Building a Stock Market Trading Agent with Genetic Algorithm

YiFan Li
ZeYuan Yang

December 15, 2019

Abstract

The stock market is an essential part of the financial market. Countless investors attempt to predict the price trend of stocks so that they can make the right buying and selling decisions to gain revenue. Defining what are the most suitable trading rules is a controversial problem in the field of investment. Genetic algorithm is one of the existing approaches to find the best combination from numerous accepted trading rules. In our paper, we adopt genetic algorithm with different parameters on 5 stocks to take the average result. The experiment is run on three years' historical data, separated into a train set and test set. Also, we involve two variants of genetic algorithm in our experiment, which are representations of gene and elitism, rather than the basic genetic algorithm. We compare the results among each single chosen parameter. Four metrics are used to analyze the performance, which are the best performance and the population average, of both two datasets. Finally, we discuss the conclusion of our experiment and what other improvements could be done to find a better combination.

1 Introduction

Predicting the future can only happen in science fictions, but predicting the future trend of some certain time series is feasible. Time series analysis has a long history, and multiple approaches have been studied.

The most traditional methods are statistical ones such as the well-known ARIMA model. Although the model is simple, it is still used in recent researches such as [?]. Some delicately designed machine learning approaches are also introduced in these years, varying from relatively simple support vector machine [?] to complex neural network [?].

The one that receives the most attention among all time series is probably the stock price. Firstly, as an essential part of the financial market, the stock market is the apple of investors' eyes. Not only financial researchers but also individuals and securities companies have tried their best to analyze the stock price time series and predict its behavior in the future, to make a fortune. Secondly, unlike most time series prediction tasks where the output must be exact future values, only knowing whether the future price will go up or down is already sufficient to make a trading decision. Therefore, the problem can be simplified thus enabling more ideas and approaches to be applicable. Thirdly, sufficient financial indexes and trading rules can serve as domain knowledge to be combined with computer algorithms.

Nowadays, there are numerous financial indexes, and over two hundred different trading rules were built upon them to suggest buying, selling or holding decisions, based on historical stock price time series [?]. Each rule has its rationality and will indicate some features of the target stock. However, the movement of a stock price is the combined result of a massive number of microeconomic and macroeconomic factors, which makes it impossible to predict the trend with any index or rule solely. As Korczak [?] pointed out, there is no rule consistently outperforming the others. Therefore, finding the most appropriate way to utilize them becomes the key to the problem.

Although some scholars argue that stock prices follow a random walk pattern, most experts in this field believe that the prices can be predicted with more than 50 percent accuracy, with eligible trading rules and reasonable combination [?]. Plenty of approaches has been examined and we will go through some of them in the literature review. Noticeably, the genetic algorithm is widely used in related researches.

Genetic algorithm is an adaptive heuristic search method, inspired by the natural process of genetic evolution. It can quickly generate high-quality solutions to optimization problems, such as scheduling and shortest path, and modeling system where randomness is involved, such as the stock market, by simulating the selection, crossover and mutation procedures in nature [?].

In our paper, we will apply the genetic algorithm to find the best combination of a set of trading rules, which can help with the buy and sell decisions. We will dig deeper into a more comprehensive and applicable solution based on the previous works.

2 Related Work

We will go through related works about the genetic algorithms first, then look into its application on the stock time series prediction.

2.1 Genetic Algorithm

The genetic algorithm was invented in the early 1970s by John Holland [?]. It is based on the mechanics of natural genetic selection. To conduct GA, a "chromosome" must be defined to represent a solution to the problem. A chromosome is a sequence of alleles, such as a bit. The genetic features of such representation enable crossover and mutation, in order to exchange information with each other [?]. A fitness function, which can evaluate the solution, has to be defined as well. It will be used to select those chromosomes that survive.

The general steps of the genetic algorithm are as follows:

1. Initialization. Some chromosomes are randomly generated to form a population.
2. Selection. During each generation, a proportion of the existing population will be selected through a process, where fitter solutions, measured by the fitness function, are more likely to be selected. They are used to generate the population for the next generation.
3. Reproduction. A second generation population is bred by the previous generation. There are typically two genetic operators involved in this step:
   (a) Crossover. Parent chromosomes are selected from the survivors and their genes are combined in some certain way to create a new chromosome.
   (b) Mutation. Some genes in some chromosomes are randomly chosen and changed.

   Generally, the population size is preserved throughout each generation, so the selection and reproduction process should counterbalance each other's effect on the number of chromosomes.
4. Termination. The selection and reproduction are repeated several times until a termination condition has been reached. Common termination conditions can be:
   (a) A solution is found which satisfies a minimum fitness criteria
   (b) A fixed number of iteration is reached
   (c) The fitness value of the best chromosomes in several successive generations is not improving

As the genetic algorithm keeps developing, several variants are proposed. The first one is the chromosome representation. Traditionally, the chromosome consists of a series of binary bits. However, in some problem settings, the gene can be a floating-point number [?], or even complex data structures like list or map [?]. The second one is elitism [?], which requires the selection and reproduction to be carefully designed so that the several best chromosome must be preserved to the next generation. The third one is the adaptive genetic algorithm [?]. Unlike the traditional ones whose strategy of selection and reproduction is constant, the possibility of selection, crossover, and mutation will vary in adaptive GA. The adjustment of these possibilities is related to the current

fitness value. All these variants improved some aspect of the genetic algorithm, and we will implement some of them in our experiment.

2.2 Application of GA to Stock Price Prediction

Given the stock price time series as the original data, Stephen Leung et al. [?] modeled it as a fuzzy time series by converting the percentage of the price change into a fuzzy value. The fuzzy logic relationship is extracted and classified to form fuzzy logic relationship groups (FLRGs). FLRGs will then be normalized to get a weight matrix. Each gene in the chromosome corresponds to a fuzzy value and the whole chromosome will output a prediction. The fitness function used in the genetic algorithm is:

$$\text{RMSE} = \sqrt{\frac{\sum_{t=1}^{n} \left(\text{actual}(t) - \text{predicted}(t)\right)^2}{n}}$$

One technique in their approach is worth mentioning. The tournament method is used to select survivors in the previous generation. Several "tournaments" among a few chromosomes chosen at random from the population are conducted. The winner of each tournament (the one with the best fitness) is selected for crossover. Comparing with the traditional approach of directly selecting the good ones from the whole population, it can improve gene diversity thus prevent GA from being trapped in local optimum [?].

Applying the fuzzy logic as well, Kuo, Ren Jie and his team [?] further choose to combine genetic and neural networks. They developed a stock trading system, using GFNN (genetic-algorithm-based fuzzy neural networks) to build the qualitative model. The basic model is an FNN (fuzzy neural network), and the genetic algorithm is used to provide the initial weights and select hyperparameters. This research and some similar ones utilized the optimization feature of the genetic algorithm, but put more emphasis on other techniques.

In addition to those combining multiple methodologies to build a complex model without the help of domain knowledge, more studies use only GA as the algorithm and take the financial indexes and trading rules into account. They no longer produce exact value for future time series but provide trading decisions. As we mentioned before, there are numerous financial indexes, each summarizing one feature of the stock price time series. They can be used to build trading rules, which will indicate a decision for buying, selling or holding. It is the strategy to utilize them that is optimized in those researches. However, the way they model the problem setting may be slightly different.

Lin Li et al. [?] arbitrarily picked one trading rule, "the filter rule", and optimize the choice of its parameters. The fitness function is the profit of trading using its decision. Comparing with brute force grid search, genetic algorithm saved a lot of running time, but loses very little precision. Nevertheless, only using one trading rule limits the model's performance, hence the profit is still significantly lower than that of a human trader. Most studies are focused on the way of combining multiple financial indexes.

Fuentu et al. [?] selected three financial indexes, RSI, MACD, and Stochastic Index, and five of their variants are coded in the algorithm. The goal is to find the best time to buy stock, indicated by the values of the five index variants. Each gene in the chromosome represents an interval of the value of a financial index, and the fitness function is calculated as below:

1. If there is a period that could match all the financial indexes represented by the chromosome, the chromosome's fitness value is the profit of making the buying decision at the start of that period.
2. If there is no such period, the chromosome will not be selected into the next generation.

In the future, when the five financial indexes have the value represented by the best chromosome, it can be concluded to be the best time to buy a stock.

In some other researches [?] [?] , the suggestions of trading rules are integrated to give a final trading decision and such integration is optimized with the genetic algorithm.

Firstly, a set of representative trading rules are selected to examine, such as Moving Average and Relative Strength Index. Each gene in the chromosome corresponds to one rule, with only 1 and 0 as their possible

values representing whether the rule should be considered in the final decision. The final decision is the voting result of all the rules considered. On each day in the stock time series, the chromosome will give such a final decision, and the fitness function is the profit made by these decisions. After that, a general genetic algorithm can be applied to find the best chromosome.

This approach is intuitive and relatively easy to implement. Rohit and Kumkum [**?**] take advantage of it and further uses support vector machine to determine the weight of each rule in the final decision.

Two concerns about this approach are raised, though. Firstly, the previous experiments are always run on one stock. Whether this methodology is suitable for more stocks in the market remains a problem [**?**]. Secondly, whether the resulting chromosome, i.e. combination of trading rules, remains useful as time goes by is unknown. Some experts argue that the influence of time is not obvious, but it still requires examination [**?**]. Our work will be based on the approach introduced above, try to address these two issues and hopefully find other insights.

3 Approach

3.1 Problem Setting Specification

Similar to [**?**] and [**?**], we will setup our problem setting as follows:

- Given a historical stock price time series in the past n periods, $P_t = [p_{t-n}, p_{t-n+1}, \cdots, p_{t-1}]$, where t represents the present,
- and a set of k trading rules suggesting "buy", "sell" or "hold" action based on the time series $R = \{r_1, r_2, \cdots, r_k\}, \forall r_i(P_t) \to s_i \in \{\text{sell}, \text{buy}, \text{hold}\}$,
- design an agent A who can integrate all the rules to make a final decision $A(R(P_t)) \to d_t \in \{\text{sell}, \text{buy}, \text{hold}\}$,
- that can generate profit in the following m period, which can be calculated as $\sum_{t=0}^{m} p_t^{\text{sell}} - p_t^{\text{buy}}$,

$$p_t^{\text{sell}} = \begin{cases} p_t & \text{if} A(R(P_t)) = \text{sell} \wedge A \text{ currently holds the stock} \\ 0 & \text{otherwise} \end{cases},$$

$$p_t^{\text{buy}} = \begin{cases} p_t & \text{if} A(R(P_t)) = \text{buy} \wedge A \text{ currently not holds the stock} \\ 0 & \text{otherwise} \end{cases}$$

As examining only one stock may be not representative enough, we will conduct the experiment on five stocks and take their average result. Additionally, because the maximum possible benefit varies for different stock price time series, it is unwise to compare the absolute value of profit directly. The benchmark of our evaluation is an agent who can "predict the future". It knows the whole time series and make decision based on it, $A'(P_m, t) \to d'_t$, which can be implemented with the following trivial algorithm.

Algorithm 1: Benchmark Agent

```
Data: prices
Data: today
Result: decision
if prices[today + 1] > prices[today] then
    return buy;
else if prices[today + 1] < prices[today] then
    return sell;
else
    return hold;
end
```

In all, the evaluation of the agent will be $\frac{\sum_{i=1}^{5} \frac{\text{Profit}_i(\text{genetic agent})}{\text{Profit}_i(\text{benchmark agent})}}{5}$, where i represents different stocks,

and the goal is to maximize it.

3.2 Genetic Algorithm

3.2.1 Chromosome Representation

In our experiment, each gene in the agent's chromosome will be a number [1] corresponding to one trading rule. It will be used as the weight to take average on the trading rules' decision. The decisions can be considered as:

- Buy $\rightarrow$ 1
- Sell $\rightarrow$ -1
- Hold $\rightarrow$ 0

After that, a positive weighted average will result in a Buy decision, while a negative one will lead to a Sell decision outputted by the agent.

3.2.2 Genetic Operators

Various implementations for genetic algorithm operators are proposed in previous works. For example, there are trivial highest-score and tournament method[**?**] selection, single-point, k-point and uniform crossover[**?**], etc. Therefore we considered it necessary to describe our implementation of the genetic operators explicitly.

We applied the basic highest-score selection, which selects a percentage of agents with the highest score to go to the reproduction process.

In the crossover phase, a percentage of agents who passed the selection will be chosen. They will act as parents, producing new agents, but themselves will not go to the next generation. The children and those who are not selected as parents will go to the mutation phase. When generating new agent, uniform crossover will run repeatedly, until the total number of agent is the same as original. Uniform crossover, to be more specific, means that each gene in the children's chromosome is chosen from either parent with equal probability.

During mutation, a percentage of the agents will be chosen to mutate, the others will remain unchanged. Each gene in the chromosome of mutating agents will have a 50% chance to be reset as random one.

3.2.3 Variants

Chromosome representation[**?**] is applied in our experiment. Besides the bit representation used in [**?**] and [**?**] where all the genes are either 0 or 1, they can also be:

- A floating number ranging from 0 to 1
- A tuple, whose first element is a floating number and second element is a list, containing the parameters for the financial rule

As mentioned before, the genes will be used as weights to take average on the rules output. The bit, the float and the first element in the tuple, respectively, will be of this usage. The second element of the tuple representation is used to configure the financial rules, who actually have some adjustable parameters, but set to a default value based on financial researches in the bit and float representation. The more complex the chromosome representation is, the more flexible it can integrate all information. However, the searching space will be larger.

Elitism is also implemented in the experiment. Note that in the basic procedures described above, it is possible that the best agent is selected as parent therefore will not proceed to the next phase, or its gene could be changed in the mutation phase. If a rate of elitism is specified, a copy of that percentage of the best agents will be directly used as the agents for the next generation. The numbers of agents generated by crossover will correspondingly adjust so that the total amount of agent stays constant between each generation. With this technique, it can be guaranteed that the best agent will only improve as it evolves.

[1] In the complex representation variant, a part of the gene will be a number. See section 3.2.3 for detail.

3.3 Experiment Plan

The possible configurations in our experiment are:

- survival rate
- crossover rate
- mutation rate
- elitism rate
- chromosome representation

We will try different combination of these parameters/choices and compare their results. Some fixed hyper-parameters are:

- population: 15
- total training epoch: 8

We will split the data into two parts, from 2016–03–01~2017–12–29 as the train dataset, and use the data from 2018–01–02~2018–12–28 as the test dataset. The genetic optimization will only run on the train set, and the final population will be applied on the test set to evaluate their performance. This idea is borrowed from machine learning, that a model with high performance may not be accurate on unseen data. We can test the final agent's ability to generalize with this approach.

3.4 Other Technical Details

3.4.1 Data Preparation

The open, high, low, close price and trading volume of the five randomly chosen stocks, CMS, MKC, MMP, THS and XEL, between 2016–01–04 and 2018–12–31 are pulled from Alpha Vantage. They will be used as our dataset. The data quality is good, therefore no further preprocessing is required.

3.4.2 Programming Language and Packages

We chose Python as the programming language it is simple. Its Pandas and Numpy packages also significantly simplify array and matrix computation. There are also various modules providing fundamental framework for genetic algorithm, however, we decide not to use them and build the genetic algorithm related modules from scratch, because we want to know the exact details of every single line of code in the genetic algorithm.

3.4.3 Trading Rules Specification

As the financial trading rules are verbose to explain and the details are actually off-topic, we will specify them in section 6, appendix.

4 Results and Analysis

We tracked the performance of the best agent and the population average of each generation in the training phase. The best and average performance of the final generation in the testing phase are also recorded. In the following several sections, we will interpret the effect of the genetic operators and variants by looking at the figures which depicts the training process and the testing result. The lines in the figures are explained as follows:

- Blue line: best agent in each generation (epoch), performance on train set
- Orange line: population average in each generation (epoch), performance on train set
- Green dashed line: best agent in the final population, performance on test set
- Orange dashed line: average of the final population, performance on train set

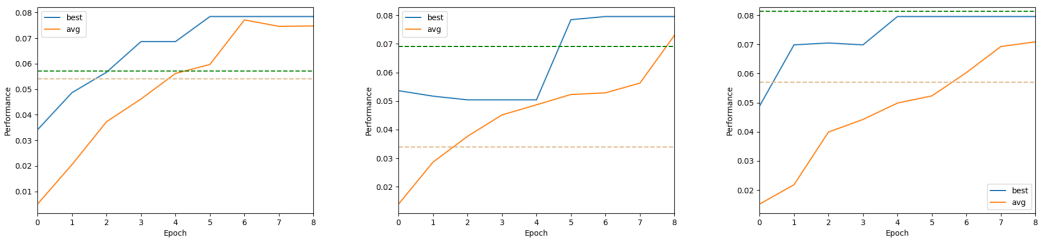
Although all the information is demonstrated, we would like to focus on the training performance in the following sections first. Comments on the testing performance will be made in section 4.6.

4.1 Survival rate

Survival rate is the percentage of well-performed agents who can go to the reproduction procedures. The effect of survival rate is analyzed in this section by controlling the following parameters:

- crossover rate: 0.75
- mutation rate: 0.1
- elitism: off

The following figure shows the experiment result of applying different survival rate in the bit representation.

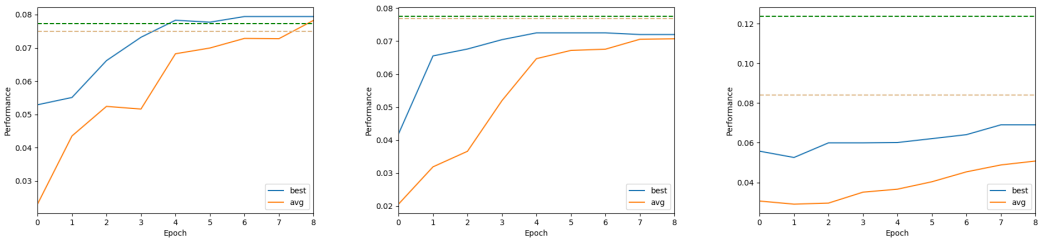


(a) Survival rate: 0.4 (b) Survival rate: 0.5 (c) Survival rate: 0.6

Figure 1: Bit representation

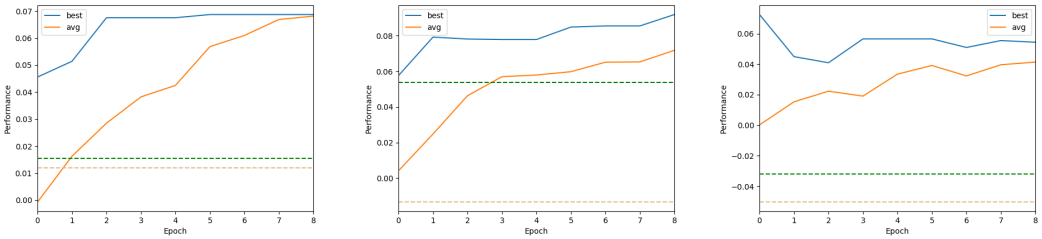
All the best performers reached a performance of about 0.08 in about 5 generations. The population average converge to the best agent gradually and is fairly close in the final generation. Such convergence happens somewhat faster when survival rate is 0.4 but it is not significant. In all, no prominent difference can be identified in the figures.

However, the situation differs a lot in the float and tuple representation, as shown below.



(a) Survival rate: 0.4 (b) Survival rate: 0.5 (c) Survival rate: 0.6

Figure 2: Float representation



(a) Survival rate: 0.4 (b) Survival rate: 0.5 (c) Survival rate: 0.6

Figure 3: Tuple representation

When survival rate is 0.4, the population average converges to the best agent, and is almost the same in the final population. Such convergence means that the genetic diversity of the population is really low, all agents are likely to have highly similar chromosomes. However, when survival rate is 0.5, the tuple representation, maintains a good gene diversity even in the final epoch. The same goes for a 0.6 survival rate for both float and tuple representation.

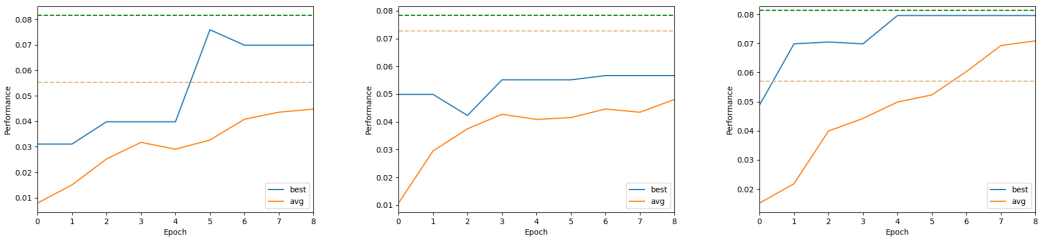
Although a higher survival rate improves the diversity of the gene pool, it set back the efficiency of optimization. When survival rate is 0.6 in float representation, hardly does the best agent perform better. The best agent in tuple representation even perform worse than the initial best, making the whole genetic algorithm approach almost meaningless.

4.2 Crossover Rate

Crossover rate is the percentage of agents who are used to generate new agents. The effect of crossover rate is analyzed in this section by controlling the following parameters:

- survival rate: 0.6
- mutation rate: 0.1
- elitism: off

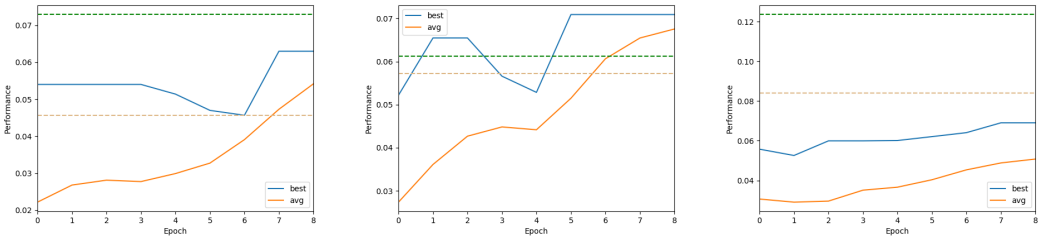
The following figure demonstrate the experiment results using the bit chromosome representation.



(a) Crossover rate: 0.25 (b) Crossover rate: 0.5 (c) Crossover rate: 0.75

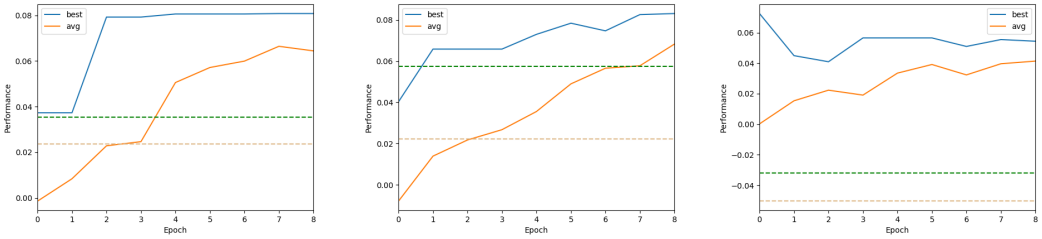
Figure 4: Bit representation

All the average performance grows fast and smoothly, but there are a lot of horizontal lines in the performance of the best agents. Changing the crossover rate does not help changing this situation in bit representation, but it makes a slight difference in float and tuple representation as shown by the following figures.



(a) Crossover rate: 0.25 (b) Crossover rate: 0.5 (c) Crossover rate: 0.75

Figure 5: Float representation



(a) Crossover rate: 0.25 (b) Crossover rate: 0.5 (c) Crossover rate: 0.75

Figure 6: Tuple representation

When the crossover rate is low, there are a lot of horizontal lines for the best agent as well. However, with a crossover rate of 0.75, the best agent gets updated in almost every generations, even though the improvement is not significant, if not deterioration. A high crossover rate also prevent the population average from improving fast.

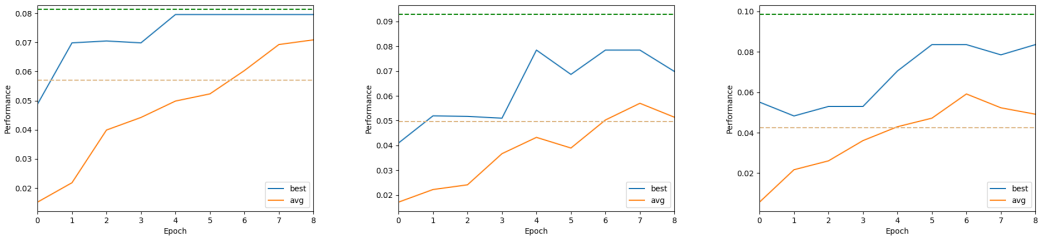
In all, crossover rate does not have a clear pattern of affecting the genetic optimization in our problem. It helps ensure the updating of the best agent in each generation, but whether such update is positive or negative cannot be guaranteed.

4.3 Mutation Rate

Mutation rate is the percentage of agents whose genes will be randomly modified. The effect of mutation rate is analyzed in this section by controlling the following parameters:

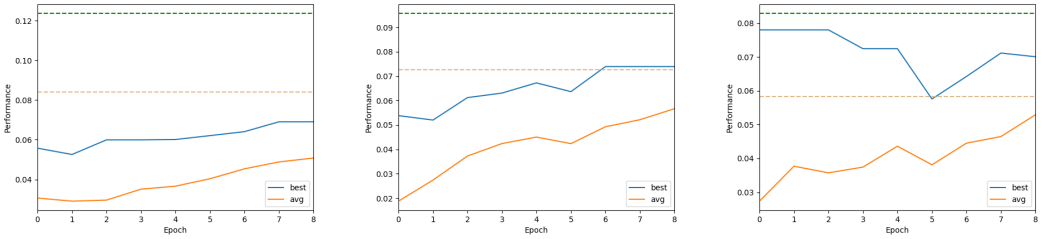
- survival rate: 0.6
- crossover rate: 0.75
- elitism: off

The following figure demonstrate the experiment results using the bit and float chromosome representation.



(a) Mutation rate: 0.1 (b) Mutation rate: 0.2 (c) Mutation rate: 0.3

Figure 7: Bit representation

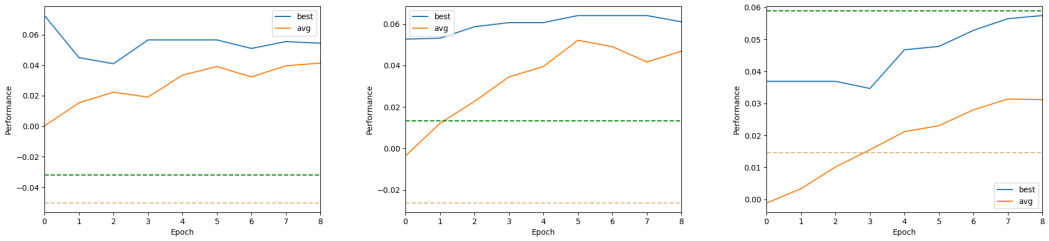


(a) Mutation rate: 0.1 (b) Mutation rate: 0.2 (c) Mutation rate: 0.3

Figure 8: Float representation

When the mutation rate is low, the best performance and the average performance of both representations are all monotonically increasing. However, when the mutation rate is higher, the performance starts to fluctuate. A mutation rate of 0.3 in float representation even cause the best agent's performance to deteriorate.

Such pattern of fluctuation seems to be reversed in the tuple representation, see the figure below.



(a) Mutation rate: 0.1 (b) Mutation rate: 0.2 (c) Mutation rate: 0.3

Figure 9: Tuple representation

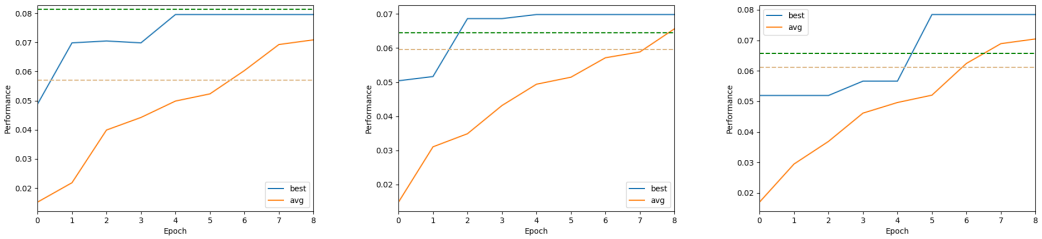
With a low mutation rate, the fluctuation is more severe than that with a high mutation rate. It can be observed that with a mutation rate of 0.3, the average of population performance has a stable monotonic increasing trend.

4.4 Elitism

Elitism ensures a percentage of best agents go to the next generation unmodified. The effect of elitism is analyzed in this section by controlling the following parameters:

- survival rate: 0.6
- crossover rate: 0.75
- mutation rate: 0.1

The following figure demonstrates the experiment results using the bit representation.

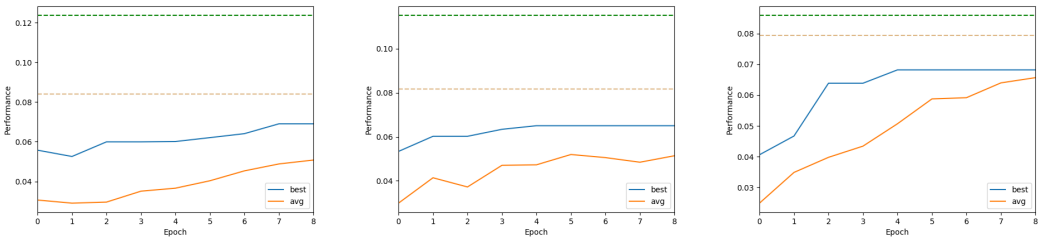


(a) Elitism rate: 0 (b) Elitism rate: 0.1 (c) Elitism rate: 0.2

Figure 10: Bit representation

There is no significant difference between different elitism rate. Even without the elitism, the best/average performance are both monotonically increasing. And in the final generation, the performance of best agent and population average are fairly close.

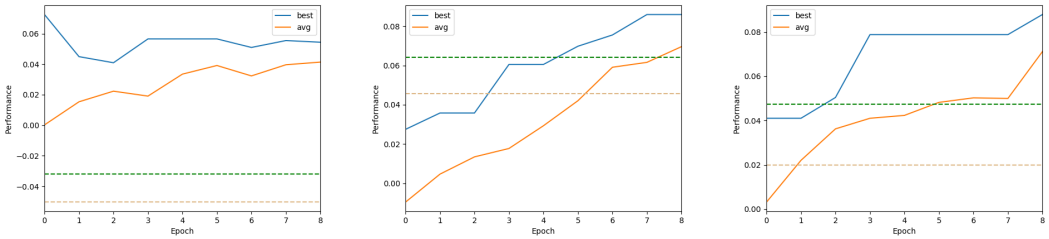
The following figure demonstrates the result with the float representation.



(a) Elitism rate: 0 (b) Elitism rate: 0.1 (c) Elitism rate: 0.2

Figure 11: Float representation

With a high elitism rate of 0.2, the average performance quickly converges to the best performance. It shows that elitism could decrease the gene diversity. However, this pattern is not observed in the tuple representation.



(a) Elitism rate: 0 (b) Elitism rate: 0.1 (c) Elitism rate: 0.2

Figure 12: Tuple representation

The divergence of population average and best agent's performance seems to be not accelerated by the increasing elitism rate. On the other hand, the fluctuation of performance without elitism no longer exists with elitism adopted, which is a trivial result of the concept of elitism itself.

4.5 Chromosome Representation

The figures listed in the previous sections can be used to compare the three chromosome representations we implemented. Therefore, we will not display them here again.

The bit representation has only a limited possible genes. The search space of float representation, however, is infinite. The tuple representation is even larger than that of float representation, and it dropped the support of the domain knowledge. The the underlying rules no longer use the generally accepted parameters, instead, the genetic algorithm is in charge of optimizing them so that they can ideally fit the current stock price series even better.

Such difference is the reason why the same genetic algorithm parameters show different patterns. With a limited search space, the bit representation itself tends to have a undiversified gene pool. Smaller survival rate and larger mutation rate will help introducing diversity in the gene pool, hence improve the overall performance. On the contrary, the float and tuple representation are more unstable. Excessively aggressive crossover and mutation will deteriorate the optimization by disrupting the survival of high quality genes. A moderate elitism rate, on the other hand, will protect the good genes therefore improving the result of genetic algorithm. The 0.1 elitism rate of tuple representation most clearly demonstrate that.

Another interesting phenomenon of different representations is the final training best and average. The following table listed the three configurations that results in the best agents, and the three configurations that results in the worst agents in the training phase.

Representation	Survival	Crossover	Mutation	Elitism	Training Best	Training Avg
tuple	0.5	0.75	0.1		0.09	0.07
tuple	0.6	0.75	0.1	0.2	0.088	0.072
tuple	0.6	0.75	0.1	0.1	0.086	0.069
bit	0.6	0.5	0.1		0.057	0.048
tuple	0.6	0.75	0.3		0.057	0.031
tuple	0.6	0.75	0.1		0.052	0.04

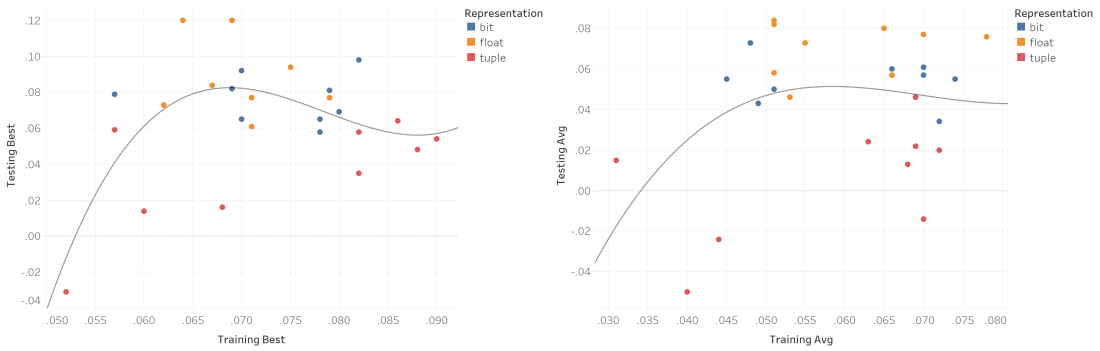
Table 1: Top and Bottom 3 Configurations for the Training Best

Interestingly, tuple chromosome dominates the top three, while two of the bottom three are also tuple chromosomes. As mentioned before, the tuple representation has the most flexibility for fitting the characteristic of

a certain stock price time series. On the other hand, if it stuck in a local optimum, or the gene pool is disrupted by the aggressive mutation, or not enough number of epochs are conducted, the flawed trading rule parameters will prevent the rules from outputting an effective decision. Therefore the performance of the agent is certainly limited.

4.6 Generalization

In the previous figures, the testing performance in each experiment is also depicted. The relationship between them and the training performance can be more clearly identified by plotting them on a scatter plot.



(a) Training and Testing Best

(b) Training and Testing Average

Figure 13: Training and Testing Results

Both best and average measurements show such a pattern, that as the training performance improves, the testing performance will increase firstly then decrease. It corresponds to the underfitting–optimum–overfitting routine in a lot of machine learning problems. Blindly pursue a better performance on the train set does not necessarily leads to a higher profit in the future. Finding appropriate configuration for the genetic algorithm is critical to building a successful trading agent that can be put into the real world.

The differences between the chromosome representations once again shows interesting pattern in this visualization. Those having the highest test scores are all float represented agents. The limitation of flexibility of bit representation constrains its performance so it can hardly be the best. On the other hand, although some tuple represented agents can have incredible training performance, they tend to perform poorly in the test. Its abandoning the benefit of domain knowledge and the overfitting leads to such result. The exact configuration of the top 3 testing results are listed in the following table.

Representation	Survival	Crossover	Mutation	Elitism	Testing Best	Testing Avg
float	0.6	0.75	0.1		0.12	0.084
float	0.6	0.75	0.1	0.1	0.12	0.082
bit	0.6	0.75	0.3		0.098	0.043

Table 2: Top 3 Configurations for the Testing Best

5 Conclusion

6 Appendix: Financial Rules

TODO: Financial rules