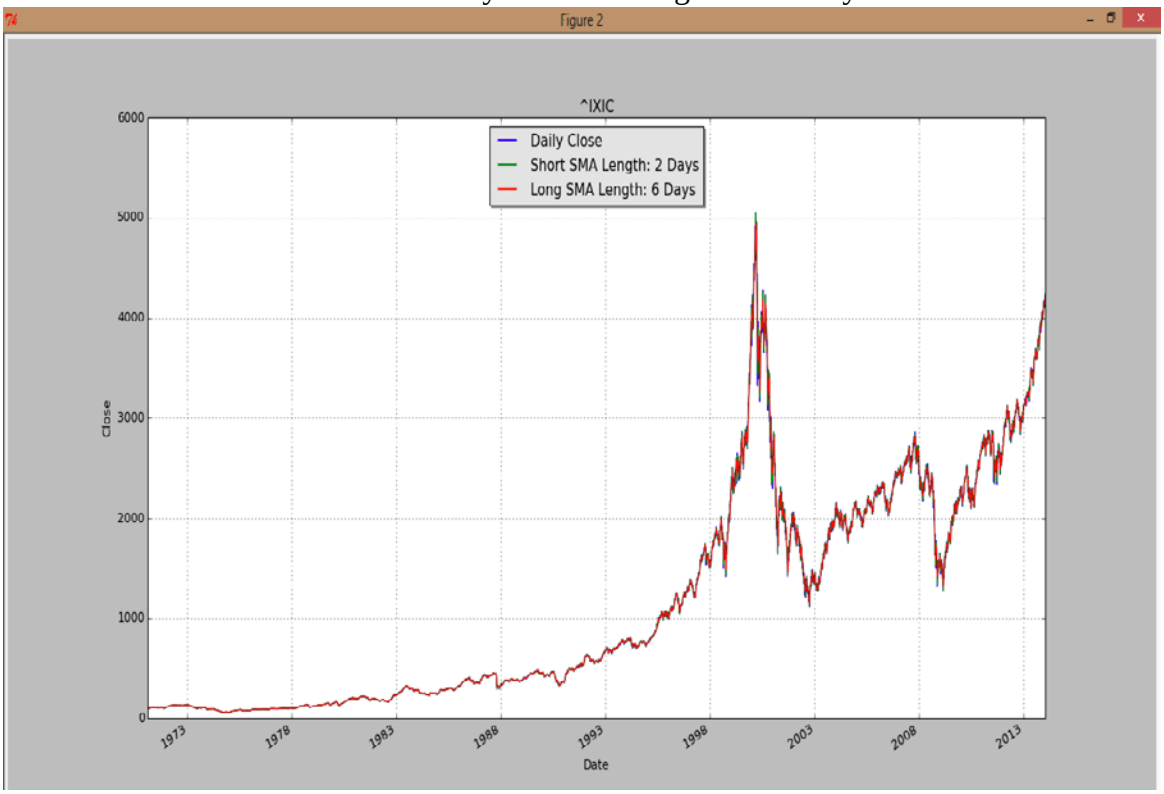

Figure 8 Best Crossover Output

5. Results

This PSO-optimized SMA crossover trading strategy was backtested on historical Dow Jones data. Using PSO, the best crossover from a generic bull-market (1949-1968) was determined. A simulation then traded the Dow based on that crossover, buying when the crossover occurred and selling 5 days later, for a subsequent bull market (1983-2000). Hence, the simulation traded just as a person in 1983 would, with no a priori knowledge of what was to occur. From 1983-2000 the SMA method returned about 14.9171% (after transaction costs), while simply buying the Dow in 1983 and holding it until 2000 with no intermediate transactions would have returned 894.4561 %. Thus, the SMA method's results were sub-par.

The same process was repeated for bear-markets. Using the best crossover from a generic bear-market (1940-1949), a simulation traded the Dow during two later bear-markets: 1969-1982 and 2001-2009. From 1969-1982 the SMA method lost about -13.518%, while a buy-and-hold strategy would have gained about 10.271%. The SMA method clearly fails here. However, from 2001-2009, when a buy-and-hold strategy would have lost about -15.209%, the SMA method actually made about 9.512%.

6. Conclusions

Thus, the SMA method produced below-market returns in the bull-market and one bear-market. Yet it beat the buy-and-hold strategy in another bear-market. Hence, in general a buy-and-hold strategy seems to work best. If one were to buy the Dow Jones right now, he would almost certainly see a sizable profit 10 or 20 years from now. Nevertheless, the SMA method does seem to have some application in long-term bear-markets, such as 2001-2009.

Further investigations might include developing a more complex trading strategy that either takes into account more variables, or uses a variable sell-point instead of just selling a fixed five days after the crossover. Such a trading strategy might produce better returns. But more research would be required to verify this hypothesis

References

Cerny, V. (1985). A thermodynamic approach to the traveling salesman problem: An efficient simulation. J. Optim. Theory AppL 45 41-51.

Eberhart R, Kennedy J. (1995). A New Optimizer Using Particle Swarm Theory. Proc of 6th International Symposium on Micro Machine and Human Science, Nagoya, Japan. IEEE Service Center Piscataway NJ,1995:39-43.

Holland, J. H. Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Ann Arbor, MI: University of Michigan Press, 1975.

KIRKPATRICK, S., GELETT, C. D. and VECCHI, M. P. (1983). Optimization by simulated annealing. Science 220 621-630.

J. A. Nelder and R. Mead, A simplex method for function minimization, Computer Journal 7 (1965), 308-313.

Storn, R. and Price, K. (1997), 'Differential Evolution - A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces', Journal of Global Optimization, 11, pp. 341–359.

Kennedy, J.; Eberhart, R., "Particle swarm optimization," *Neural Networks, 1995. Proceedings., IEEE International Conference on* , vol.4, no., pp.1942-1948 vol.4, Nov/Dec 1995.

Koza, John, Martin Keane and Matthew Streeter. "Evolving inventions." *Scientific American*, February 2003, p. 52-59

Nelder Mead Original ref
http://citeseerx.ist.psu.edu/viewdoc/download?rep=rep1&type=pdf&doi=10.1.1.120.6062

DJIA Data Source:
https://research.stlouisfed.org/fred2/series/DJIA/downloaddata?cid=32255/