Karlsruher Institut für Technologie

Institut für Operations Research
Lehrstuhl für stochastische Optimierung
Prof. Dr. Steffen Rebennack

Trending Topics in Machine Learning and Optimisation (Bachelor/Master)

Reinforcement Learning-based hyper-heuristics for Financial Forecasting

by

Keim, Robin
Matr. Nr.: 2496035
Wirtschaftsinformatik M. Sc.

Date
20.12.2024

Adviser: Dr. John Alasdair Warwicker

Erklärung

Ich versichere wahrheitsgemäß die Arbeit selbstständig angefertigt und alle benutzten Hilfsmittel und Quellen vollständig angegeben zu haben, die wörtlich oder inhaltlich übernommenen Stellen als solche kenntlich gemacht zu haben und die Satzung des KIT zur Sicherung guter wissenschaftlicher Praxis beachtet zu haben.

30.12.2024
Datum

RL=C=
Name

Abstract

Financial forecasting has consistently been a significant area of interest, engaging both scientific researchers and the general population. As monetary assets form the backbone of the global economy, optimizing financial investments continues to be a critical endeavor.
Previous research has explored incorporating hyper-heuristics to enhance the performance of genetic algorithms in predicting financial outcomes.
This implementation focuses on extending this work by including Reinforcement Learning-based hyper-heuristics to not only improve results but also speed up the genetic algorithm. By rewarding performance improvements of the strategies, the actions of the hyper-heuristics are trained. These actions include setting the probability for mutation operations and local search methods.
The resulting implementation has been compared to baseline models such as an artificial neural network, a support vector machine and a random forest decision algorithm. Although the model outperformed its counterparts in terms of Sharpe Ratio, maximum drawdown, and overall profit, significant opportunities for further optimization remain. After implementing a better starting point for the algorithm by ensuring a more meaningful initialization of strategies as well as setting custom initial probabilities for the actions of the Reinforcement Learning algorithm, the second implementation was able to score even more promising results in terms of Sharpe Ratio and total profit.

Contents

1 Introduction

Algorithmic trading has a long history of being used to forecast future price movements for commodities. There have been numerous improvements lately related to recent developments in computational resources. With greater computational power comes the potential to increase the search space for algorithmic trading.
Recent advancements in artificial intelligence have revolutionized trading by enabling more adaptive and efficient strategies. Neural networks, in particular, have become more computationally efficient, making them increasingly practical for trading applications. Their user-friendly nature further enhances their appeal to non-expert users. By using various indicators, a simple neural network can autonomously determine their weights to generate predictions. Looking at current research on neural networks by Gupta et al. [3] one can observe that these strategies already can result in profitable growth.
Machine learning techniques, including support vector machines and random forests, have garnered significant attention for their efficacy in forecasting trading signals. Also different methods of artificial intelligence like support vector machines, random forests and Reinforcement Learning. Concerning the dynamic nature of trading, especially Reinforcement Learning becomes an interesting algorithm. The continuous influx of data and the emergence of novel market trends necessitate the frequent updating of prediction models. Unlike neural networks, which require retraining to incorporate new trends, Reinforcement Learning offers a dynamic framework to adjust strategy selection by rewarding high-performing models.
Genetic algorithms enhance trading strategy performance by facilitating the exploration of extensive search spaces using evolutionary processes. These algorithms can start with a small population and genetically modify their individuals to explore a larger search space. While enabling an accelerated exploration of diverse strategies, it is crucial to effectively guide this evolution toward optimal outcomes. Because the genetic operations used on the population just introduce more diversity, different methods need to be performed to achieve a convergence on better results. These methods involve selection techniques designed to reduce population size while retaining high-performing individuals for subsequent iterations. By performing this selection not only bad results get filtered out but also good results become more often selected by the genetic operations.
A supplementary approach involves augmenting genetic operations with advanced guiding mechanisms to improve optimization efficacy. By performing

local search methods like hill climbing or simulated annealing in multiple steps along the genetic algorithm convergence can be achieved much faster resulting in better computational efficiency.
A question arising when using different operations in the genetic algorithm is when to apply which algorithm. One way is to choose which method is used while constructing the algorithm, whilst another way is to learn probabilities for how often to apply these actions. An adequate framework for this purpose are hyper-heuristics. Here the different mutations of the algorithm are understood as low-level heuristics. Hyper-heuristics dynamically assign probabilities to low-level heuristics, enabling adaptive strategy selection. By using hyper-heuristics a model can dynamically learn when to mutate different strategies to achieve better convergence. An effective approach to achieve this learning process is by combining the hyper-heuristic with Reinforcement Learning. Reinforcement Learning can dynamically learn to adjust weights according to current conditions in markets. This approach combines the efficiency of genetic algorithms with the adaptability of dynamically responding to current trends, while preserving a large search space.
In the following an implementation of a Reinforcement Learning-based hyper-heuristic approach to genetic algorithms for financial forecasting shall be discussed. Firstly, there is a literature review on current research on hyper-heuristics and the Reinforcement Learning-based approach to it. With the rise of artificial intelligence this field is also getting more attention. Next a look into the topic of financial forecasting and what problems may arise will be provided. Afterwards different approaches are discussed tackling the difficulties in developing trading strategies. While there is a large interest in artificial intelligence there are also benefits towards using different approaches. To dive deeper into these topics an implementation of the hyper-heuristics approach is discussed. It uses multiple different low-level heuristics such as mutation, crossover and hill climbing. While the training process will be done on a single share the results of this approach will also be evaluated on different performance metrics and time frames. Lastly the results are discussed and conclusions will be drawn.

2 Literature Review

This section reviews existing work on genetic programming, hyper-heuristics, Reinforcement Learning and financial forecasting to contextualize the proposed methodology. First the different literature will be reviewed, then limitations in the existing literature will be shown. In the end it will be shown how this work relates to the literature.
First a comprehensive look at current research about genetic programming will be provided. Afterwards a look at how hyperheuristics and feature engineering can improve the performance of these algorithms with Reinforcement Learning is given. The third section of this chapter will focus on different current methods in financial forecasting and which technical indicators to use for prediction.
The main focus throughout this review remains on how Reinforcement Learning-based hyperheuristics can improve the prediction power on financial trends.

2.1 Background on Genetic Programming

John Holland provides the first work published on genetic algorithms [4]. These algorithms focus on reproducing the concepts of natural selection and reproduction in mathematical optimization problems.
John R. Koza [6] adapts this approach and focuses on how to effectively formulate and solve this problem. He suggests a dynamic tree-like structure for the individuals undergoing the genetic adaptation process.
These trees consist of functions and terminals, where functions can contain logical, mathematical and iterative operators. Terminals represent variables, constants, or argument-free functions within the system. The main advantage of this approach is that the length of the individuals is dynamic, which results in a more complex and broader search space. Also the tree structure can be computed quite fast.
For initialization a random function gets selected first and next arguments respective to its arity are selected. These can be also functions or terminals, whether the latter results in the end of expanding the tree path.
Next a fitness function needs to be defined which evaluates the populations performance on solving the problem. John R. Koza proposes that this function returns the accumulated distance between the calculated value of one tree and the desired output.
He then suggests two different operations to change the population in every so called generation. One of them is the reproduction proportionate to their

fitness resulting in the chance to be used again in the next generation for an individual is higher if it has a higher fitness value. The second method he suggests is crossover which creates offsprings of two parents by replacing subtrees of them with subtrees of the other one. Other operations mentioned are mutation, permutation, editing and define building blocks.
He mentions that there are two ways to terminate the algorithm. One of them is to specify a maximum of generations after which the algorithm gets stopped. The second way is to stop after the population reaches a certain performance level. After the algorithm has terminated the best individual can be considered as the output of the algorithm.
According to Katoch et al. [5] one of the largest fields of application for genetic algorithms is the operation management. Some of the specified tasks include facility layout, scheduling of jobs or processes and forecasting for inventory or consumer predictions. Several reasons for this stem from the nature of operation management such as a large discontinuous search space, multiple objectives and constrains.
Another field is multimedia, which contains as an example the selection of parameters for encryption. There are also promising results in image and video processing.
Load balancing is also a promising field for genetic algorithms. This also includes setting bandwidth and allocation of channels.
Genetic Algorithms in their purest form have several problems and limitations, which result in suboptimal results. Luckily there are solutions to tackle these problems, which will be named here and also discussed throughout the paper.
Key parameters, such as population size and number of generations, must be optimized to balance search space exploration and computational efficiency. The trade-off that has to be done here is between the size of the search space and the computational speed of the algorithm.
Another problem concerning the population is how to construct the initial population. The most important factor is to have diversity in the initial population to discover most of the search space. Also in some optimization problems there may be several great strategies, which can be included to get a more meaningful initial population.
The fitness function is one of the most important factors for the genetic evolution. If this function solely focuses on the accuracy of the solution it may result in overfitting to the given data. Thus it is important to include performance on unseen data.
Also there may be multiple objectives, which are simultaneously tried to be achieved. For this purpose there are also other methods than just weighting these values into one fitness value. One of them is the concept of pareto-

based fitness functions. In this approach the best individuals are selected as the ones which are not dominated by other ones. This means that no single objective is optimized in another individual while no other objective gets worsened. The second approach to achieve this is a decompisition-based approach. It solves the problem for each of the fitness values and exchanges solution between the different solvers.
Premature convergence is another common issue, where the algorithm settles on a local optimum rather than a global one. While wanting the algorithm to converge in a reasonable time, it is important to keep a diverse population to not run into local optima. This can be achieved by keeping a large population and not overly strong selection of the best individuals.
Lastly it is also a difficult task to choose the right degree of mutation and crossover to apply on the population. While wanting to explore the search space it is also important for convergence to keep the best individuals. This can be achieved through elitism which ensures the best individuals are passed on to the next generation. Also adjustment of the mutation probabilities is an option, which will be discussed in the following chapters.

2.2 Reinforcement Learning-based Hyper-Heuristics

Hyper-heuristics are a framework, which is deployed on a set of low-level heuristics. Low-level heuristics can include diverse search algorithms. Some of these include also the above mentioned genetic methods like mutation and crossover. Other algorithms can be of the type local search like hill climbing or simulated annealing. Another sort of genetic methods is ruin and recreate, which will not be used in this approach.
Hyper-heuristics are broadly classified into generating low-level heuristics and selecting among existing low-level heuristics. The selection of the low-level heuristics can be either be achieved by choosing them or by setting probabilities for them.
So selection hyper-heuristics can be seen as an advanced version of genetic algorithms to extend their functionality. It also tackles the problem of choosing the probability of the genetic methods.
There are also quite a different approaches to selection hyper-heuristics out there. These all have different ways to choose the probabilities to which the different low-level heuristics are chosen. For classification these can be firstly divided into static and dynamic algorithms.
The static algorithm is a rule-based model where given the current state of the problem specific probabilities are selected for the different low-level heuris-

tics. These rules are pre-specified and thus need human domain knowledge to be correctly set. While the probabilities can change, the rules cannot. Therefore this system is not able to adapt to current trends in the search space.
To encounter this problem dynamic selection hyper-heuristics shall now be discussed. The approach that also will be chosen in this implementation is a Reinforcement Learning-based selection. By assessing the performance of the current population the hyper-heuristic learns the effect of the chosen probabilities. It learns how to adapt these probabilities given the current state. This approach learns to balance exploration and exploitation which is essential to problem solving as mentioned earlier.
Another simple solution for this problem would be based on simulated annealing, which ensures high exploration in the beginning and later on more exploitation. To keep the implementation more simple, this method will not be used.
There are also other methods to learn the probabilities like neural networks or evolutionary algorithms. Also these can be combined into a learning classifier system which evolves rules for setting probabilities and updates them accordingly.
While all these dynamic approaches learn the probabilities while also solving the problem, there is an approach which first learns the probabilities on a different dataset. This so called offline system then applies the learned probabilities onto the problem set.
While there are several different methods to deal with hyper-heuristics, this work will focus on Reinforcement Learning as a basis. There have been several improvements in Reinforcement Learning over the last years. Thus this implementation focuses on using these advantages to improve the performance of hyper-heuristics.

2.3 Financial Forecasting

Financial forecasting has long been a focal area of research due to its profound economic implications and potential for monetary gains. This results in a lot of different literature that needs to be reviewed. To keep this section more dense some traditional methods will be named and afterwards some current methods rising in interest. The main focus will be on the use of genetic programming and hyper-heuristics in financial forecasting.
A very simple method to predict values is linear regression. It uses independent variables to predict the dependent variable which in this case is the stock price. It is not hard to show that most values influencing stock prices

are not independent thus resulting in poor performance of this method.
Another traditional method is AutoRegressive Integrated Moving Average (ARIMA) which uses past values of the stock price to predict future ones. While it is very effective for univariate time series it assumes linear relationships as well as linear regression, which also limits its performance.
The third traditional method to be named here is exponential smoothing which smoothes data to forecast short-term trends and seasonality. While it no longer focuses on linear relationships, it struggles with abrupt structural changes as well as the before mentioned algorithms. With financial markets being volatile and dynamic in their behavior this is a great concern for these traditional methods.
Given these limitations next there will be some artifial intelligence methods mentioned which raise interest currently because they are able handle some or even all of the problems.
The first method will be support vector machines which calculate a hyperplane to seperate data in different classes. With this approach it is able to handle high-dimensional spaces and to model non-linear relationships. While they also are not great in detecting sudden changing conditions, the results of them are also hard to interpret.
An approach with much more interpretable strategies are random forests which combine multiple decision trees to improve their accuracy. They are also quite efficient against overfitting. But with their normal approach not being designed for sequential dependencies they need feature engineering to solve time-series data.
Artificial neural networks have garnered widespread interest across numerous disciplines due to their versatility and computational efficacy. With multilayer perceptrons getting more efficient in computing, they are able to use the large amount of data that is needed to achieve accurate modeling. In financial fields providing this amount of data is not the problem with lots of historical data on stock markets out there. One special method which should be mentioned here are Long Short-Term Memory neural networks with their ability to detect time-dependent patterns in sequential data.
To get better insights on the implementation of this work in the end it will be compared to a support vector machine, a random forest model and a basic artificial neural network.

2.4 Hyper-Heuristics in Financial Forecasting

Technical indicators are quantitative tools used to analyze price movements, trading volume, and other market data to forecast future price movements in financial markets. They are the main used features used in the above mentioned algorithms. Below is an overview of some widely used technical indicators.
There are several different classes of indicators. The indicators can be grouped into trend indicators, volatility indicators and identification of overbought and oversold conditions. There are also indicators which combine the different classes.
The selection of the correct indicators and their corresponding parameters is a quite hard problem especially given the large search space. For this problem genetic algorithms provide a great solution. While maintaining a large population this algorithm can search in a large space of the possible solutions and is able to find great trading strategies containing a combination of different indicators and their best parameters.
To speed up the computation time of the algorithm precomputed indicators will be used. This means several parameters, which have been proven effective, will be chosen and the values for the features will be computed forehand to ensure that they are not computed multiple times at runtime.
While genetic algorithms do provide an effective approach to search for optimal solutions, it is often quite hard to balance exploration and exploitation. For this task a hyper-heuristics framework is a good solution because it not only extends functions used inside the algorithm, it also learns to set the probabilities for the different methods.
In Cui et al. [2] a different definition is shown. Instead of using the low-level heuristic to manage different strategies by methods like genetic mutations or hill climbing, the low-level heuristics are directly defined as the different strategies. The hyper-heuristic then in this work is the Reinforcement Learning which is applied to the strategies to sequentially select the best strategy which currently suits the data. While this approach also seems to lead to promising results, in this paper the heuristics will be abstracted to a level above the mentioned ones.
Another work to be mentioned here is by Kampouridis et al. [1] which follows an almost similar workflow as this implementation. It also abstracts on the higher mentioned dimension. Therefore it is also able to leverage multiple different optimization strategies like genetic mutations, crossovers and hill climbing. As mentioned in the paper this framework leads to significant improvements in the performance of the strategy. The limitation for this work

is that the probabilities for the different optimization techniques are directly updated with a strict value by whether the action has improved or decreased the performance on the data set. To enhance this approach Reinforcement Learning will be used in this implementation to balance exploration in the action space and then turn more to exploitation in later phases of the algorithm.

2.5 Limitations of Current Approaches

While the above discussed approach do implement hyper-heuristics for financial forecasting, they do not use Reinforcement Learning on the weighting of multiple optimisation techniques. Reinforcement Learning specifically has be shown to improve the balance between exploration and exploitation. Not only does it provide a great balance it also learns the parameters on its own to apply this.
Combining the existing work with the implementation of Reinforcement Learning should still contain the efficient way of genetic algorithms, the convergence of local search algorithms and the balance of all those aspects.
This work addresses limitations shown of basic hyper-heuristics frameworks and compares it to several different algorithms. It advances hyper-heuristics with a Reinforcement Learning-based approach to tackle the problem of choosing the correct probabilities for the methods applied.
Also Reinforcement Learning can be used to adjust for the dynamic nature of the financial world when it is applied over time. This should not be part of this work, instead it just focuses on a training set of data and then applying the found results to test data.
There are also several tweaks which will be discussed in the following chapter to optimise convergence and diversity of the algorithm.

3 Main Body

This section describes the proposed implementation of a genetic programming framework with Reinforcement Learning-based hyper-heuristics for financial forecasting. Also the results will be provided and compared to different state-of-the-art methods to predict financial trends.
The implementation integrates precomputed indicators, custom GP operators, and adaptive hyper-parameter tuning to address gaps in existing methodologies. It also uses different methods to optimize the algorithm to provide a larger search space and better convergence onto good results.
The methodology, experimental setup, results, and analysis will be discussed in the following subsections.

3.1 Methodology

Problem Definition

This implementation aims to forecast stock prices, with a specific focus on early detection of market trends to enable preemptive trading actions that maximize profitability.

Genetic Programming Framework

Operator	Number of Operands	Function
Addition	2	a + b
Subtraction	2	a - b
Multiplication	2	a * b
Safe division	2	a / b
Safe power	2	abs(a) ** b
Safe exponentation	1	np.exp(x)
Safe logarithm	1	math.log(abs(a))
Sinus	1	math.sin
Cosinus	1	math.cos
Mean	2	(a + b) / 2
Standard deviation	2	math.sqrt(((a - b) ** 2) / 2)
Clamp	3	max(min(a, max), min)
Compare	3	0 if a - b $<$ threshold elif a $>$ b 1 else -1

The primitives of the genetic algorithm present nodes in the tree structure which do not terminate the tree but take parameters for their computation. These mostly include basic mathematical and also more complex ones like logarithms, exponential functions, square functions and also trigonometric functions. Functions like division or squares are kept safe with returning default values for invalid inputs. Other functions that are also included are logical functions like a compare function to check whether two values differ for a given value.

Terminal	Parameters
Momentum	'period': [5, 10, 20]
Moving average	'period': [10, 20, 50]
Volatility	'period': [10, 20, 50]
RSI	'period': [14, 20, 30]
MACD	'short': [10, 12], 'long': [26, 30], 'signal': [9, 10]
ATR	'period': [10, 14, 20]
Williams %R	'period': [10, 14, 20]
Bollinger Bands	'period': [20, 30], 'stddevmult': [2, 2.5, 3]
ADX	'period': [14, 20, 30]
CMF	'period': [20, 30, 40]
Constant	-100 $<$ x $<$ 100

The terminals provided are all the indicators which are derived mostly from the closing price. To keep computation more simple common values have been tested and best performing parameters were chosen to precompute these indicators.
Therefore several metrics that will be used to evaluate the performance of the strategy. All these metrics will come together in the fitness function.
First a target function is computed on the data set. Here a simple moving average crossover function is used. A notable aspect is that the function leverages next-day prices for action predictions, simulating informed decision-making. By using this method the function not only knows future data to make informed decisions but it also does not trade daily which avoids highly frequent and volatile trading. This is useful because it would result in the strategy overfitting to the data set instead of learning to make meaningful decisions.
The Sharpe Ratio, a key performance metric, quantifies risk-adjusted returns by computing the excess return relative to a risk-free rate. As risk free rate a 3 percent annually return rate will be used, which is expected to be raised without committing to any risk like investing into government bonds. The mean return rate minus the risk free rate is then weighted with the standard deviation of the returns. While negative values mean not investing would be better also ratios between zero and one can be seen as bad results. The range one to two can be seen as acceptable results and a value between two and three can be seen as good. Everything above three can be seen as exceptional great performance.
Although the Sharpe Ratio accounts for losses, large drawdowns are particularly undesirable for traders. Consequently, the third metric included is the maximum drawdown in the strategy's returns.
The last metric that will be used is the complexity of the strategy function. While this not only ensures the explainability of the function, it also reduces the risk of overfitting to the data set.
Pareto optimization ensures balanced performance by selecting solutions where no single fitness objective improves without degrading another, ensuring balanced optimization across objectives. This means after the fitness function assigns an array of fitness values to the individuals, the selection method extracts a pareto front of them. This ensures the selection of all non-dominated individuals within the population. Non-dominated individuals are all strategies which can just be beaten in one of the fitness values with sacrificing another value.
This method is appropriate because combining the fitness values in one weighted value also needs correctly defined weights. While these can manually be set, wrong values can result in worse behavior of the algorithm. Thus

it is better to use the pareto front to guarantee all objectives are getting correctly optimized.
While selection ensures better individuals get picked, the best individuals can still get changed through mutation methods later on. Therefore the best individual gets stored in the hall of fame to ensure the best individual can be returned in the end.

No.	Individual
1	Difference between short-term and long-term momentum
2	Difference between short-term and long-term moving averages
3	Divergence between RSI and momentum for overbought/oversold detection
4	Combines stochastic oscillator (Williams %R) with momentum
5	Volatility to normalize RSI
6	Combines volume and momentum for identifying divergence
7	Combines trend-following (MA) with momentum (RSI)
8	Weighs trend strength (ADX) against moving average crossover
9	Checks for breakouts between momentum and RSI
10	Identify breakout points where moving averages exceed volatility-adjusted levels
11	Detects price levels where average trends and momentum diverge from volatility
12	Detects volatile periods where short-term momentum and RSI exceed
13	Measures trend strength and reversal signals relative to short-term momentum
14	Identifies long-term trend shifts when moving averages cross adjusted RSI levels
15	Captures divergences between volatility and combined momentum and RSI
16	Moving average crossover alone and with bollinger low
17	Moving average crossover alone and with bollinger high
18	Moving average crossover with upper and lower BB
19	MACD signal and lower BB crossover
20	MACD signal and upper BB crossover
21	MACD signal and both BB crossover
22	MACD signal and RSI
23	MACD signal and MO MA crossover
24	MACD signal and williams
25	MACD signal and mean of ADX and Williams

Another aspect of the genetic algorithm, which needs to be defined is how the individuals get initialized. Therefore a combination of customly defined strategies and randomly created trees is used. For custom creation several different methods which have been shown to work in trading are used. These custom strategies can be seen the table above. The randomly created trees

are generated using the genHalfandHalf method. This method creates one half of trees with the maximum depth and the other half is grown from root to incorporate random tree lengths.

Reinforcement Learning-Based Hyper-Heuristics

For the hyper-heuristics framework four different low-level heuristics are used. The first one will be the mutation of the indicators to introduce more exploration of the search space. To also ensure exploitation another method that will be used is hill climbing. In the implementation a hill climbing method on the period parameters of the indicators with a step size of five will be used, which means that a maximum of five neighbors is checked whether they are better than the original strategy. Another more simple method is the period parameter mutation which is added to accelerate the computation compared to the computational cost of hill climbing. The fourth method which is used is the mutation of constants. Because constants can be used as terminals, it is also important for exploration to change these. For a default method there is also a method which does nothing. This method allows the algorithm to maintain individuals for stability, preserving high-quality solutions across generations.
The reinforcement agent in this implementation is a q-learning agent. The agent maintains a table of q-values, which represent the returned reward given a certain action in a specific state. The reinforcement learning agent employs an epsilon-greedy strategy to achieve an optimal balance between exploration and exploitation. The epsilon value is declining with every generation, resulting in more exploration in the beginning versus more exploitation at the end.
The agent is able to interact with the hyper-heuristics framework by learning to effectively update the probability for the low-level heuristics. It observes the current state and then selects the best action based on the q-table. After the action is executed, the reward is calculated and with it the q-value of the action gets updated. The reward is calculated with the change in the average Sharpe Ratio over every valid strategy. Strategies get marked as invalid if they trade less than five times or over 1500 times to prevent overfitting.
The benefit of using Reinforcement Learning compared to the approach in Kampouridis et al. [1] is that not only the update to the q-values is proportionate to the change in the Sharpe Ratio, also the agent is able to first explore all actions and then try to focus on optimizing the actions to retrieve the best strategy. Therefore it is also ensured that per individual in every

generation just one mutation method is applied.

Implementation Details

In this subsection first all the used libraries for the implementation are cited. Afterwards all relevant parameters will be provided in a table.
The first library that is used for retrieving all the important stock data is 'yfinance'. It offers a great Rest API to get all the important data from yahoo finance. The next library is 'deap', which is used for all the genetic algorithm parts of the implementation. It offers simple and efficient ways to build tree like individuals. The last important library to be mentioned is 'matplotlib'. With its help all the visualization coming up in the results chapter were created. The models which the implementation is tested against are imported from 'sklearn'.
Next up the hyper parameters chosen in the implementation will be discussed. The initial population size is kept relatively high at 200+ to ensure a high diversity. Also the initial tree depth is kept dynamically between one and four to ensure diversity. Trees with depth of over 4 would become extremely complex, therefore the maximum is not higher. The maximum generations are kept high with 60 to give the q-agent enough time to converge. Initially all the probabilities are the same with 0.2 to ensure no bias in the starting point and also the change value with +/- 0.04 and the limits at 2 are all the same. There is no problem if the sum of the probabilities is higher than one, because a random value is chosen between 0 and the sum of the probabilities. The state space consists of the average and the maximum Sharpe Ratio, the population diversity and the number of generations. The initial epsilon value is 1.12 to ensure the agent chooses randomly its action in the first four generations. The generation in which epsilon=0.5 is reached is the 21st generation, which is relatively early in the run time of the agent to ensure convergence can be reached. To still keep enough diversity in the actions chosen to adapt their q-values to the changing state space, a minimum value of 0.3 is chosen for epsilon.

No.	Individual
Population Size (n)	200 + number of custom individuals
Initial Tree Depth (min, max)	min=1 and max=4
Maximum Generations	60
Indicator Mutation Probability	Initialized at 0.2
Period Mutation Probability	Initialized at 0.2
Constant Mutation Probability	Initialized at 0.2
Hill Climbing Probability	Initialized at 0.2
No Operation Probability	Initialized at 0.2
Action Space	Increase or decrease pb
Factor of change for pb	+/- 0.04
Maximum value for pb	2 (minimum is 0)
State Space	avg sharpe, maximum sharpe, diversity, generation
Initial epsilon	1.12
Decrease factor for epsilon	0.03 (until minimum value of 0.3)

3.2 Experimental Setup

Dataset

In a first step the apple share was tested. But because of the nature of a technical index there was a quite high volatility. Also in terms of apple there is a specific trend in spiking prices in November because of the release of every new iPhone generation.
Thus we consider a more stable share in this implementation. Coca-Cola was chosen due to its representation of consumer market trends, offering a balance of stability and volatility. Another option would have been to select indices that directly display a large proportion of the market. But in this paper a more volatile environment is considered to have a more challenging market to predict.
While now a special index is chosen there are still several parameters to be identified.
The first question is in which frequency the stock price will be looked at. While a lot of data is required for the algorithm hourly stock prices would result in highly volatile prices which do not always align with market trends. Thus it will be looked at daily prices. To still ensure enough training data the time range to train on will be from 2014-2024.
There is still another question to be answered concerning which price to look at when using daily prices. For single prices the opening, the highest, the

lowest and the closing price can be considered. Another option is to use the mean value for the day. In this implementation the closing price will be chosen because it is also the standard benchmark of most forecasting methods. It also represents the final price the market agrees on. Compared to the opening it is better because the latter can be influenced by pre-market activity. Also close prices tend to be less volatile than the highest and lowest price. Also the mean price can be skewed by brief spikes and drops thus the closing price is the most definitive price.
An aspect to consider in financial markets are dividends of shares. Thus it is important to include these profits into the price of the share. Herefore an adjusted closing price is calculated which accounts for these values.
Another important step in feature engineering is to handle missing values. While it is mostly uncommon in current problems to just drop these rows of the data, this is actually the approach that will be used in this implementation. Most of the non defined values stem from calculations for moving averages. These values all arise in the beginning of the data and therefore dropping these rows does not result in any problems. Other non defined values are randomly distributed and are just a relatively small number so it is no problem to drop these too.

Experimental Design

Overfitting is a well-known challenge of different optimisation methods. Therefore a common approach is to use different data for training the algorithm and afterwards testing it. One method particularly suited for time-series data is an expanding window method. In this method the data gets sequentially trained on a subset of the data and then tested on data following in time. Afterwards the range is increased and the next test data after that will be used. This method also will be used in this application to calculate its accuracy. The expanding window method prevents overfitting by ensuring the algorithm is trained on increasingly large datasets while being tested on unseen data.
Other evaluation will be Sharpe Ratio, maximum drawdown and complexity of the strategy. All these metrics will be calculated on the whole data set.

Baselines

The baseline methods which the algorithm will be compared to include algorithms currently raising a lot of interest. These algorithms will be basic implementation of support vector machines, random forests and neural networks.
These algorithms will be kept simple because the main focus should remain on the Reinforcement Learning-based hyper-heuristics framework.
There will also be an implementation provided which is based on domain knowledge to optimize a sequential application of the different hyper-heuristics methods instead of using them in parallel with the Reinforcement Learning model. This model will also compared in the same metrics to the implementation.

Computational Environment

As a computational environment google colab will be used. Not only does it offer powerful computational resources, it also provides an ease of use. Especially it has a wide range of python libraries pre-installed which will be the go to language for this implementation. It also integrates seamlessly with jupyter notebooks to keep the code clean and also allows for great visualization methods.

3.3 Results

In this section first the results of the Reinforcement Learning-based hyper-heuristics framework will be demonstrated. Afterwards some modifications will be discussed and then the results of this advanced implementation will be shown. In the end the results will be compared to the baseline models.

Performance Basic

The run of the basic implementation of the Reinforcement Learning resulted in the output 'add(compare(F31, F10, 0.01), compare(F0, F6, 0.01))', which can be interpreted as the formula 'add(compare(Williams10, RSI20, 0.01), compare(MO5, VO10, 0.01))'. The function basically combines two different prediction indicators into one prediction.

The performance measurements of this best individual on the training data can be depicted in the table below. As it can also be observed in the function, it is not overly complex and achieves a very good Sharpe Ratio of 3.4 on the data. The accuracy of 41.6% seems to first be quite low, especially taking in consideration that random guessing should achieve an accuracy of 33%. But looking at the logs of the algorithm one can be observed that the highest accuracy of the population always remains around 50%. Therefore the selection method of the function has been changed to just choose the individuals with the highest Sharpe Ratio. Also a maximum drawdown of 6.47% is a great value especially for risk averse traders and given the fact that it was no longer included in the selection method. All these good values result in a large profit of 1697% on the training data.

Metric	Performance
Accuracy	0.4161585365853659
Sharpe Ratio	3.4016455359713555
Max Drawdown	0.064796
Complexity Penalty	4.19722457733622
Profit	16.979596

Taking a look at the table below a high imbalance of prediction values can be observed. It seems like the strategy is just trying to short the share at the right times. Also the figure 1 presents the prediction values over the last year. Here the signals have to be read as shorting after the sell signals and no position after the buy signals.

Signal	Frequency
Buy Signal	264
Sell Signal	265
No action	1937
Short Position	1556
No position	908
Long Position	3

Therefore a deeper look has been taken into the two components of the function onto the whole data set. When looking at the parts of the function one can be observed that the first part almost always has an output of -1. So the first part is responsible for the function just focusing on shorting the share. The second part forms the decision when the share has to be shorted,

Figure 1: Yearly predictions

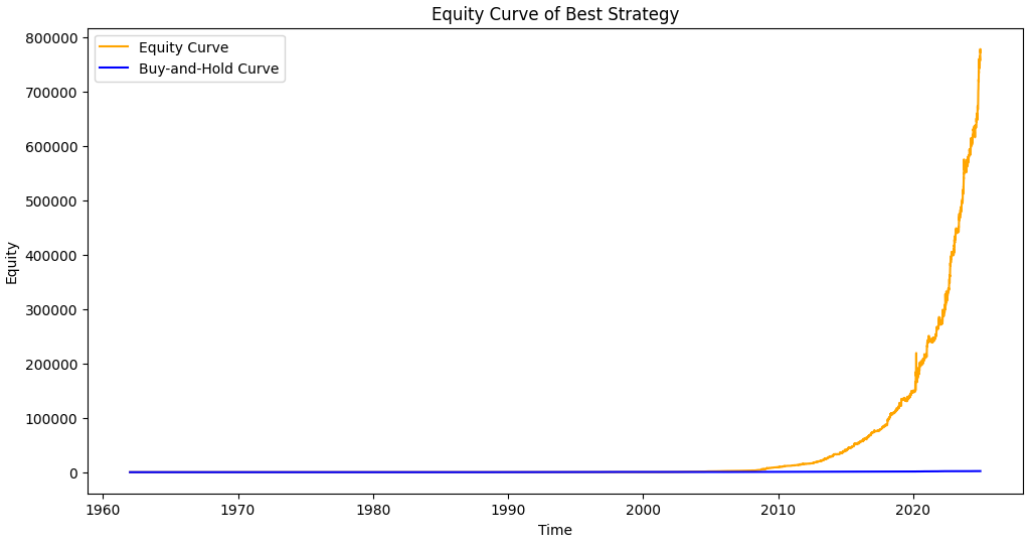
which can simply be extracted from its structure. The fewest predicted value is 1, which correlates to periods in which the 5-day momentum is higher than the 10-day volatility. It is important to note that there are almost the same amount of periods, where the momentum and volatility do not differ much, as periods, where volatility is higher than momentum. With volatility always returning a positive value, the strategy is always trying to sell the share if the momentum is negative. This can be overcompensating as well as the fact, that defaulting to also shorting the share if there is no significant difference between momentum and volatility.

Part of function	1 predicted	0 predicted	-1 predicted
compare(Williams10, RSI20, 0.01)	0	19	15783
compare(MO5, VO10, 0.01)	3456	5448	6898

Despite concerns about overfitting, the strategy demonstrates significant performance improvements on the testing dataset with a total return of about 80,000,000% over the period of 60+ years as it can be seen in figure 2. But it needs to be kept in mind, that this value can be respective to overfitting.

In figure 3 the reward values and the actions chosen by the algorithm can be seen.

Figure 2: Cumulative returns over the full dataset



Although it seems like no convergence is achieved concerning the reward function, it needs to be considered that all invalid strategies are not included in this value. Therefore the selection method may be to weak resulting in just filtering out strategies with invalid trade counts and not selecting just the best of the valid strategies.
From the chosen actions one can observe that the action 6 is chosen the most often. This is a good sign, because this action increases the probability of hill climbing being applied to the individuals. With hill climbing being the only method using local search it is the only method that just makes updates which improve the fitness function. Therefore it is great to observe that the algorithm focuses on the correct methods to ensure convergence.

Changes

After observing the results of the algorithm, several changes have been considered to improve the performance of the algorithm.
The first change is in the initialization of the population. In the last step it has been observed that there are always some invalid strategies in the population and Sharpe Ratio seems to play an important role for the strategies. To keep it simple yet effective hill climbing is applied to the initial population especially with the goal to replace invalid strategies with valid ones if this can be achieved by replacing parameters of the indicators.
Another change that has been applied is that the selection function is set

back to the pareto optimization as discussed in the beginning. Especially with hill climbing focusing on Sharpe Ratio, the fitness functions focus shall remain on all objectives.
Also there has been a change to the Reinforcement Learning part of the algorithm. Considering the weak convergence of the implementation before, higher initial probabilities are chosen for the not mutating methods to strengthen convergence.
Also considering the overly high focus on action 6 before the maximum value is reduced to 1 from 2. While this may result in actions with no impact at the end of the algorithm, it ensures that there will still be diversity in the end.

Performance Advanced

The function returned of the advanced algorithm is 'add(safelog(MACD103010macdline), add(MA50, Williams20))'. This function also uses two strategy components.

Signal	Frequency
Buy Signal	153
Sell Signal	153
No action	2159
Short Position	1076
No position	1341
Long Position	49

Part of function	1 predicted	0 predicted	-1 predicted
safelog(MACD103010macdline)	324	20	15458
add(MA50, Williams20)	3033	51	12718

With the safe version of the logarithm returning just the input for invalid inputs, the first part returns just buy signals if the input is larger than 1.11. With the macd line just being a crossover strategy, the first part can be viewed as if the crossover of the 10-day moving average and the 30-day moving average produces large signals for a long position also long position suggestions get returned. In other cases the first part almost always tries to short the share as it can be observed in the table above.
The second part of the function uses an addition of the 50-day moving average and the 20-day williams. This method can be interpreted to just return

the signal of 20-day williams, which indicates oversold for short positions and overbought conditions for long positions. It gets scaled by the 50-day moving average, which in terms of the rising trend of the price means that earlier on this part will create more short signals and later proportionally create more long signals. This part probably only works good on this share, because the price is always between 0 and 100 and the williams is between -100 and 0.
By combining both parts the function balances out a crossover method with the williams function.

Metric	Performance
Accuracy	0.3622967479674797
Sharpe Ratio	6.320831023840211
Max Drawdown	0.06440773650507481
Complexity Penalty	3.791759469228055
Profit	523.4212027528049

At first glance, this function may not appear promising, but it actually produces a very large Sharpe Ratio of 6.32 as it can be observed in the table above. Also the maximum drawdown and the complexity penalty are lower than the ones of the strategy of the implementation provided earlier. But the strategy seems to sacrifice points in accuracy for this performance. This behavior can surely be contributed to the pareto optimisation inside of the fitness function. Also the value of 52,342% profit on just the training data seems very impressive.
While the function seems to perform better and also shows a better balance between short and long positions, it does almost never choose no position as it can be observed in the table above. This also may result in a not optimal strategy because in indecisive situations it may be better to refrain from taking a position. But looking at figure 4, where periods after buy signals indicate long positions and periods after sell signals indicate short positions, it can be seen that the function indeed uses price trends great to its advantage.

Looking at figure 5 the action selection seems to still be focused on action 6, which is a good sign. But with the integrated changes the algorithm is able to still have a diversity at the end.

Looking at the reward function values at first look no difference to the previous implementation can be observed. But with the the hill climbing step

used on the initial population, the Sharpe Ratio values have already been at a better starting point. Therefore a better performance can be conducted from this implementation and also a better proportion of the search space may have led to the better performing result.

Comparison to baseline models on whole data

Model	Accuracy	Sharpe Ratio	Drawdown	Cumulative Return
GP 1	0.248829	7.633341	0.196721	841717.732912591
GP 2	0.198076	8.494254	0.196721	14112035.216463797
NN	0.300600	4.368057	0.493827	60513.560390272905
SVM	0.536744	4.127070	0.246914	3356.8750866312184
RF	0.439641	4.241017	0.493827	13157.110863980215

In the table above the results of both outputs of the different implementation are compared to the baseline models. As it can be observed all the baseline models do have pretty similar Sharpe Ratios of above 4, which is already a pretty high value. Also support vector machines seem to have the lowest maximum drawdown out of the baseline models. While the Sharpe Ratios do not differ much, it is important to note that already a slight increase leads to a much higher cumulative return over the period of 60+ years.
Not only do both implementations occur to have much higher Sharpe Ratios, they also both have lower maximum drawdowns than the baseline models. As it can also be seen in the charts of figure 6 these values result in much higher cumulative returns as the baseline models. For comparison there is also a chart without the cumulative return of the implementations.

3.4 Discussion

This section will provide a deeper interpretation of the result and take a broader look at the topic. Also the strengths and limitations of the implementations will be revisited.

Insights

The key finding of this work is that the Reinforcement Learning-based has great potential for financial forecasting. While the baseline models in this study were not optimized, the performance disparity between these models

and the hyper-heuristics approach is substantial.
Also the advancements used in the second implementation seem to improve the performance even further. Especially the still remaining reward indicates the great performance on the large search space.
The observed high Sharpe Ratios highlight the algorithm's risk-adjusted profitability, but further analysis on out-of-sample performance is necessary.
Also the algorithm was successful in identifying the correct actions to take for convergence. With the adjustment to focus more on exploration of the actions in the beginning, the algorithm is able to identify the correct actions. Then it uses these actions to ensure better solutions.

Strengths

While obtaining a large search space the algorithm still manages to propogate the most promising individuals. With this advantage it is able to find great solutions in the search space.
The algorithm provides options to set hyper parameters to adjust for different aspects. For example it is easy to shift the focus of the algorithm in the balance of exploration and exploitation. Also the diversity of the population can be easily controlled by setting the correct parameters.
Another strength of the algorithm is the implementation of the hall of fame. This feature makes sure that the best solutions get stored and can be retrieved after the algorithm has run through.

Limitations

But there are also several limitations faced in the implementation that need to be mentioned.
One of them is the computational cost. While the genetic algorithm base provided an efficient way for calculation, the computational overhead of the hyper-heuristic layer necessitates advanced hardware, limiting scalability for real-time applications.
Also the focus on just one share of the market may also have drawbacks. While the Coca-Cola share seems to be quite representative in terms of market conditions, there is still room for improvement, such as extending the approach to predict multiple shares. This advancement could negotiate problems like the integration of direct prices in the prediction function and is also able to lower the risk of overfitting.

4 Conclusions

In this chapter the main contributions will be summarized. Also an outlook is provided on what future work in the field can be done.

Summary

This work provides a robust framework for integrating Reinforcement Learning into hyper-heuristics, showcasing significant potential for improving algorithmic trading strategies.
The initial implementation employed Reinforcement Learning to optimize the probabilities associated with low-level heuristics, including various mutation functions and a period-based hill-climbing technique.
The second implementation has ameliorated this approach by using hill climbing already on the initialization of the population and using different probabilities for several low-level heuristics.
While the first implementation was already outperforming the baseline models, the second one did have a much more outstanding performance.

Contributions

The most important contribution of this approach is the Reinforcement Learning used to learn correct probabilities for all methods. It was able to learn which methods can be used to increase convergence. With incremental steps the algorithm is able to adaptively balance out exploration and exploitation in different stages.
Other key findings include the precomputational steps. One the one hand, these are the precomputed features to effectively reduce the search space, while still guaranteeing meaningful results. On the other hand applying hill climbing beforehand also ensured a much more meaningful search space.

Future Work

For further investigation different aspects of the implementation can be improved. One of them is to include more and different indicators to diversify the search space even more.
Another option would be to implement more different low-level heuristics.

While this is an effective approach to diversify the action space of the agent, it is important to keep in mind that the agent does not have many generations to learn the probabilities.
Moreover there can be experiments done with the data set. One simple improvement is using multiple different stock prices to lower the chance of overfitting.
Another analysis which is worth being looked at is whether the agent is able to adjust the probabilities in a dynamic environment, which can be given through different inputs throughout the time dimension of the data set. Future research should explore real-time adjustment of probabilities in dynamic market conditions, enabling the algorithm to adapt to shifting trends and volatilities.

References

[1] On the investigation of hyper-heuristics on a financial forecasting problem. *Annals of mathematics and artificial intelligence*, 68:225, 2013.

[2] Multi-period portfolio optimization using a deep reinforcement learning hyper-heuristic approach. *Technological forecasting and social change an international journal*, 198:122944, 2024.

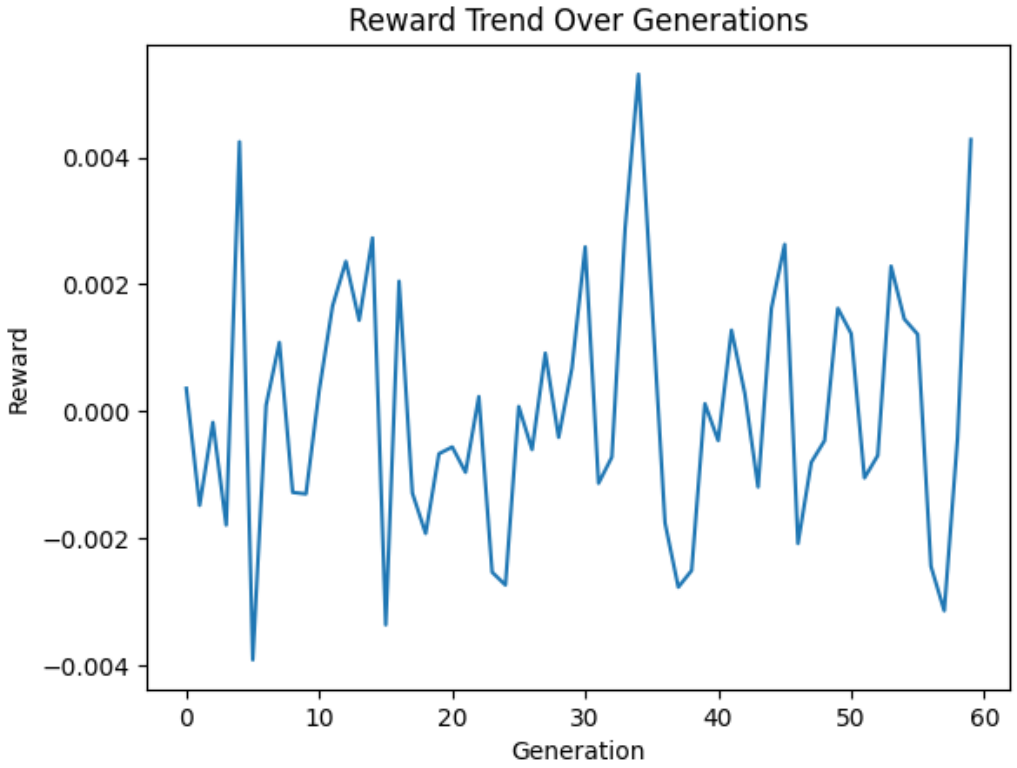
[3] Atika Gupta, Divya Kapil, Abhishek Jain, and Harender Singh Negi. Using neural network for financial forecasting. In *2024 5th International Conference for Emerging Technology (INCET)*, pages 1–6, 2024.

[4] John H. Holland. *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. The MIT Press, 04 1992.

[5] Sourabh Katoch, Sumit Singh Chauhan, and Vijay Kumar. A review on genetic algorithm: past, present, and future. *Multimedia tools and applications*, 80:8091–8126, 2021.

[6] J.R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. A Bradford book. Bradford, 1992.

Figure 3: Reinforcement Learning statistics



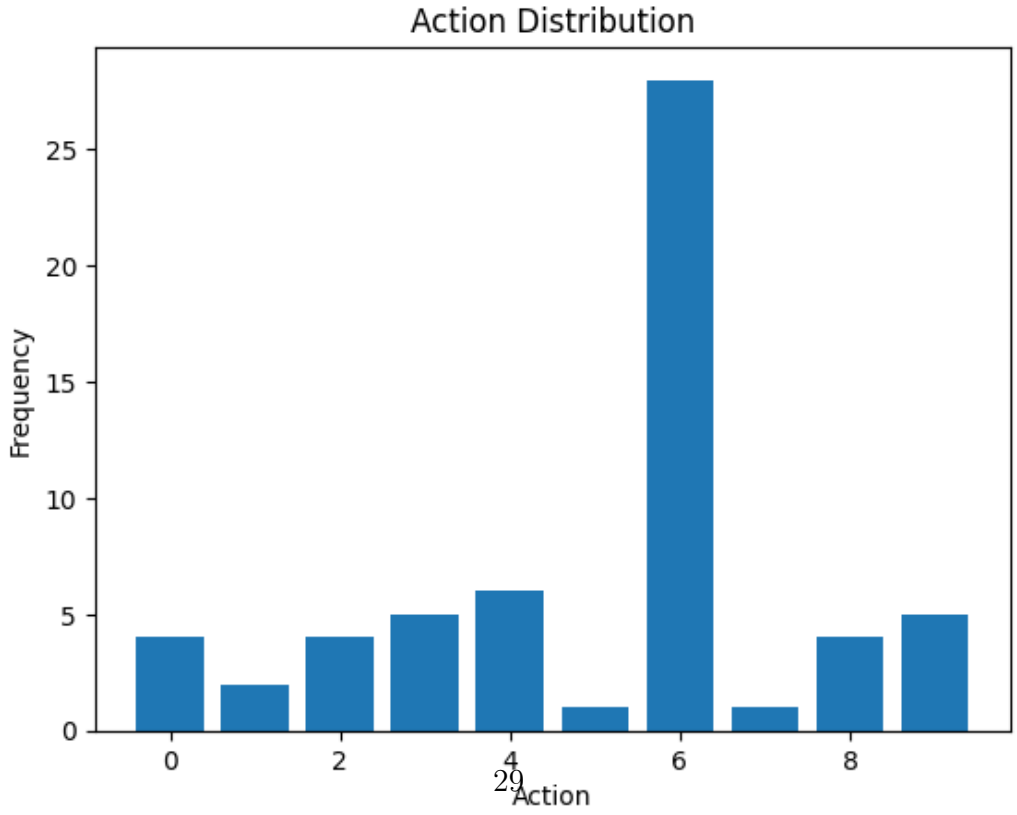


Figure 4: Predictions for this year

Figure 5: Reinforcement Learning statistics

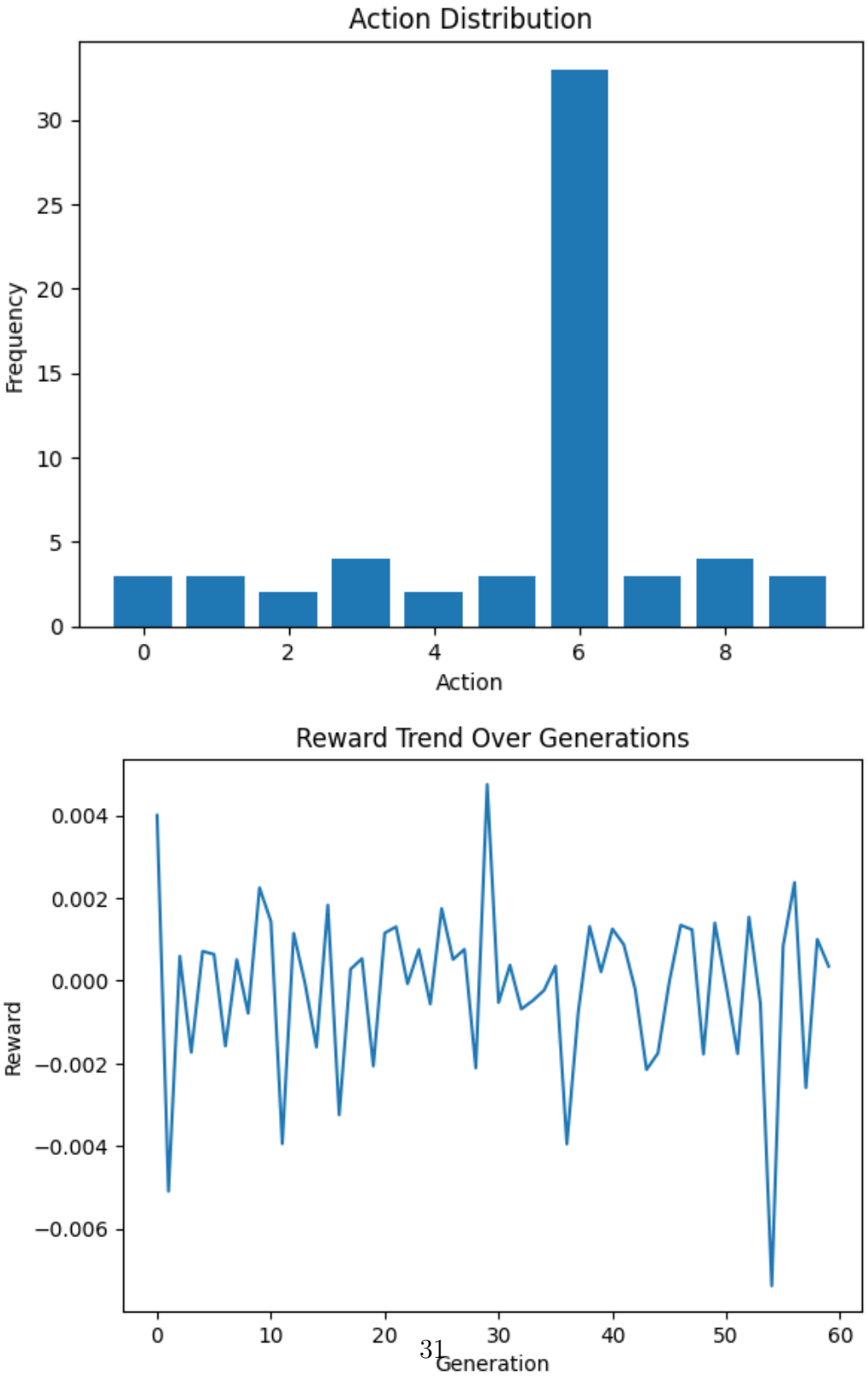


Figure 6: Comparison of cumulative return