Optimization of a Statistical Arbitrage Strategy using the Genetic Algorithm

António Pedro de Oliveira Alcobia Vassalo Lourenço
a.pedro.lourenco@tecnico.ulisboa.pt

Instituto Superior Técnico, Lisboa, Portugal

September 2020

Abstract

This work explores the optimization of a proposed system, that generates trade signals based on statistical arbitrage principles, using machine learning techniques, in particular, the Genetic Algorithm. This work begins by reviewing the growing literature on statistical arbitrage origin, evolution, and strategies followed by a review in the Engler-Granger cointegration testing and a review on genetic algorithms. The proposed system is presented in the next chapter. The system uses historical market data from 2012 to 2018 to perform the backtests, with the optimized results generating an average return of 12% per anum and a Sharp Ratio of 1.82. The backtest returns were independent from the overall market direction given by the S&P 500 confirming the returns are uncorrelated with the overall market. In this work it is possible to conclude that the genetic algorithm can be used to optimize statistical arbitrage trading strategies.

Keywords: Genetic Algorithm, Statistical Arbitrage, Cointegration, Stocks

1. Introduction

Due to the strong correlation between a country's stock market and its economy and the potential financial opportunities it generates, predicting the price movements have become one of the main focuses on the interest in this area. Due to the nature of the markets, these are highly noisy systems under the influence of various forces, either economic, political, or natural disasters. This makes predicting price movements extremely hard.

Investors with large pools of capital face difficult challenges in deploying and allocating it across investments with uncorrelated risk since most products and investments available have some level of correlation to the economy and global indexes returns. Global index returns are cyclical by nature as they tend to indicate the general health of the economy or industry they represent. Having a portfolio tied to these returns mean the portfolio will oscillate with the economy. Beta-neutral strategies, such as arbitrage strategies, have the advantage of having its returns uncorrelated from the general market direction. This allows investors to safely diversify their portfolios and have their returns independent from the overall market.

With the advent of computer trading and the evolution of quantitative strategies, there has been a big attraction from the Artificial Intelligence community to explore possible applications in this area.

Pairs trading focuses on the principle of using a collection of stocks to create various legs of a single trade. Some of these legs are betting on an increase of the price of the share while others are doing the opposite, working as a hedge. The pair's selection is based on co-integration testing to check for the existence of shared hidden variables between the pairs.

It is of interest to explore how we can apply machine learning optimization techniques, such as the genetic algorithm, to optimize and create new trading strategies based on these market-neutral principles.

In this work, I have architecture and developed a system that is capable of taking in market data and generate trading signals based on a series of parameters provided. The system works by combining informative signals from a simple statistical arbitrage strategy based on technical analysis and co-integration testing to generate a tradable index, and then trade this index based on a trend-following strategy.

Since there are millions of possible different combinations to configure the system, I use an artificial intelligence technique, the genetic algorithm, to find an optimal calibration. This process involves performing several backtests with different possible calibrations in separate training and validation sets.

I have used the sharp ratio as the ranking function. The training set is used to determine the best individuals, while the validation set is used to validate the results and control for over fitness.

The main contribution from this work is the creation of this trading system based on technical analysis and pairs trading that is capable of generating profitable, beta-neutral, trading signals. A second contribution is the study of the genetic algorithm to perform optimizations over this system inputs. The results from this study allowed me to conclude that the genetic algorithm remains useful in calibrating and optimizing trading strategies.

2. Background

Arbitrage is, in theory, a risk-less trading strategy consisting of the buying and selling of equivalent goods in different markets to take advantage of a price difference. Any situation where it is possible to make a profit without taking any risk or making an investment is called an arbitrage opportunity. [15]

Arbitrage is a practice of historical importance since it contributed to the development of society by increasing liquidity where it is most in need contributing to a more efficient market and in the development of important principles such as the Law of One Price.[13]

2.1. Technical Analysis

Technical analysts believe that, due to human nature, certain prices might generate a net increase in buying or selling. In this section I explore some concepts that are used to define these prices.

Moving Averages - Moving averages are a time-series of the average price of a security in over the last t elements. Moving averages can be weighted, such as the exponential moving average that is weighted based on how close the elements of the series are to the present, or just simple moving averages [4]. The averages are also levels of interest that might signal price reversals.

Supports and Resistances - Changes in the direction of the price naturally create these levels of interest that should be taken into consideration since they are potential points for a reversion of a trend. Some examples of these prices are round numbers or local and global maximum and minimum values of prices in the past. Suppose the price of a stock is falling and fails to go lower than a certain price, then that price has become a support [5]. Resistances are the opposite of supports. They are created when prices fail to go higher than a certain level.

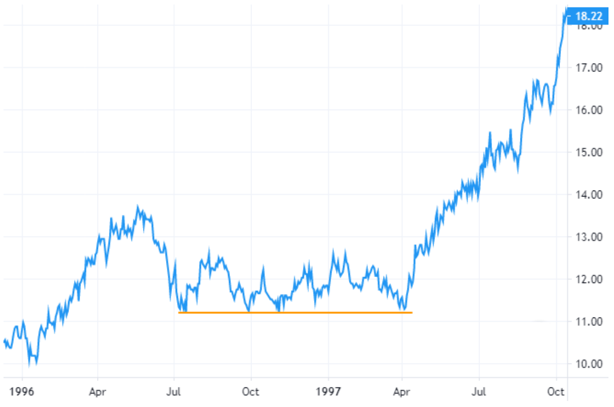

Figure 1: Support in Ford - The stock has repeatedly reverted when approaching 11$.

2.2. Cointegration

According to [14], stocks might share a weakly dependence over a certain time period. This is based on Bossaerts work [17], which finds evidence of price co-integration for US stocks. This interpretation implies that certain stocks move together not because of coincidence but because they are a product of individual integration (in the time series sense), and some linear combination of them have a lower order integration. We can also have an intuitive approach on this matter: On a fundamental level, stocks in the US share identical underlying variables or share a determined sector, so there is expected to exist some level of correlation. This interpretation implies that certain assets are weakly redundant, and when there is a deviation of their price from the linear combination of prices in other assets, it is expected to be temporary and reverting.

Vidyamurthy in his book [7] provides us a theoretical framework for co-integration based pairs trading, his work is one of the most cited in the area. The author suggests a selection of financial instruments based on fundamental and statistical analysis followed by a tradeability assessment based on Engle-Granger [3] co-integration test. Their trade signals are generated with non-parametric methods.

Other authors have studied and compared the usage of other co-integration tests. They have shown is possible to achieve similar and even better results depending on a case-to-case basis.

I briefly explain Engler-Granger co-integration test and how Boassaerts applied it to stocks since it is the most used in the approaches I reference:

According with this method, if two time-series are non-stationary and integrated of order 1, then a linear combination of them must be stationary:

$$y_t - \beta x_t = u_t \tag{1}$$

In this case,

$$u_t \tag{2}$$

is a stationary time-series. Bossaerts [17] applies this principle to stocks, he supposes that prices obey a statistical model of the form:

$$p_{it} = \sum \beta_{it} p_{lt} + \varepsilon_{it}, \quad k < n \tag{3}$$

Where

$$\varepsilon_{it} \tag{4}$$

denotes a weakly dependent error

$$p_{it} \tag{5}$$

is weakly dependent after differentiating once. Under these assumptions and according with Engle and Granger [3] and Bossaerts [17], the price vectors

$$p_t \tag{6}$$

is co-integrated of order 1 with co-integrating rank r=n-k, thus, there exists r linearly independent vectors

$$\{\alpha_q\}_{q=1..\mathrm{r}} \tag{7}$$

such that

$$z_q = \alpha_q`p_t \tag{8}$$

are weakly dependent.

2.3. Genetic Algorithm

Some authors have suggested the usage of evolutionary algorithms, such as the genetic algorithm, in the optimization of trading strategies [2] [8] [1]. The principle of these algorithms is that the strongest, or in this case, best performing, individuals survive and seed the next generation. As such, each consecutive generation is closer to full-filling the requirements and thus becoming a solution.

According to these principles [19], each individual (chromosome) is a collection of variables (genes), each group of individuals is called a population. The goal is to find the best collection of variables that solve our problem. As such, it is first necessary to encode the variables that we want to optimize into genes. In the next step, a random population is generated based on the available genes. To find the solution, I apply a simple principle based on natural selection until the new population performance is considered satisfactory for the problem by following three steps iteratively:

1. Evaluating the best individuals - The process begins by decoding each individual's genes into the corresponding collection of variables. We use these variables to attempt to solve the problem at hand and measure its performance using what is called an evaluation function. After each chromosome is tested for its ability to solve the problem, they are ranked.

2. Generate a new population - Using the best individuals from our current population, we want to generate a new population. We want to introduce variations in the variables with the objective that each proceeding population is overall better at solving our problem. There are two ways in nature that variations are generated that we can use: Crossover and Mutation.

Crossover corresponds to recombine the genes from parents to create a new chromosome. It is then a binary operation. This is shown in the image below, where the offspring inherit a part of each parent.

Mutation, a process in which each gene is slightly changed, as exemplified below where only one bit of each individual has mutated. It is a unitary operation as it corresponds to a random change in the variables [6].

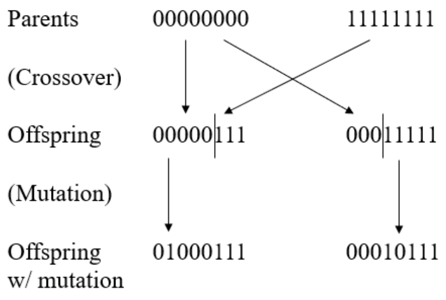

Figure 2: Crossover and mutation over one byte.

It is important to keep diversity across generations under the risk that only a concentrated portion of the solutions is tested in the solution space. This is done by spreading individuals across the search space. This can be achieved by not generating new individuals that were previously tested or that are present in our population. Secondly, it is possible to increase diversity by introducing random immigrants to each population. This is, in each generation, a random set of individuals is created similarly to the first population and introduced in the current generation.

3. Population replacement - The newly generated population is used to substitute individuals from the previous one. Not all individuals need to be replaced. Different techniques can be used to perform these selections, being the most common:

- Generational selection: No individuals from previous generations are kept. The new generation is the new population generated at each iteration.

- Steady-state selection: In this selection, all individuals that were generated, the offspring, and the individuals that were used to generate the offspring, the parents, are selected for the new generation. The individuals that were not used are discarded.

- Elitist selection: Only the best individuals from each generation are used in combination with the newly generated ones. The best individuals are the ones that scored the highest rank in the evaluation step. This selection has the advantage of making sure that the best individuals are not lost between generations.

- Tournament selection [10]: The individuals are randomly split into subgroups. The Elitist selection is applied to each subgroup, and only the fittest survive.

- Fitness proportional selection: Each individual inside each population can be ranked on how they fit relatively to its population. For example, we could pick only individuals who are one standard deviation above from the mean. We can apply different ranking functions, calling this a ranking selection.

3. Implementation

I have created a trading system intending to backtest the generation of consistent beta-neutral trading signals that make steady returns with acceptable risk levels independent of the overall direction of significant equity indexes. An arbitrage strategy is particularly challenging to backtest due to the necessity of massive computing power to generate trades across combinations of instruments instead of single ones.

The system takes in a series of calibration inputs plus the open, high, low, and close trading data of US stock prices and indexes on a minute interval. It then generates a list of trade signals to be taken at specific prices in specific quantities. Half of the signals are buying signals, while the other half is selling ones. This information is used to create a beta-neutral index that can be traded by replicating the underlying trades. A trend-following strategy is then used to trade this index both in good periods, when the trade pairs converge by longing the index. and in bad periods, when the trade pairs diverge by shorting the index. The genetic algorithm is used to calibrate the inputs for this system.

The system divides into three modules: the algorithmic trading, which composed of four submodules and responsible for creating and backtest the trading signals, the optimization module which is responsible for optimizing the algorithmic trading inputs, and the visualization module which displays the generated signals and intermediary steps in human readable way.

3.1. Algorithmic Trading Module

I begin by describing the algorithmic trading module and its five sub-modules, inspired by Prado's framework [16]: market data curation, feature analysis, strategy and backtest.

The market data is fed to the Data Curation submodule on a minute by minute basis, which is then resampled into different intervals depending on the calibration that is provided. The minimum interval for trading signals is on a daily scale. This module is capable of finding irregularities in the data, such as substantial variations in very short periods of time and prevent using that instrument. This is done so by highlighting and filtering out datapoints with unrealistic values. An unrealistic value is categorized as a twenty percent change in values both before and after this datapoint within a one-minute period.

The curated data is exported to CSV files, which are reused to save computational resources in future runs. The processed that is then instantiated as a list of vectors with information regarding the date, high, low, open, and close of the instruments used.

Since I am exploring the trading of pairs of equities, it is also in this module where common trading intervals between the pairs are computed and saved. Trading pairs also means that each instrument will be backtested with several other combinations of instruments; as such, this list of vectors is cached in memory to reduce runtime duration significantly.

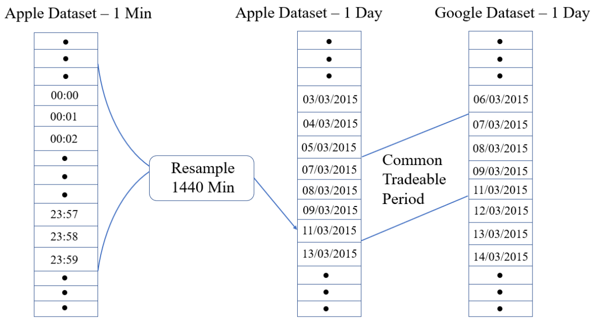

Figure 3: Resampling of the historical market data.

As it can be seen in the image above, the minute data of each dataset is resampled into higher timeframes such as on the one day scale by combining 1440 minutes from the original set. Common tradeable periods are found between the pairs of instruments using the resampled data. I pass this data to the Feature Analysis sub-module, in which I used the transform output of the previous module to create informative signals. I generate signals by applying filters and simple strategies to the curated datasets.

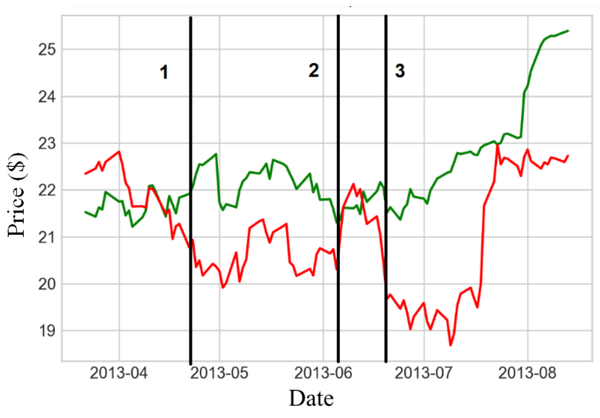

Figure 4: Trading signals in the pair HOLX - FISV.

The signals generated come from breaking a support or resistance in one of the instruments in the pair. In the picture above, I have identified three signals, where one of the stocks made a new local high/low, and the other did not. If a stock makes a new high and its pair does not, then a signal is generated: A buying signal for the stock that did not make the new high and a selling signal for the one that did. This strategy works in reverse as well, buying an instrument that made a new low and selling the one that did not. The period used for the highs and lows is defined by the calibration provided. In the case of contradictory signals, both are invalidated.

For each signal, the beta coefficient between the pairs within a certain time window is calculated. This coefficient is determined by performing a linear regression between the two prices. Using this coefficient, the spread between the instruments can be determined as explained in the literature review. Following this determination, stationary tests are performed using the Augment Dickey-Fuller test, this process follows the Engler-Granger's co-integration test. The window duration for the co-integration tests, along with the cut-off threshold, is defined in the calibration provided. Using the remaining signals, an index is calculated. This index is calculated by simulating a portfolio of 1000 dollars, which evenly exposes its capital between all the signals generated. The quantities of capital allocated to each share are defined by performing a linear regression between the trading instruments, similarly to the beta-coefficient estimation.

Next is, the Strategy sub-module in which the informative signals are taken, and trades are generated. The strategy will provide indicative prices and quantities for the instrument to be shorted or longed along with holding periods based on pairs combinations that are fed.

In the Strategy sub-module, two moving averages are calculated for the index generated in the Features Analysis. These moving averages correspond to the average of the index over two periods of time, which are passed by the calibration used. A slow-moving average is the one that extends over a longer period of time; it is called slow due to being less sensitive to each individual point in time.

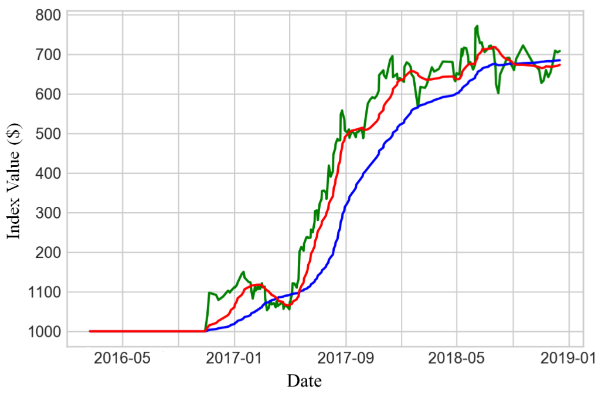

Figure 5: The index generated by a random calibration and its averages.

The image demonstrates the index generated by the high lows strategy in green for a period of two years.

In blue, the slow-moving average is used to determine the overall trend of the signal. If the signal is above this average, we consider it bullish, and we should replicate the underlying trades of the index. If the signal is below this average, we should do the opposite that the underlying portfolio of the signal is suggesting.

The red line is the fast-moving average, which determines when to trade. I first start trading the index portfolio once the index crosses the slow-moving average, and we stop when the index crosses with the fast-moving average in the opposite direction. This aims to exploit the mean-reversion nature of the technical analysis in the underlying index, which tends to consist in a short burst of movement after prices and spread have over-extended during a large period of time.

The usage of a fast-moving average to exit the positions helps mitigate losses during periods when the direction of the movements in the spread of the underlying stocks of the index is changing.

The Backtest sub-module uses the information produced by the Strategy sub-module and the Data Curation sub-module to reproduce an hypothetical portfolio returns that would follow the strategy in the pervious sub-module. This backtest also calculates risk metrics for the portfolio generated.

By using the curated data in combination with the trading signals produced by the strategy, a portfolio simulation of past returns is created. For each trade, I have considered a round-trip commission of 0.2% of the total trade value. The period for this backtest is between the year 2013 and the year 2018. Due to a lack of historical price data, it was not possible to perform backtests over other periods. The backtest results are used to calculate risk and return metrics: growth, drawdown, winning/losing rates, sharp ratio. The backtest sensitivity to S&P 500 price, beta risk, is determined and is used to validate that our portfolio is beta neutral as statistical arbitrage systems should.

3.2. Optimization module

The optimization module is responsible for calibrating the algorithmic trading module. This is done by applying the genetic algorithm over several backtests for different possible calibrations. Each calibration input is considered a gene. An individual's performance is compared using the sharp ratio generated by the backtest. In total, there are 5 040 000 different possible combinations for the calibration; thus, finding the ideal one is a challenging and computational intensive task.

The calibration process is separated into generations and epochs. Each generation is composed of six epochs, and each epoch is composed of eight individuals. Initially, these individuals are generated by randomly selecting the dominant genes from the gene pool shared across all chromosomes. In each epoch, a backtest is run for every single individual with a limited set of the available data. Each individual and its backtest result are cached and reused in future epochs. At the end of an epoch, the individuals are ranked based on sharp ratio. The sharp ratio is defined as the difference between the returns of the investment and the risk-free return, divided by the standard deviation of the investment [18]. I have consider the risk-free return to be 0%. The standard deviation is the volatility of the investment.

In the selection step, I followed a steady-state selection. Since I generate the offspring using the two individuals with the best sharp ratio, only they get to carry their genes to the next epochs and generations while all the others are discarded.

The number of offspring resulted from these two individuals is dependent on the generation. In the first generation, to ensure that enough variety of possible solutions are tested, the best individuals of the epoch only produce one offspring, which consists of a crossover of the parent individuals and the introduction of slight mutations. The remaining individuals are again randomly generated. This is done to make sure there is enough genetic diversity across generations, effectively constituting an implementation of the immigrant technique. In the next four generations, the number of offspring increases by one, which allows a breath search for the solution at the beginning of the algorithm and an increasing depth search to be made as generations progress. At the end of each epoch, the population evolves using a mutation ratio of 30% and generating g, where g equals the current generation, of offsprings from the best individuals in the population.

These calibrations are then tested against a different dataset. This set is taken from the same pool of companies, however, its market information is in the future in comparison to the previous set. The resulting sharp ratio of each calibration is saved and used later in analysis to control for over-fitness of the calibration and validated the results.

Each calibration is saved in a separate file with a name that facilitates the translation to the inputs of the system, which we can see in the following example:

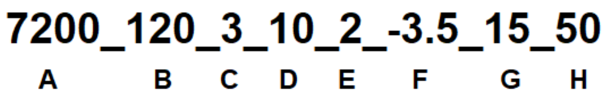

Figure 6: Each input of the system encoded in its name.

Each section is divided by an underscore. We can map each section to its respective input:

- A. Number in minutes for the resampling. In this case, 7200 minutes equal to five days, or one week.
- B. The number of periods to be used in the co-integration test window. In this case, 120 periods or 120 weeks.
- C. The number of periods each trade is held. In this case, 3 weeks.
- D. Number of periods to consider in the window for the new high and new low values of each instrument in the trading pairs, 10 weeks in this case.
- E. Max-lags for the testing of serial correlation in the Augmented Dickey–Fuller test. In this case 2, but could be instead determined using significance tests.
- F. The cut-off for the p-value in the co-integration test. In this case, -3.5 corresponds to 1% [12].
- G. Fast-moving average in tradable days. In this case, 15.

- H. Slow-moving average in tradable days. 50 weeks in this calibration.

  Besides the leverage, which is not coded as a gene of the calibration along with the market data, these are all the inputs that are needed to perform a backtest.

3.3. Visualization module

Finally the visualization modules consist of a python library that is able to parse the computational artifacts generated in the Algorithmic Trading and Optimization Modules into objects in memory and transform them into pandas datasets that are loaded in a Jupyter [11] notebook, making it possible to render these objects into human-readable graphics and tables using an open-source visualization: mathplotlib [9]. I have used this module to manually validate the results on each computational step during development, facilitating debugging in case of errors, and allowing the study of each separate component in the system.

3.4. Computational Optimizations

Pairs trading strategies deal with an increased amount of data when compared with strategies that are used on single instruments. The combinatory explosion means it is required that the system implements several computational optimizations in order to process the large amounts of data in a timely fashion. When performing a backtest, only 113 stocks would be introduced into the system if we were not performing a pairs trading strategy. However, by generating all possible pairs of 2 of this number increases to 6434. Splitting calculations between processes was one of the first optimizations performed, but I was still limited by the CPU – 16 virtual processors.

This could be further improved by making improvements in the code in a way it is able to distribute the computing workload across various machines in the cloud. However, the high costs involved in keeping a grid of servers operating did not make this improvement a viable option.

The next approach was to use the caching of previously calculated tasks in future backtests. After analyzing the execution time of separate parts of the code, it was determined that loading and resampling the market data was consuming about half of each backtest. I have optimized the system by keeping in RAM the market data of each individual stock in different sampling rates. Caching the market data provided a significant improvement of about 7-10 seconds on each backtest (average duration was 16-18 seconds). To be able to run the calibration uninterruptedly, I have configured a cloud web server to run the optimization module and later secure copied the results. Since I had to test different providers, I have created shell scripts that automatically perform the configuration for Ubuntu 18.0 servers to be able to run the proposed system.

Finally, to ensure reproducibility and as a safety mechanism for events that might disrupt or interrupt the calibration, there are several checkpoints where objects that resulted from lengthy computations are exported and written in the disk along with the logs of the application. In case of failure, when resuming the calibration process, the existence of already previously calculated objects is checked before performing computationally intensive parts and loaded into memory.

4. Results

I explored the main problem-question: if it is possible to use machine learning techniques to optimize a simple statistical arbitrage strategy. To do so, I use the previously described system and collect its backtest performance using random calibrations. I then attempt to optimize the calibrations using the optimization module and compare the results against several risk metrics.

4.1. Hypothesis

The main problem question I am tackling in this section is if it is possible to use artificial intelligence techniques to optimize a statistical arbitrage strategy. As such, this statement should hold as truth if I can find and measure the results of a statistical arbitrage strategy and if later, I can generate improved results by optimizing some part of the statistical arbitrage strategy, or it's whole. If I am able to generate improved backtest results by calibrating the system using the genetic algorithm, then I can conclude it is possible to use artificial intelligence techniques to optimize these types of strategies.

However, if I am not able to determine with confidence that the results have been improved due to the usage of the genetic algorithm, or if the results are not improved by the usage of the genetic algorithm, then the hypothesis remains open as it is still possible that there are other systems or other artificial intelligence optimization techniques that could be used to improve the results.

4.2. Results Generation

I began by generating random calibrations for the system within a range of parameters where these calibrations would still make sense (for example, the slow-moving average could not be faster than the fast-moving average). Next, I have separated the available market data into two sets: a test set and a validation set. I have then performed backtests over the validation set for each of the randomly generated calibrations and collected the results.

After this, I have proceeded to feed the randomly generated configurations as the first population of the optimization module. I have run the algorithm for eight generations with six epochs each. Each epoch had eight individuals. Instead of using the validation set, the optimization applied the genetic algorithm over these individuals using the market data in the test set. This is done in order to not bias the training towards the same set of data from where I will extract the performance. Having this separation also allows me to check for over-fitness of the training.

Since the system uses information up to 200 weeks prior to the current day to generate the signals and perform co-integration tests, the historical data in the validation set also contains data from the training set. However, no trades are performed under the same period. As such, the data from the training set in the validation set is only used to calculate indicator data in the feature analysis sub-module of the algorithmic trading module.

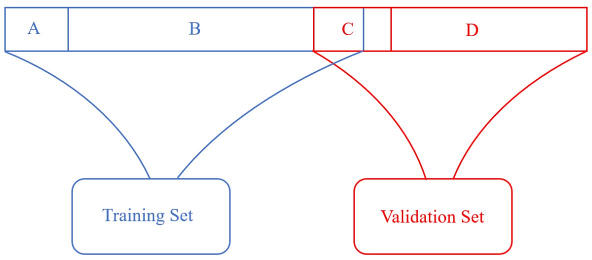

Figure 7: Overlapping of historical data in the sets.

The image above shows the data that is used to generate the trading signals and the separation between the training, letters A and B, and validation set, letters C and D. Each set is broken and categorized into two other sets, indicator calculation, letters A and C, and trading set, letters B and D.

The optimization module feeds on data that spans from the beginning of the set in 2014 until a cut-off in late 2016 to find the best calibrations. The results are validated against the data that spans from mid-2016 until the end of the set in 2018, identified by the letters C and D. Therefore, there is one year of data where both sets are overlapping, which happens due to C being partially contained in B.

While the co-integration tests and indicator data use information from the past days that are used in the training set, it should be noted that the core trigger for a trade signal generation does not have information contained in this previous set. This is because the trigger is given by the technical analysis/arbitrage signal resulting from a break of a local support or resistance in one of the pairs being watched, and the overlapping only exists during indicator calculation of the validation set: There is no overlapping between the trading periods.

4.3. Results Interpretation

The results from the experiments were collected and displayed in Jupyter [9] using the visualization module. Backtesting the first randomly generated calibration under the validation set, I arrived at the following results:

Name	Returns (%)	Max Drawdown(%)	Sharp Ratio
Random 1	2.000	-11.000	0.004024
Random 2	1.000	-8.000	-0.020731
Random 3	-6.000	-14.000	0.017443
Random 4	9.000	-26.000	0.006485
Random 5	3.000	-17.000	-0.068782
Random 6	-18.000	-26.000	-0.033517
Random 7	-32.000	-49.000	-0.033517
Random 8	18.000	-6.000	0.044985
Average	-2.875	-19.625	-0.010452

Table 1: Random calibrations portfolio statistics.

From this table, we can see that the best individual, named Random 8, has a daily sharp ratio of 0.044985, roughly 0.71 annually. The average daily sharp-ratio under the validation set was of only 0.005. The rest of the individuals show far worst returns. The average daily return of the strategy using a random calibration is around -0.9%. The average sharp ratio is also negative. These indicate that random calibrations lose money.

Since the optimization module uses the genetic algorithm to find better calibrations, if I can find a better calibration using this optimization module, then I can validate the hypothesis under study. The optimization module must be able to generate better returns than random calibrations by showing a clear trend of improved performance across the training and validation sets, with eventually a decline in the validation set due to overfitting. If by using the genetic algorithm, I can beat these results, then I can affirm it is possible to use artificial intelligence techniques to optimize statistical arbitrage systems. After iterating the genetic algorithm over a total of 155 populations, it has become apparent the new populations were getting more and more overfit.

In the following figure with the green line, we see the average sharp-ratio of the population of a specific generation and epoch over the training set. This line is getting higher and higher with each iteration, indicating that the system is getting improvements under the training data. The red line indicates the average sharp-ratio of the same population, only this time over the validation set. It is possible to see that the red line begins to track the green line until iteration 27, coinciding with the fifth epoch of the fourth generation. After this iteration, it begins to decrease sharply, an indication that the calibrations are beginning to overfit towards the training set.

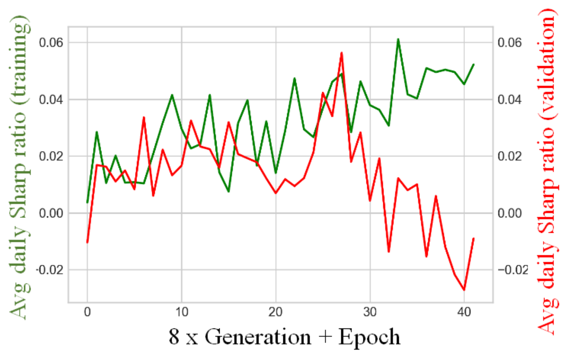

Figure 8: Average population sharp-ratio evolution over generation and epoch by dataset.

This analysis is not enough to determine what is the best calibration to use. However, we can validate the performance of the optimization module, clearly increasing the performance of the calibrations until the calibrations start to enter in overfitting. To find the best candidate, I have ordered each calibration by the training sharp-ratio plus validation sharp-ratio. The top 8 calibrations can be found in this table and are indicated by the column INDIVIDUAL:

Individual	Total	Training	Validation
7200_120_3_10_2_-3.5_15_50	0.205	0.091	0.114
7200_120_5_10_2_-3.5_15_30	0.198	0.040	0.158
7200_45_2_50_2_-3.5_70_120	0.187	0.071	0.117
7200_120_3_6_2_-3.5_5_20	0.177	0.054	0.124
7200_200_4_40_1_-3_70_20	0.174	0.047	0.127
7200_120_3_8_1_-3.5_5_50	0.173	0.058	0.116
7200_120_3_10_2_-3.5_40_50	0.143	0.039	0.104
7200_200_3_6_2_-2.5_15_20	0.137	0.089	0.048

Table 2: Ranks of the best calibrations found by the optimization module.

The TRAINING and VALIDATION columns are the corresponding sharp-ratio for training and validation sets. The TOTAL is the sum between the training sharp-ratio and the validation sharp-ratio.

From this table, we can conclude that we should use one of the calibrations in the top 3. We can see that the two calibrations on the top are extremely similar, being the only difference in the speed of the fast-moving average and the number of holding periods. Although I decided to pick the top configuration, the second one had a better performance in the validation set. The training set was far worst. In the top calibration, the validation set surpassed the training set slightly, which is a sign that there is no overfit.

Finally, I am interested in understanding if this strategy's returns remain independent from the overall market direction after applying the trend-following strategy to the signal index generated by applying the high-lows strategy described before. To do so, I plotted the best calibration returns against the S&P 500 returns and calculate the beta coefficient of the portfolio as can been seen in the following image:

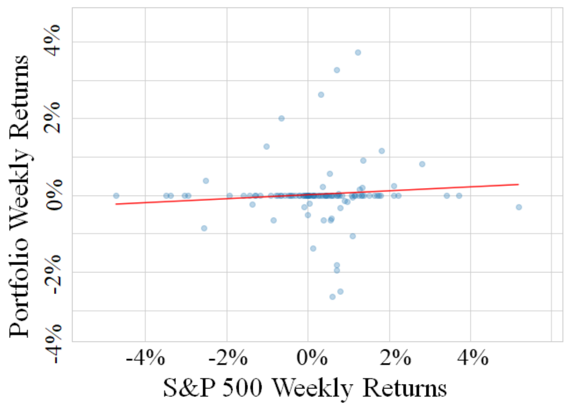

Figure 9: Portfolio returns compared to S&P returns.

The beta coefficient of this strategy is only 0.05, contrasting with values close or above one as seen in individual stocks, meaning that the beta neutral properties are present in the strategy. The portfolio returns are highly uncorrelated from the S&P 500 returns. This result was expected since every trade his composed by one long and a short. Since the amounts of each instrument are controlled based on the volatility of each, the signals generated by the high-lows strategy described in the previous section are beta-neutral. From this analysis, we can also conclude that after applying the trend-following strategy optimization over the generated index, this same beta-neutral property has been preserved.

5. Conclusions

This study attempted to optimize a pairs-trading strategy using the genetic algorithm.

After the usage of the optimization module, the performance of my statistical arbitrage strategy has increased as shown by the increase in sharp ratio, and since the beta-neutrality properties of the statistical arbitrage strategy were preserved, these results allow me to conclude that it is possible to use artificial intelligence techniques to optimize trading strategies based on statistical arbitrage principles, in particular, using the genetic algorithm in order to find the best calibrations for the strategies.

For further work, I suggest the exploration of this system under a larger dataset of historical market prices to further validate the results. I believe it is of interest to explore the backtesting of the system using different asset classes and across other markets besides the U.S.

References

[1] N. H. A. Gorgulho, R. Neves. Using gas to balance technical indicators on stock picking for financial portfolio composition. *Journal of Financial Economics*, pages 2041–2046, 2009.

[2] K. R. Allen F. Using genetic algorithms to find technical trading rules. *Journal of Financial Economics*, 5:245–275, 1999.

[3] D. B. and F. R. Does simple pairs trading still work? *Financial Analysts Journal*, 66(4):83–95, 2010.

[4] R. Edwards and J. Magee. *Technical Analysis of Stock Trends*, chapter 15, page 295. AMACOM, 2017.

[5] R. Edwards and J. Magee. *Technical Analysis of Stock Trends*, chapter 13, page 231. AMACOM, 2017.

[6] S. G. Uniform crossover in genetic algorithms. *Proceedings of the 3rd International Conference on Genetic Algorithms*, pages 2–9, 1989.

[7] V. G. Pairs trading: Quantitative methods and analysis. Hoboken, New Jersey, 2004.

[8] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*, page 295. Addison-Wesley Publishing Co., Inc, Redwood City, Ca., 1989.

[9] P. Jupyter. Project jupyter. https://jupyter.org, 06 2020. Last accessed 25th June 2020.

[10] H. K. and H. M. Equivalence of probabilistic tournament and polynomial ranking selection. *Evolutionary Computation - IEEE World Congress on Computational Intelligence*, pages 564–571, 2008.

[11] M. P. Library. mathplotlib. https://matplotlib.org, 06 2020. Last accessed 25th June 2020.

[12] J. MacKinnon. Critical values for cointegration tests. Queen"s University, Dept of Economics, Working Papers. Available at http://ideas.repec.org/p/qed/wpaper/1227.html, 2010.

[13] NASDAQ. Law of one price definition. NASDAQ Glossary - https://www.nasdaq.com/investing/glossary/l/law-of-one-price, 04 2020. Last accessed 25th April 2020.

[14] B. P. Common nonstationary components in stock prices. *Journal of Economic Dynamics and Control - 12*, 1988.

[15] G. Poitras. Arbitrage: Historical perspectives. *Encyclopedia of Quantitative Finance*, 2016.

[16] M. Prado. *Advancements in financial machine learning*, page 8. Wiley, 2018.

[17] E. R. and G. C. Co-integration and error correction: Representation, estimation, and testing. *Econometrica*, 55(2):251, 1987.

[18] W. F. Sharpe. The sharpe ratio. *The Journal of Portfolio Management*, pages 49–58, 1994.

[19] M. Z. *Genetic Algorithms + Data Structures = Evolution Programs.* Springer, 1992.