Coevolutionary Algorithms and Pairs Trading

Raymond Luo, Zhiheng Wang

March 8th, 2021

1 Introduction

In the paper *A Game-Theoretical Approach for Designing Market Trading Strategies* [1], the authors suggest using co-evolve fuzzy trading rules to handle the partial form features. The results show that the co-evolutionary process creates trading rule-bases that produce positive returns. While the authors also suggest some possible improvements of their model, our team further investigates this problem by first recreating their program and then modifying the statistics. Our conclusion is that out algorithm is valid by not beating the Brownian Simulation while also producing optimized strategies.

2 Methods

While the original authors utilized co-evolutionary algorithms, and the LEAP library [2] provides various examples to implement co-evolutionary and island models, we chose to design our own classes to implement a singular evolution algorithm. The process of generating an optimized strategy with genetic algorithm follows the following steps:

1. Calculate related features day membership with fuzzification and defuzzification
2. Generate a population
3. Evolve the population with training environment (trading data) and mutation
4. Test resulted optimized strategies with testing environment

The below mentioned methods are originated from the original project [1], where the author provided detailed functions and possible parameters values. They first defined feature days and membership functions to predict trading day, then they briefly introduced their co-evolution algorithm. Our trading mechanics are in the same manner as the original project but our evolution algorithm are modified for better fitness judgement.

2.1 Feature days

The detecting of a signal trading days are decided by the prices of the prior day. The recorded price for each day are: open share price (O), the high (H), low (L), and the closing price (C). We have also added two more statistics based on the daily price: $Range = H - L$ and $RangeOC = O - C$. The definitions for trading days are shown below in equation 1.

$$O \leq L + 0.1(H - L) \tag{1}$$

$$C \geq H - 0.2(H - L) \tag{2}$$

Studies suggest that numerous features day can predict the occurrence of trend day. The index i to represent today is i=0 while previous days are indexed i=1,2,3... An example is that H[1] refers to the previous day's high.

- NRk

  NRk days are defined to represent volatility contraction, thus an NRk day will exist if the Range today R[0] is smaller than the ranges on each of the previous k-1 days. That is,

$$R[0] < min(R[1], \cdots, R[k-1]) \tag{3}$$

- DOJI

  DOJI days means temporary price indecision and indicates that O is within a percentage x of C on trading day. The predicate function DOJI is defined as,

$$DOJI(x) = \begin{cases} 1 & |O - C| \leq x \cdot (H - L) \\ 0 \text{ otherwise} \end{cases} \tag{4}$$

- Hook day

  Hook days represents a reverse direction in future prices, which occurs when O[0] (today's open price) is outside of R[-1] (previous day's range). Hook days are also divided into two groups. An up hook day is defined as when, $O[-1] < L[0] - \delta$. Similarly, an down hook day is defined as when $O[-1] > H[0] + \delta$. Usually, δ is with in the range of [0,10] and can be adjusted for use.

2.2 Membership functions

We have also used the same set of membership functions to describe the similarity of each day to the three types of feature day. Let $\mu(x)$ be the membership functions. Here are the membership function definition for each type of feature day.

- NRk

$$\mu_k(x) = \begin{cases} 0 & x < v_{min} \\ c(x - v_{min}) & v_{min} \leq x < v_{max} \\ 1 & x \geq v_{max} \end{cases} \tag{5}$$

  For c, v_{min}, and v_{max}, the original authors offered reference parameter values as shown in Table 2.2.

k	c	v_{min}	v_{max}
4	1/2	2	4
6	1/3	3	6
7	1/3	4	7

  Also, the above mentioned x is defined to be $x = D + \eta$ where $D \leq k$ is the number of days where the criteria for NRk holds for the previous days. Also, $\tilde{R} = \max_{1 \leq j \leq k} R[j]$.

$$\eta = \frac{\tilde{R} - R[0]}{\tilde{R}}$$

- DOJI

For DOJI membership, the function is shown below. The variable x refers to the percentage change between O and C while the parameter ρ refers to the threshold percentage. In addition, the parameter ρ is usually from the range of $[0.05, 0.30)$ [1].

$$\mu(x) = \begin{cases} 1 - x/\rho & 0 \leq x \leq \rho \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

- Hook day

  The membership function for Hook day are defined below in 7.

$$\mu(x) = \begin{cases} 0 & x < -\frac{1}{2} \\ 2(x + 0.5) & -\frac{1}{2} \leq x < 0 \\ 1 & x \geq 0 \end{cases} \tag{7}$$

  Since there are two types of hook days, the definition of x is defined differently in these two conditions. For up hook day, $x = L[0] - \delta - O[-1]$, while for down hook day, $x = O[-1] - \delta - H[0]$.

2.3 Fuzzy system

- Fuzzy rule selection - 5 features

  To perform fuzzy system, the authors first chose 5 different features: NR4, NR6, NR7, DOJI, Up Hook Day ($\delta = 0.5$) [1]. Our project utilized the same set of features and parameters the authors provided.

- Fuzzy numbers - Vector A

  A vector of A is created for every trading day, containing the membership value of each fuzzy rule. The vector of fuzzy numbers are shown below:

$$A = (\mu_{NR4}, \mu_{NR6}, \mu_{NR7}, \mu_{DOJI}, \mu_{hook})$$

  The vector A and all its relevant information is calculated outside of the evolution algorithm and passed into the algorithm in the form of a data frame time series along with the price for the corresponding day.

- Singleton fuzzifier - Vector Λ

  When given a fuzzy rule, each number in the Singleton fuzzifier represents the investor's probability of buying or selling the stock. The Singleton fuzzifier is defined as follows:

$$\Lambda = \{0.25, 0.5, 0.75, 1.0\}$$

- Fuzzy rule encode - Matrix M

  For every $\lambda_i \in \Lambda$ the column of M encodes a fuzzy rule-base. Similarly, for every fuzzy rule that we have picked, the row of M represents a rule. A typical M matrix would contain the same number of columns as elements in Λ and same number of rows as numbers of fuzzy rule.

- Fuzzy output - Vector B

  The fuzzy vector B is created by performing an algorithm between A and M which we will denote as:

$$A \circ M = B$$
$$\text{where } b_j = \max_{1 \leq i \leq 5} \min\{a_i, m_{ij}\}$$

- Crisp output - U

$$U = \frac{\sum_{i=1}^{N} b_i \cdot \lambda_i}{\sum_{i=1}^{N} \lambda_i} \tag{8}$$

The fuzzy output vector B is averaged to generate a single number U, which is the crisp output of the trading day. $U \in [0, 1]$ indicates the probability or likelihood of an up-trend day and the investor's desire to have trade action in the market.

- Trade strategies

To utilize the crisp output U, we designed two thresholds to generate a *signal* of whether the investor should take an action to buy, sell, or hold. Then, we designed a linear function of U scaling up to 20 stocks and the two thresholds to output the amount of stock the investor should be trading. We then adjust the position that we hold with the amount we have computed.

We can then proceed to pull the price data of the day. When the investor decides to buy for this trading day, the algorithm will use the opening price; on the other hand, if the investor decides to sell, the algorithm will use the closing price of the day. Our algorithm keeps two system of trade, the *debt* variable records the money transactions while the *result* variables records the PL change of each day. In addition for calculating fitness, we included upside and down side potential ratio as an intermediate fitness and sterling ratio as the final fitness.

2.4 Evolution Algorithm

The setup of our algorithm is based on the belief that the strategies of individuals in a firm can be represented by the genomes of a population in the evolution algorithm. We then observe the evolution of the strategies where newer strategies of a time period are modelled after the fittest strategies of the firm at that time and worse-performing strategies are replaced by better performing ones.

Our specific setup follows that of the LEAP general purpose Evolutionary Computation package in python. We encapsulate our model into four components: an individual class, a population class, a decoder class, and a problem class. We have designed our own individual and problem class and innovated the population class to better assist the population need for fitness score under our context. For decoder, we have chosen the LEAP Identity Decoder as it merely returns the original genome. The UML diagram for our solution is depicted below:

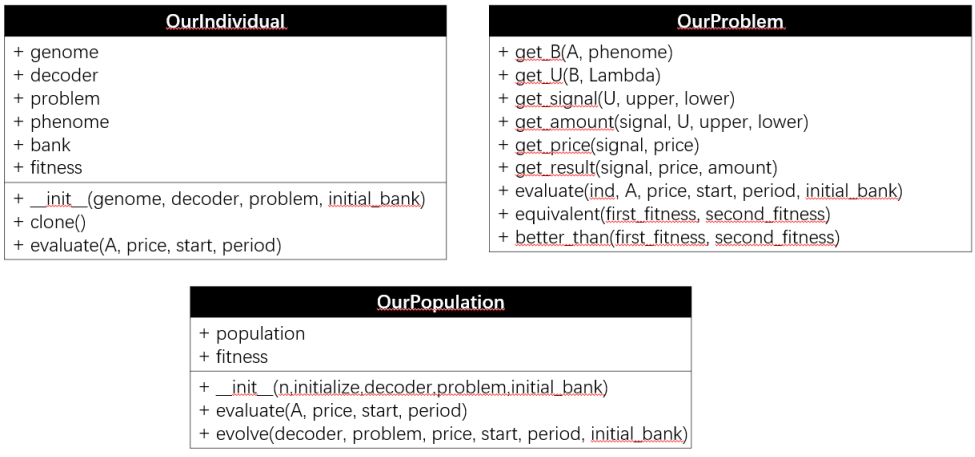

Figure 1: UML Diagram

The individual class represents individual objects each containing a genome, which encodes a matrix M representing the fuzzy rule-bases that the individuals uses to trade. Given observed features A on a

given day, this matrix M translates into a crisp output U which determines the individuals perception of the threshold for trading based on a number of feature Λ. The individual class can initialize new individual, clone itself, and evaluate itself.

Derived from the individual class, the population class contains a list of individuals and a list of their corresponding fitness score. The design of the population class aims to help evaluate and evolve the population at the same time since the evaluation involves trading data. In the original design, the valuation of the same population are done with the same trading start day and duration. Also, the evolve process involves mutation, crossover, and pooling, which can only be done on a population base and not a individual base.

Our decoder class utilizes the Identity decoder, implying there is no clear distinction between phenotype and genomes for our problem. The problem class describes how we evolve the individuals of our population, how mutations between parents and children during our evolution, and the definition of our fitness function.

Inside the problem class. At the end of every training period, we evaluate the best individuals by evaluating them by a fitness function f which denotes the performance of their strategy.
Specifically, in order to balance the consideration of expected gain, expected loss, and hit ratio, we considered the following two ratios:
The Sterling Ratio

$$SR = \frac{CompoundRateofReturn}{ABS(Avg.AnnualDD - R_f)}, \text{where } R_f \text{ is the risk-free rate}$$

The Upside Potential Ratio

$$U = \frac{\mathbb{E}[(R_r - R_{min})_+]}{\sqrt{\mathbb{E}[(R_r - R_{min})_-^2]}}$$

where $(X)_+$ represents

$$X_+ = \begin{cases} X, \text{if } X \geq 0 \\ 0, \text{otherwise} \end{cases}$$

$$X_- = -(X_+)$$

Frequency was fixed to be daily in accordance to our data while holding positions were allowed.

After taking the two best performing individuals of population size n by metric f, we created n offspring from these two individuals as parents. We used their genomes M and did one point crossovers on the matching vector columns of the parents. To incorporate mutation, we added Gaussian mutations to each scalar value of the resulting child matrix M' with standard deviation 0.1 and mean 0. During this mutation, we maintained that all scalar values had to be bounded between 0 and 1.

We then evaluate the performance of these new individuals' strategies over that same period to order the $2n$ individuals (of the old and new population) to decide which n individuals will be going forwards to the next training period. In a sense, the new children introduced act as "interns" of the firm, with strategies adapted from more recently high performing individuals and will help replace poor performing individuals in the firm.

3 Experimental Results

The evaluation of f involved taking an individual's genome M and translating it into a strategy according to the fuzzy rules which were used to determine up-trend days and down-trend days for trading. In this process we kept track of the cumulative P&L of the individual by their stock position during this period and their daily change in PL by the difference in their stock and money positions.

As in Greenwood [1], we hold that in each generation there are 40 individuals and that there are 40 new offspring at the end of each generation. In our case, these individuals are modelled after the two best performing individuals of the previous generation. By observing the performance of the 80 individuals over the most recent time period, we sort them according to the Sterling Ratio and eliminate the 40 worst performing individuals. Each generation lasts 150 trading days and we run the algorithm across 100 generations. We specifically observed AMZN trading strategies from 1997 to 2021. To reduce bias and over-fitting, we

reserved the last 2000 trading days of this data for testing and randomly chose a sequence of 150 days from the training set for each generation. For testing, we chose a random sequence of 200 days from the reserved test set to see how our trading strategies performed in different time periods. The specific statistics we looked into for our evaluation were the hit ratio, the expected gain, the expected loss, the Sterling ratio, and the Upside potential ratio.

Over our test set, formed from random selection of 200 consecutive days from the reserved last 2000 days of the our data set, we have the following result:

```
Test attempt:  1
Hit Ratio:  137 : 63
Expected Gain: 17.96
Expected Loss: -0.77
Sterling Ratio: -224245.36
Upside Potential Ratio: 21.27
Test attempt:  2
Hit Ratio:  138 : 62
Expected Gain: 11.23
Expected Loss: -1.32
Sterling Ratio: -60699.19
Upside Potential Ratio: 6.19
Test attempt:  3
Hit Ratio:  149 : 51
Expected Gain: 33.66
Expected Loss: -0.90
Sterling Ratio: -41434.40
Upside Potential Ratio: 30.17
Test attempt:  4
Hit Ratio:  140 : 60
Expected Gain: 20.36
Expected Loss: -0.79
Sterling Ratio: 3151.84
Upside Potential Ratio: 22.84
Test attempt:  5
Hit Ratio:  145 : 55
Expected Gain: 30.63
Expected Loss: -0.86
Sterling Ratio: -48568.93
Upside Potential Ratio: 34.10
```

Figure 2: Results of running strategy on Test Set

We see that as our training is very dependent on the fitness of individuals over a time period, the test set results provide very varied sterling ratios and performances as expected. However, we note that the hit ratio is almost consistent across the board.

A common caution amongst backtesting trading strategies is the possibility of bias within the model, whether it be accidentally reading in signals too early or over-fitting the model. It is important to verify that the prediction of upward trends in our results were attributed to the strong upward trends of the AMZN stock and not from the setup of our algorithm. To that end, we ran our algorithm on brownian motion to make sure that on a random trend that there would not be an obvious winning strategy. The brownian motion should be producing results where our algorithm could not produce much success as the information of the market is random. We constructed our brownian motion by adding identically distributed normal increments with mean 0 and standard deviation in what would be each hour of real life.

To match the setup of the financial markets, we took the 8th and 16th hr of each 24 hr period from this construction to be the opening and closing price with the high and low prices matching the highest and lowest points in this 8 hr window. With this construction, we created results that verified that our model performed as expected in a random setting:

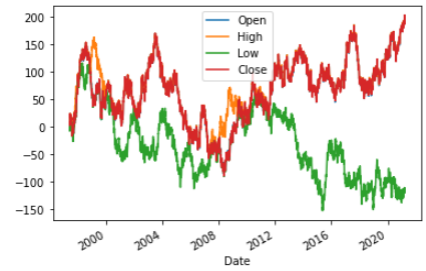

Figure 3: Brownian Motion construction

Figure 4: Statistics of the Brownian Motion

```
Test attempt:  1
Hit Ratio:  112 : 88
Expected Gain: 2.38
Expected Loss: -0.67
Sterling Ratio: 1602.62
Upside Potential Ratio: 1.31
Test attempt:  2
Hit Ratio:  102 : 98
Expected Gain: 0.49
Expected Loss: -320.99
Sterling Ratio: -0.17
Upside Potential Ratio: 0.00
Test attempt:  3
Hit Ratio:  108 : 92
Expected Gain: 0.59
Expected Loss: -1.12
Sterling Ratio: -227.00
Upside Potential Ratio: 0.12
Test attempt:  4
Hit Ratio:  105 : 95
Expected Gain: 1.56
Expected Loss: -2.66
Sterling Ratio: -84.48
Upside Potential Ratio: 0.10
Test attempt:  5
Hit Ratio:  102 : 98
Expected Gain: 0.79
Expected Loss: -1.27
Sterling Ratio: -379.27
Upside Potential Ratio: 0.14
```

Figure 5: Test Results of Algorithm on Brownian Motion

4 Conclusion

We designed a genetic evolution algorithm basing on the Greenwood paper [1] to optimize trading strategies for a certain stock. We implement our algorithm with reference to the LEAP python library [2] and modify in accordance to our personal setups. The evolutionary algorithm, as suggested in the Greenwood paper, provides a model of producing strategies in an inter-firm setting as a zero sum game. Our implementation removes the tournament and inter-firm setting while focusing more on modifying the fitness functions to better represent strategies evaluated on expected gain, expected loss, and hit ratios. Here, we thought it would be more realistic to consider that as evolutionary algorithms are very much dependent on the epochs which the population is trained, as much consideration should be made towards drawdowns as well as net returns. Additionally, to verify that our model is not over-fitted to a specific regime, a separation of training and testing periods, from which training and testing sets were randomly sampled, were made. A further comparison involved observing the model's behavior across brownian motion simulations. While the evolutionary algorithm provided a wide array of performances given the different regimes of testing periods, they generally showed a consistent heat ratio and a solid expected gain.

References

[1] Garrison W. Greenwood, Tymerski Richard. (2008). A Game-Theoretical Approach for Designing Market Trading Strategies. Symposium on Computational Intelligence and Games

[2] Mark Coletti, Eric Scott, Jeff Bassett, *"LEAP general purpose Evolutionary Computation package"*, Github, https://pypi.org/project/leap-ec/, (Sep. 2019); Jan. 30th, 2021