Journal of Theoretical and Applied Information Technology
31st October 2023. Vol.101. No 20
© 2023 Little Lion Scientific

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

MULTI-AGENT PROXIMAL POLICY OPTIMIZATION FOR PORTFOLIO OPTIMIZATION

FIRDAOUS KHEMLICHI [1,*] , HIBA CHOUGRAD [1], SAFAE ELHAJ BEN ALI [1], YOUNESS IDRISSI KHAMLICHI [1]

[1] SIGER Laboratory, Sidi Mohamed Ben Abdellah University Fez, Morocco

E-mail: [1,*] firdaous.khemlichi@usmba.ac.ma

ABSTRACT

Deep reinforcement learning is a subfield of machine learning that combines the ideas of deep learning and reinforcement learning to enable agents to learn and make decisions in complex environments. It has been applied to a wide range of tasks, including gaming, robotics, and finance, among others. In finance, reinforcement learning (RL) has emerged as a promising technique for solving strategic decision-making problems in complex financial environments using reward-based approaches for optimal control. In this paper, we propose a novel algorithm that leverages the power of Multi-Agent Reinforcement Learning (MARL) coupled with Proximal Policy Optimization (PPO) to tackle the complex problem of portfolio optimization. What sets this approach apart is its utilization of MARL, which involves multiple agents learning and interacting within the same environment. This is in contrast to the traditional single-agent approaches commonly used in portfolio optimization. In portfolio optimization, MARL enables agents to learn from the interactions with other agents and the environment, leading to more realistic and robust investment strategies. The performance of the algorithm was assessed on the S&P 500 market using various numbers of agents and assets, and its performance was compared to several benchmarks. The performance metrics used for evaluation consisted of annual profit, annual volatility, Sharpe ratio, and Sortino ratio. The findings demonstrated that the algorithm outperformed the benchmarks in terms of all the performance metrics considered, regardless of the number of agents and assets involved.

Keywords: *Deep Learning, Reinforcement Learning, Portfolio Optimization, Proximal Policy Optimization.*

1. INTRODUCTION

Traditional methods of portfolio optimization can be limited due to the presence of correlations between price and other factors in financial environments, as well as the presence of substantial noise. Considering this, advanced machine learning techniques are presently being utilized in financial market dealings [1]. However, numerous models primarily utilize past asset prices to forecast future price movements through neural networks, which then enable trade agents to make decisions based on such predictions [2] [3] [4]. The notion appears logical, but the efficiency of these algorithms heavily relies on accurately forecasting future market prices. Consequently, some researches [5] [6] address this challenge by using reinforcement learning instead of predicting future prices.

Reinforcement learning involves an agent that interacts with the environment to acquire an optimal policy through trial and error for sequential decision-making tasks [7] [8]. The task of dynamic portfolio optimization is considered to be highly challenging in finance, as noted by researchers such as [9] [10]. It involves continuously adjusting the allocation of funds among various financial investment products, with the ultimate objective of achieving maximum return while minimizing risk. In order to address the problem of selecting the best stock portfolio, researchers have developed a reinforcement learning method based on hypergraphs to learn an effective policy for generating suitable trade actions, as described in [11]. In addition, an RL framework based on policies was presented for managing stock portfolios and its effectiveness was compared to alternative trading strategies [12].

In recent years, there's been recognition of the complexity in decision-making, especially in scenarios resembling multi-agent systems, where multiple agents make strategic decisions [13]. To tackle this, the Multi-Agent Reinforcement Learning (MARL) framework was developed for decision-making in shared environments [14]. MARL, akin to

Reinforcement Learning (RL), involves neural networks for agent decision-making [15], but adds complexity by considering agents' dynamics as part of the environment [16]. In financial markets, researchers consider MARL to enhance portfolio management and returns [17]. An alternative approach involves the Multi-Agent Simulator (MAS), a tool for assessing market activity and replicating market metrics [18].

Building upon this foundation, our paper addresses the critical challenge of portfolio optimization. The objective is to simultaneously maximize returns while minimizing risks in the context of complex and dynamic financial markets. To achieve this, we design an algorithm to solve the portfolio optimization problem, which involves maximizing returns while minimizing risks. It is a multi-agent reinforcement learning algorithm that uses neural networks as function approximators. The algorithm uses Proximal Policy Optimization (PPO) [19] as the training method, which is a state-of-the-art reinforcement learning algorithm that is known for its stability and ability to handle continuous action spaces. The exploration strategy used is Boltzmann exploration, which is a stochastic method that allows agents to select actions based on a probability distribution that depends on the estimated value of the actions. This allows the agents to explore the action space more effectively and find better solutions. The algorithm also uses Prioritized Experience Replay (PER) [20], which is a method that allows the algorithm to prioritize and replay important experiences more frequently, leading to faster learning and better performance. The algorithm also incorporates the TD error, which is a measure of the difference between the predicted value and the actual value of a state or action, to guide the learning process. Additionally, a shared memory mechanism is employed to facilitate communication and data sharing among the agents in the portfolio optimization task. In general, the algorithm is designed to learn optimal investment strategies for a given set of assets and constraints, by training multiple agents to work together in a cooperative manner. The use of PPO [19], Boltzmann exploration, PER [20], TD error, and a shared memory allows the algorithm to learn quickly and efficiently, while ensuring stability and robustness. Broadly, our contributions are as follows:

- The algorithm is based on a multi-agent reinforcement learning framework, which allows for better modeling of the complex interactions between different assets in the portfolio.
- The algorithm uses a deep neural network to learn the optimal portfolio allocation strategy, which makes it more flexible and adaptable to different market conditions.
- The introduction of a shared memory mechanism facilitates efficient communication and data sharing among agents. This enables agents to exchange information, synchronize their actions, and collectively learn from their experiences.
- The algorithm incorporates Boltzmann exploration, which balances the exploration and exploitation trade-off and leads to better long-term performance.
- The algorithm has been tested on real-world financial data, specifically the S&P 500 index, and has demonstrated superior performance compared to other traditional portfolio optimization approaches.
- The algorithm can be extended to potential applications in other financial domains, such as asset pricing, risk management and portfolio rebalancing.

The rest of the paper is structured as follows: Section 2 provides a concise overview of the concepts and fundamentals of reinforcement learning, Markov Decision Process, Shared Memory, Boltzmann exploration, and Prioritized Experience Replay. Section 3 details the proposed model Multi-agent PPO. The experimental results are presented in section 4. Section 5 is dedicated to the discussion, and the paper concludes with final remarks.

2. BACKGROUND

In this section, we briefly present the concepts and fundamentals of reinforcement learning, Markov Decision Process, Shared Memory, Boltzmann exploration, and Prioritized Experience Replay.

2.1 Reinforcement Learning

Reinforcement learning (RL) is a subfield of machine learning (ML) that focuses on teaching agents how to make optimal decisions based on the feedback they receive from their environment. The components that make up the RL framework are as follows:

- Agent: The agent is the entity that interacts with the environment and makes decisions. It can be thought of as a software program or an autonomous robot that has a set of actions it can take and a decision-making policy that determines which action to take given the current state of the environment.

• Environment: The environment is the external world in which the agent operates. It can be any system that the agent can sense and interact with, such as a video game, a robot, or a financial market. The environment has a set of states that the agent can perceive and rewards that it can receive.

• State: The state of the environment is a snapshot of the current situation the agent is in. It is a representation of the relevant information that the agent can perceive from the environment at a particular moment.

• Action: An action is the decision made by the agent based on the current state of the environment. The nature of actions can vary depending on the specific context and problem at hand. Some actions may involve continuous control over parameters, while others may involve discrete choices from a predefined set of options.

• Reward: The reward is a numerical value that the agent obtains from the environment as a consequence of its action. It represents the value of the action taken and can be positive, negative, or neutral.

• Policy: The policy is the decision-making strategy used by the agent to select actions based on the current state of the environment. The goal of the agent is to learn an optimal policy that maximizes the total reward it receives over time.

• Value Function: The value function is a function that estimates the expected total reward the agent will receive from a specific state or state-action pair.

In RL, the agent interacts with the environment through a series of distinct time steps. During each time step, the agent receives the current state of the environment, decides on an action to take based on its policy, receives a reward, and transitions to a new state. The goal of the agent is to learn a policy that maximizes the expected total reward it receives over time. (See Figure 1).

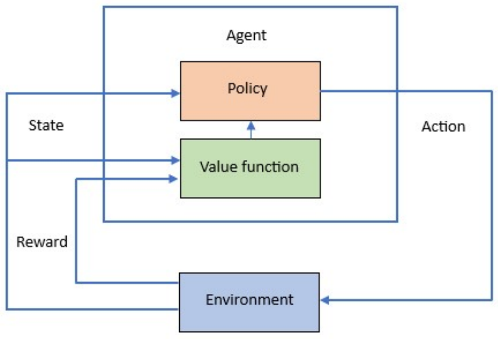


Figure 1: Reinforcement Learning Process

There are several RL algorithms, each with its own advantages and disadvantages, but they all follow the basic RL framework outlined above. Some of the most popular RL algorithms include Q-learning [21], actor-critic methods [22], deep Q-networks (DQNs) [15], and policy gradient methods.

In general, RL is a powerful technique that has been used to solve a wide range of problems, from playing video games to controlling robots to optimizing financial portfolios. It has the potential to revolutionize many fields by enabling machines to make intelligent decisions in complex environments. Our article specifically explores the application of reinforcement learning in the domain of portfolio optimization.

2.2 Markov Decision Process

Markov decision process (MDP) [23] is a framework for modeling decision-making problems in which an agent interacts with an environment over a series of discrete time steps. MDPs are widely used in reinforcement learning, where an agent learns to make optimal decisions by maximizing a long-term reward signal. In a single-agent MDP, the environment is characterized by a set of states S, a set of actions A, a transition function T, a reward function R, and a discount factor γ. For simplicity, we omit the state transition function since our work assumes that the agent's actions do not impact state transitions. The modeling of our MDP is as follows:

• The state space could include the current portfolio weights, asset returns, and other relevant market data. The current state of a stock is characterized by a set of features that provide relevant information about its performance. To achieve this, we leverage a historical sequence of the closing prices of the particular company over a period of n days.

• The action space : The available options for action encompass purchasing, selling, or holding shares of various assets.

• The reward function could be based on the change in portfolio value over time, taking into account transaction costs and other constraints. In our paper, a reward is determined by the action of a company at its current state and the return it generates on the following day. This reward is used to evaluate the performance of the agent.

• A policy, denoted as π(s, a), refers to the trading strategy adopted by an agent in a given state s. It can be thought of as a probability distribution of

actions that an agent takes in response to a particular state.
The primary objective of an agent is to determine the optimal policy that results in the highest cumulative rewards over time.

Once the MDP is defined, a reinforcement learning algorithm could be used to learn an optimal policy for portfolio optimization.

2.3 Multi-Agent Reinforcement Learning Extension

In multi-agent reinforcement learning, the MDP tuple is extended to include information specific to the multi-agent setting. The MDP tuple can be defined as follows:

• Joint State Space : A joint state space that captures the state of all agents and the environment.

• Joint Action Space : Each agent in the MARL framework selects an action from its individual action space, which could include different investment decisions or portfolio allocations.

• Joint state transition function : A joint state transition function that maps a joint state and joint action to a joint next state. In the multi-agent setting, the joint action is a tuple that consists of the individual actions taken by each agent involved in the system. Similar to the single-agent scenario, the multi-agent case also omits the explicit representation of the state transition function.

• Joint Reward Function : Reward functions for each agent that map a joint state and joint action to a reward received by that agent.

• The discount factor γ : A discount factor that determines the importance of future rewards.

In summary, the MDP tuple in multi-agent reinforcement learning extends the classical MDP tuple to include information specific to the multi-agent setting, such as joint state spaces and joint action spaces, as well as reward functions for each agent. Nonetheless, there are several fundamental issues that arise when using MARL in the context of portfolio optimization:

• Exploration-exploitation trade-off: The agents need to balance exploration (trying new strategies) and exploitation (using the current best strategy) to find optimal solutions. In portfolio optimization, this trade-off is particularly important because the agents need to balance risk and reward.

• Scalability: As the number of assets in the portfolio increases, the dimensionality of the problem increases as well, making it more difficult for the agents to learn optimal strategies.

• Agent Coordination and Communication: In MARL, multiple agents need to collaborate and coordinate their actions to optimize the portfolio collectively. Designing effective communication and coordination mechanisms between agents becomes crucial for achieving desirable outcomes and avoiding suboptimal solutions.

In the subsequent sections, we will elaborate on our approach to resolving these challenges and demonstrate how we can guide agents to act distinctively while concurrently optimizing their returns.

2.4 Shared Memory

In our algorithm for portfolio optimization using the MARL framework, we utilize a shared memory structure to facilitate seamless communication and efficient data sharing among multiple agents. The shared memory, denoted as M, is constructed as a collection of shared variables represented as $M = \{m_1, m_2, \dots, m_n\}$. Each shared variable, m_i is designed to capture and store a specific aspect of the system's state or information that is accessible to all agents involved in the optimization process.

This representation allows us to track the evolution of the shared variables over time, providing agents with access to the most up-to-date information and enabling them to make informed decisions.

Agents in the system can read from the shared memory by retrieving the current value of a shared variable m_i at a given time t. This read operation, grants agents access to the shared information, such as historical asset prices, portfolio weights, or market indicators. Additionally, agents can update the shared memory through the write operation, where they modify the value of a shared variable m_i to reflect new information or state transitions, such as adjusting portfolio weights based on new market data or learning updates from the reinforcement learning algorithm.

The shared memory acts as a central medium for agents to exchange and synchronize data, facilitating coordination among them. Agents can share observations, policy updates, and other relevant information through the shared memory, allowing them to collectively learn and adapt their policies for portfolio optimization. By leveraging the shared memory M, agents can effectively communicate and collaborate, leading to improved decision-making and overall performance in the multi-agent system.

2.5 Boltzmann exploration

The Boltzmann exploration is a reinforcement learning technique that is used to balance the exploration-exploitation tradeoff in an agent's decision-making process. It is particularly useful in situations where there are a large number of potential actions, and the optimal action is not immediately obvious. The Boltzmann distribution is used to select actions based on their estimated values and a temperature parameter that controls the level of exploration. The temperature parameter is gradually decreased over time to encourage exploitation as the agent learns. In the context of portfolio optimization, using Boltzmann exploration can help the algorithm to explore a diverse range of portfolio weights during the learning process. This is important because the optimal portfolio weights may not be immediately obvious and may require some exploration. By using Boltzmann exploration, the algorithm is more likely to discover a wider range of potential solutions, which can ultimately lead to better performance in terms of maximizing returns and minimizing risk. Additionally, Boltzmann exploration can help the algorithm avoid getting stuck in local optima, which is a common problem in optimization problems. The probabilities of selecting different actions are determined by the Boltzmann distribution.

In the context of reinforcement learning, the Boltzmann exploration equation can be written as:

$$p(a_t, s_t) = \frac{e^{\frac{Q(s_t,a_t)}{\tau}}}{\sum_a e^{\frac{Q(s_t,a_t)}{\tau}}} \qquad (1)$$

Where $p(a_t, s_t)$ represents the probability of selecting action a_t in state s_t, $Q(s_t, a_t)$ represents the Q-value, which is an estimate of the expected cumulative reward when taking action a_t in state s_t. It measures the quality or value of an action in a particular state and τ is the temperature parameter, which controls the level of exploration versus exploitation. The Boltzmann exploration equation gives a probability distribution over the set of possible actions, with the probability of each action being proportional to the estimated value of that action divided by the temperature parameter. When the temperature parameter is high, the exploration is more random, and the agent is more likely to select actions with low estimated value. When the temperature parameter is low, the exploration is more deterministic, and the agent is more likely to select actions with high estimated value.

2.6 Prioritized Experience Replay

Prioritized Experience Replay (PER) [20] is an extension of the classic Experience Replay technique commonly used in reinforcement learning algorithms. The main idea behind Experience Replay is to store past experiences (i.e., transitions) of an agent in a replay buffer and randomly sample them during training to break correlations between successive transitions and improve learning efficiency. However, not all experiences are equally informative for learning, and some may be more important than others. PER aims to address this issue by assigning priorities to experiences in the replay buffer based on their expected learning potential and sampling transitions with higher priority more frequently. The priorities of experiences are typically determined based on the magnitude of their temporal-difference (TD) errors, which measure the difference between the predicted value of a state-action pair and the actual reward received from it. The intuition behind this is that transitions with higher TD errors are more surprising and unexpected, and therefore carry more information for the agent to learn from.

Here are the steps involved in PER with TD error:

1.Initialize a replay buffer of size N and set the priority of all transitions to 1.

2.Collect experiences by interacting with the environment using the current policy. For each transition (s, a, r, s'), calculate the TD error as follows:

$$\delta = |r + \gamma V(s') - V(s)| \qquad (2)$$

where V is the agent's value function and γ is the discount factor.

3. Add the transition to the replay buffer with its priority set to $p = (\delta + \varepsilon)^{\alpha}$, where ε is a small positive constant to ensure that no transition has zero priority and α is a hyperparameter that controls the degree of prioritization.

4.Sample a batch of transitions from the replay buffer based on their priorities. The probability of selecting a transition with priority p(i) is given by:

$$\boldsymbol{p}(i) = \frac{p_i^{\beta}}{\sum_k p_k^{\beta}} \qquad (3)$$

where β is a hyperparameter that controls the degree of importance sampling correction and helps to balance the sampling bias caused by prioritization.

5.Train the agent using the sampled batch of transitions. During training, calculate the loss for each transition as usual, but weight the loss by the inverse of its sampling probability:

$$\mathcal{L} = \frac{1}{N} \sum_i \frac{1}{P(i)} l_i \qquad (4)$$

where l_i is the loss for the i^{th} transition.

6. After training, update the priorities of the transitions in the replay buffer based on their TD errors. The new priority of transition i is set to $p_i = (\delta_i + \varepsilon)^{\alpha}$.

PER with TD error has been shown to improve the sample efficiency and learning performance of reinforcement learning algorithms by allowing the agent to focus on the most informative experiences.

3. MULTI-AGENT PPO

Our algorithm Multi-Agent Proximal Policy Optimization is an algorithm designed for portfolio optimization using deep reinforcement learning, specifically using the Proximal Policy Optimization (PPO) algorithm [19] with prioritized experience replay [20]. PPO is a deep reinforcement learning algorithm that is widely used for training agents in environments with continuous action spaces. The PPO algorithm is particularly well-suited for portfolio optimization because it can handle a large number of assets and agents.

The algorithm begins by initializing the shared memory, which consists of a set of shared variables capturing different aspects of the system's state and information accessible to all agents.

Next, the algorithm initializes a population of agents, each with its own policy network and value network. The policy network is responsible for selecting actions based on the observed state, while the value network estimates the expected returns associated with different state-action pairs.

During the training process, the agents interact with the environment and collect experiences. At each time step, an agent observes the current state and selects an action based on its policy network. After selecting an action, the agent proceeds to execute it within the environment.

Subsequently, the agent receives a reward based on its action and gains observations regarding the subsequent state of the environment.

The experiences are stored in a prioritized experience replay buffer, which maintains a collection of transitions along with their corresponding priorities. The prioritization is based on the TD error, which quantifies the discrepancy between the predicted and observed values.

To update the policy and value networks, the algorithm employs the Proximal Policy Optimization (PPO) algorithm.

The PPO algorithm maximizes a surrogate objective function that balances the trade-off between policy improvement and stability. The objective function is optimized using stochastic gradient ascent/descent, where the gradients are estimated using the collected experiences from the replay buffer.

In addition, the algorithm incorporates Boltzmann exploration to encourage exploration of the action space. Boltzmann exploration selects actions according to a probability distribution based on the Q-values or action preferences. This promotes diversity in the agents' actions, allowing them to explore different strategies and avoid being stuck in local optima.

In general, the algorithm aims to optimize the portfolio by maximizing the cumulative rewards over time. Through the integration of PPO, MARL, Prioritized experience replay, and the shared memory mechanism, it enables agents to learn and adapt their policies based on their collective experiences, improving the decision-making process and the overall performance of the portfolio.

Using PPO, prioritized experience replay, shared memory, neural networks, and Boltzmann exploration in our reinforcement learning algorithm aims to provide several benefits, including:

•Improved convergence: PPO is a variant of the traditional policy gradient algorithm that allows for more stable and efficient convergence to optimal policies.

•Faster learning: Prioritized experience replay allows for more efficient use of experience data, which can speed up the learning process.

•Improved exploration: Boltzmann exploration encourages the agent to explore the action space more thoroughly, leading to better overall performance.

•Better performance: By combining these techniques, the resulting algorithm can often achieve better performance on complex tasks compared to using each technique individually.

•Effective Communication and Collaboration: The shared memory facilitates communication and data sharing among the agents. It enables them to exchange information, such as updated network parameters, observations, or learned insights. This promotes collaboration and coordination among the agents, leading to collective learning and improved performance.

The pseudo-code of our algorithm can be described as follows:

Algorithm. Multi-agent PPO

```
Initialize the shared memory M.
Initialize policy and value networks for each agent.
Repeat until convergence:
    For each agent :
        Select action $a_t$ based on current state $s_t$ using the policy network.
        Execute action $a_t$ in the environment.
    Observe new states $s_{t+1}$ and corresponding rewards $r_t$.
    Compute TD errors $\delta_t$ and advantage estimates $A_t$.
    Update value networks:
        Calculate value predictions $V_t$ for states $s_t$ using the value network.
        Calculate value loss using the TD errors and a loss function.
        Update the value network parameters using gradient descent:
            $\theta_v \leftarrow \theta_v - \alpha\nabla_{\theta_v} L_{value}$
    Update policy networks:
        Calculate action log probabilities $\pi(a_t|s_t)$ using the policy network.
        Calculate surrogate objective $L_{surrogate}$ using the advantage estimates and action log probabilities.
        Update the policy network parameters using gradient ascent:
            $\theta_\pi \leftarrow \theta_\pi + \alpha\nabla_{\theta_\pi} J(\theta_\pi)$

    Compute Boltzmann exploration probabilities $P(a_t|s_t)$ using the Boltzmann equation:
            $p(a_t, s_t) = \frac{e^{\frac{Q(s_t,a_t)}{\tau}}}{\sum_a e^{\frac{Q(s_t,a_t)}{\tau}}}$

    Add the transition $(s_t, a_t, r_t, s_{t+1})$ to the replay buffer with its priority set to $p = (\delta_t + \varepsilon)^\alpha$.
    Sample a mini batch from the replay buffer based on priorities.
    Update the value networks and policy networks using the sampled mini batch.
    Update the priorities of the sampled transitions in the replay buffer based on the TD errors.
```

4. EXPERIMENTAL RESULTS

4.1 Dataset

The S&P 500 is an index of 500 large American companies, based on their market capitalization. We selected a diverse range of stocks from various sectors to create a portfolio. Our dataset covers the period from January 2010 to December 2022, consisting of a total of 3272 rows. We divided the data into a training set of 70% (2010-2018) and a testing set of 30% (2019-2022). By training our RL agent on this large span of time, it can learn optimal strategies for both bearish and bullish markets and adapt to different market scenarios.

4.2 Performance indicators

We compared the performance of our algorithm against the following benchmark methods:

- **Equal Weight Baseline (EW) -** is a simple investment strategy that involves equally distributing investment funds across all assets in a portfolio. Under this approach, each asset in the portfolio receives the same weight or proportion of investment, regardless of its individual risk and return characteristics. The goal of this strategy is to achieve diversification across assets and reduce concentration risk.
- **S&P 500 -** The S&P 500 is an equity market benchmark comprising 500 prominent publicly traded companies in the United States. It is highly regarded as one of the most extensively tracked equity indices globally and is often used as a barometer of the US stock market's overall health. The companies in the index are selected based on market capitalization, liquidity, and other factors, and the index is weighted by market capitalization. The S&P 500 is often used as a benchmark for portfolio performance and is used to track the performance of the US stock market as a whole.
- **Weighted Moving Average Mean Reversion (WMAMR) -** is a baseline strategy used in finance and investment. It is based on the idea that the price of an asset will eventually revert to its mean, and that an investor can profit from this by buying when the price is below the moving average and selling when it is above. The WMAMR strategy calculates the moving average of the asset's price over a certain period of time, and then takes a position in the asset based on whether the current price is above or below the moving average, with weights assigned to each period of the moving average. This weighting scheme allows more recent prices to have a greater influence on the decision to buy or sell than older prices.

• **Integrated Mean-Variance Kd-Index Baseline (IMVK) -** is a baseline algorithm used to compare the performance of portfolio optimization algorithms. It is a type of mean-variance optimization, which seeks to maximize the expected return of a portfolio while minimizing its risk, as determined by the variance of the portfolio's returns.

The IMVK algorithm integrates both mean-variance optimization and kd-indexing, which is a technique used to quickly retrieve data from a large dataset. It constructs an efficient frontier of optimal portfolios using kd-indexing, and then applies mean-variance optimization to select the portfolio with the highest Sharpe ratio (a measure of risk-adjusted return).

5. DISCUSSION OF RESULTS

5.1 Performance Analysis

We start by training a multi-agent reinforcement learning (MARL) using Proximal Policy Optimization (PPO) algorithm on historical data of the S&P 500 index. We divided the data into a training set and a testing set, with the former used to train the MARL agents and the latter used to evaluate their performance. In the experiments, we considered portfolios with 5, 10, and 20 assets, respectively. For each portfolio size, we trained a MARL agent with 10, 20, and 30 individual agents, respectively. For each portfolio, we use the same hyperparameters and training parameters to ensure a fair comparison between the results.

The number of assets in a portfolio can have a significant impact on portfolio performance. The more assets there are in a portfolio, the more complex the decision-making process becomes for the MARL algorithm, as there are more possible combinations of actions that can be taken. Therefore, testing our algorithm on portfolios with varying numbers of assets can help us understand how well it can handle increasing levels of complexity and diversification. Additionally, testing on portfolios with different numbers of assets can help us identify the optimal number of assets for a given market or investment strategy. In addition, training a MARL algorithm with different numbers of individual agents can have a significant impact on the results. Increasing the number of individual agents in the MARL algorithm can increase the complexity of the problem and improve the agent's ability to capture the interactions between different assets. This can lead to better performance and higher returns in the portfolio. However, increasing the number of agents also increases the dimensionality of the action space, which can lead to slower convergence and increased training time. Additionally, increasing the number of agents can also increase the likelihood of overfitting to the training data, especially if the number of training samples is limited.

In our study, we applied our algorithm to portfolio optimization with different numbers of agents and assets and evaluated its performance based on annual profit, annual volatility, Sharpe ratio, and Sortino ratio.

To evaluate the performance of our reinforcement learning (RL) agent against standard portfolio optimization techniques, we selected five non-RL based benchmarks. These benchmarks include Equal Weight Baseline (EW), S&P 500, Weighted Moving Average Mean Reversion (WMAMR), and two variants of Integrated Mean-Variance Kd-Index Baseline (IMVK), namely Moderate and Aggressive Strategy, which were discussed earlier. We compared the performance of these benchmarks against each other and our RL agent.

We first tested the algorithm on a 5-asset portfolio consisting of AAPL, FB, GOOGL, JPM, and XOM. The results demonstrated that our algorithm outperformed the benchmarks in terms of Annual profit, Sharpe ratio and Sortino ratio as shown in figure 2 and in the tables 1, 2 and 3.

Table 1 : the portfolio performance of our algorithm using 10 agents across different asset configurations, including 5 assets, 10 assets, and 20 assets.

10 agents	**Portfolio balance**	**Annual profit**	**Annual volatility**	**Sharpe ratio**	**Sortino ratio**
5 assets	27576	0.439	0.259	1.75	2.01
10 assets	30869	0.521	0.254	2.24	2.48
20 assets	37517	0.687	0.253	2.82	2.92

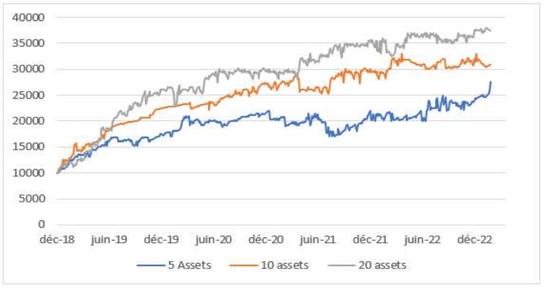
Table 2 : the portfolio performance of our algorithm using 20 agents across different asset configurations, including 5 assets, 10 assets, and 20 assets.

20 agents	**Portfolio balance**	**Annual profit**	**Annual volatility**	**Sharpe ratio**	**Sortino ratio**
5 assets	30604	0.515	0.253	1.96	2.18
10 assets	34875	0.621	0.252	2.45	2.69
20 assets	47702	0.942	0.249	2.97	3.14

Table 3 : the portfolio performance of our algorithm using 30 agents across different asset configurations, including 5 assets, 10 assets, and 20 assets.

30 agents	Portfolio balance	Annual profit	Annual volatility	Sharpe ratio	Sortino ratio
5 assets	37563	0.689	0.251	2.35	2.67
10 assets	41547	0.788	0.248	2.48	2.83
20 assets	53621	1.09	0.248	3.15	3.38

For the 5-asset portfolio, the MARL agent with 10 individual agents achieved an average annual profit of 0.439, with a Sharpe ratio of 1.75, and
indi
of

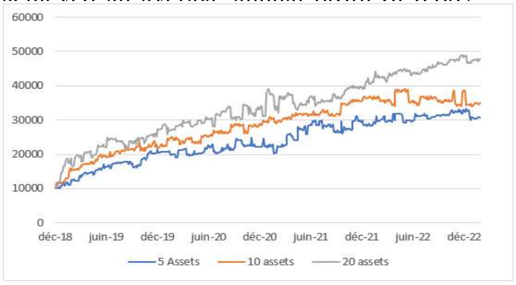


(a) : The portfolio balance achieved by employing 10 agents for different numbers of assets, specifically 5, 10, and 20.

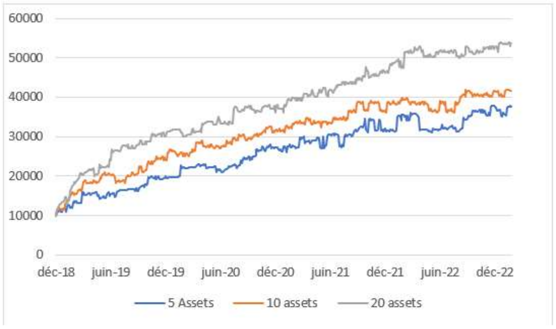
ratio of 2.18. Finally, the agent with 30 individual agents achieved an average annual profit of 0.689, with a Sharpe ratio of 2.35, and a Sortino ratio of 2.67.

We then tested the algorithm on a 10-asset portfolio consisting of the same five assets as before, plus five additional assets: AMZN, BRK-B, JNJ, MSFT, and PG. For the 10-asset portfolio, the MARL agent with 10 individual agents achieved an average annual profit of 0.521, with a Sharpe ratio of 2.24, and a Sortino ratio of 2.48. The agent with 20 individual agents achieved an average annual profit of 0.621, with a Sharpe ratio of 2.45, and a Sortino ratio of 2.69. Finally, the agent with 30 individual agents achieved an average annual profit of 0.788, with a Sharpe ratio of 2.48, and a Sortino ratio of 2.83.

Next, we tested the algorithm on a 20-asset portfolio consisting of the same 10 assets as before, plus another 10 assets: ADBE, BAC, CMCSA, CVX, DIS, GE, JPM, KO, PFE, and T. For the 20-asset portfolio, the MARL agent with 10 individual agents achieved an average annual profit of 0.687,
with a
2.92. T



(b) : The portfolio balance achieved by employing 20 agents for different numbers of assets, specifically 5, 10, and 20.



(c) : The portfolio balance achieved by employing 30 agents for different numbers of assets, specifically 5, 10, and 20.

Figure 2: The portfolio balance achieved for different configurations, including 5 assets with varying numbers of agents (10, 20, and 30), as well as 10 and 20 assets with the same agent numbers.

an average annual profit of 0.942, with a Sharpe ratio of 2.97, and a Sortino ratio of 3.14. Finally, the agent with 30 individual agents achieved an average annual profit of 1.09, with a Sharpe ratio of 3.15, and a Sortino ratio of 3.38.

The results showed that as the number of agents and the assets increased, the annual volatility decreased for all scenarios. For instance, when there

ISSN: 1992-8645 www.jatit.org E-ISSN: 1817-3195

were 10 agents and 5 assets, the annual volatility measured 0.259. However, with an increased number of agents to 30, the annual volatility decreased to 0.251. Alternatively, with 20 agents and 5 assets, the annual volatility recorded a value of 0.253. Notably, when there were 20 assets alongside 20 agents, the annual volatility further decreased to 0.249. This implies that incorporating multiple agents in the portfolio optimization process can contribute to risk reduction. Similarly, the inclusion of multiple assets can also aid in mitigating the risks associated with portfolio optimization.

Furthermore, as shown in tables 4, 5, and 6, we observed that our algorithm outperformed all the benchmarks, including the WMAMR and the two variants of IMVK. For example, when using 30 agents and 20 assets, our algorithm achieved an annual profit of 1.09, a Sharpe ratio of 3.15, and a Sortino ratio of 3.38, compared to 0.348, 1.45, and 1.72 for the WMAMR benchmark, and 0.312, 1.02, and 1.15 for the Aggressive IMVK benchmark.

These results indicate that our algorithm performs better with a larger number of agents and assets in the portfolio. The increase in the number of agents and assets allows for a better diversification of the portfolio and the agents are able to cooperate and learn from each other to achieve higher returns with lower risk. Additionally, the results demonstrate that our algorithm can achieve impressive results in terms of average annual profit, Sharpe ratio, and Sortino ratio, indicating its potential for real-world application in portfolio optimization.

There are several potential reasons for this improvement in performance as the number of assets increases. One possible explanation is that with more assets, there is a higher degree of diversification in the portfolio, which can help to mitigate risk and stabilize returns. Additionally, with more assets to choose from, there may be a greater likelihood of finding optimal trading strategies for each individual asset, leading to overall better performance for the

Table 4 : Comparison of the portfolio performance between our Multi-agent PPO algorithm and other benchmark strategies for a portfolio consisting of 5 assets.

	MAPPO, 10 agents	MAPPO, 20 agents	MAPPO, 30 agents	EW	Aggressive Strategy	Moderate Strategy	S&P 500	WMAMR
Portfolio balance	27576	30604	37563	22456	15329	16295	18342	16591
Annual Profit	0.439	0.515	0.689	0.311	0.133	0.157	0.208	0,164
Annual volatility	0.259	0.253	0.251	0.355	0.369	0.375	0.344	0.357
Sharpe ratio	1.75	1.96	2.35	1.63	0.63	0.58	1.26	1.15
Sortino ratio	2.01	2.18	2.67	1.82	0.87	0.76	1.51	1.45

Table 5 : Comparison of the portfolio performance between our Multi-agent PPO algorithm and other benchmark strategies for a portfolio consisting of 10 assets.

	MAPPO, 10 agents	MAPPO, 20 agents	MAPPO, 30 agents	EW	Aggressive Strategy	Moderate Strategy	S&P 500	WMAMR
Portfolio balance	30869	34875	41547	24819	17543	19784	22471	20146
Annual Profit	0.521	0.621	0.788	0.370	0.188	0.244	0.311	0.253
Annual volatility	0.254	0.252	0.248	0.351	0.366	0.371	0.339	0.351
Sharpe ratio	2.24	2.45	2.48	1.67	0.75	0.65	1.46	1.24
Sortino ratio	2.48	2.69	2.83	1.93	0.98	0.81	1.72	1.57

Table 6 : Comparison of the portfolio performance between our Multi-agent PPO algorithm and other benchmark strategies for a portfolio consisting of 20 assets.

	MAPPO, 10 agents	MAPPO, 20 agents	MAPPO, 30 agents	EW	Aggressive Strategy	Moderate Strategy	S&P 500	WMAMR
Portfolio balance	37517	47702	53621	25541	22519	23548	25412	23952
Annual Profit	0.687	0.942	1.09	0.388	0.312	0.338	0.385	0.348
Annual volatility	0.253	0.249	0.248	0.348	0.361	0.368	0.332	0.347
Sharpe ratio	2.82	2.97	3.15	1.72	1.02	0.88	1.67	1.45
Sortino ratio	2.92	3.14	3.38	2.08	1.15	1.02	1.95	1.72

portfolio as a whole. It's important to note that this general trend may not hold true in all cases and for all types of algorithms. The optimal portfolio size may depend on various factors such as market conditions, trading costs, and the specific characteristics of the assets being traded. However, in the context of the experiments discussed here, it appears that increasing the number of assets in the portfolio leads to improved performance for the proposed algorithm. In general, our algorithm showed consistent outperformance across all three portfolios with different numbers of assets and agents.

6. CONCLUSIONS

In conclusion, our algorithm applied to portfolio optimization in the S&P 500 index has shown promising results. The experiment was conducted with different numbers of individual agents and assets in the portfolio. The results indicate that the algorithm performs better as the number of agents and assets increases. In general, the algorithm can be used to help investors optimize their portfolio in the S&P 500 allocation. The algorithm can also be extended to other financial markets and assets beyond the S&P 500 index. However, there are some limitations to the study that require further investigation in future research studies. These encompass the need for exploring alternative MARL algorithms; While our algorithm achieved good results, there are many other MARL algorithms that can be explored in the context of portfolio optimization. These include algorithms such as Q-learning, and Deep Deterministic Policy Gradient (DDPG). Additionally, sentiment analysis can be included in the algorithm to improve the performance of portfolio optimization. By incorporating sentiment analysis, the algorithm can use sentiment scores as an additional input feature to improve the accuracy of predicting the future price movements of assets. In conclusion, the algorithm shows great potential in revolutionizing the way investors allocate assets and manage risk in financial markets, and future research can further improve its performance and applicability.

REFERENCES:

[1] S. Park, H. Song, and S. Lee, ‘Linear programing models for portfolio optimization using a benchmark’, The European Journal of Finance, vol. 25, no. 5, pp. 435–457, Mar. 2019, doi: 10.1080/1351847X.2018.1536070.

[2] Z. Jiang, D. Xu, and J. Liang, ‘A Deep Reinforcement Learning Framework for the Financial Portfolio Management Problem’. arXiv, Jul. 16, 2017. [Online]. Available: http://arxiv.org/abs/1706.10059

[3] J. B. Heaton, N. G. Polson, and J. H. Witte, ‘Deep Learning in Finance’. arXiv, Jan. 14, 2018. [Online]. Available: http://arxiv.org/abs/1602.06561

[4] S. T. A. Niaki and S. Hoseinzade, ‘Forecasting S&P 500 index using artificial neural networks and design of experiments’, J Ind Eng Int, vol. 9, no. 1, p. 1, Feb. 2013, doi: 10.1186/2251-712X-9-1.

[5] CUMMING, James, ALRAJEH, Dr Dalal, et DICKENS, Luke. ‘An investigation into the use of reinforcement learning techniques within the algorithmic trading domain’. Imperial College London: London, UK, 2015, vol. 58.

[6] Y. Deng, F. Bao, Y. Kong, Z. Ren, and Q. Dai, ‘Deep Direct Reinforcement Learning for Financial Signal Representation and Trading’, IEEE Transactions on Neural Networks and Learning Systems, vol. 28, no. 3, pp. 653–664, Mar. 2017, doi: 10.1109/TNNLS.2016.2522401.

[7] ‘(PDF) Dynamic Programming and Optimal Control, Vol I’, dokumen.tips. https://dokumen.tips/documents/dynamic-programming-and-optimal-control-vol-i.html.

[8] R. S. Sutton and A. G. Barto, ‘Reinforcement Learning: An Introduction’.

[9] ‘Markowitz H M. Portfolio selection: efficient diversification of investments. New York: Wiley, 1959.’.

[10] R. A. Haugen, Modern investment theory, 2nd ed. Englewood Cliffs, N.J.: Prentice Hall, 1990. [Online]. Available: http://catalog.hathitrust.org/api/volumes/oclc/20265113.html

[11] X. Li, C. Cui, D. Cao, J. Du, and C. Zhang, ‘Hypergraph-Based Reinforcement Learning for Stock Portfolio Selection’, in ICASSP 2022 - 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), May 2022, pp. 4028–4032. doi: 10.1109/ICASSP43922.2022.9747138.

[12] H. Zhang, Z. Jiang, and J. Su, ‘A Deep Deterministic Policy Gradient-based Strategy for Stocks Portfolio Management’. arXiv, Mar. 21, 2021. [Online]. Available: http://arxiv.org/abs/2103.11455

[13] K. Zhang, Z. Yang, and T. Başar, ‘Multi-Agent Reinforcement Learning: A Selective

Overview of Theories and Algorithms', in Studies in Systems, Decision and Control, in Studies in Systems, Decision and Control. Springer, 2021, pp. 321–384. doi: 10.1007/978-3-030-60990-0_12.

[14] L. Busoniu, R. Babuska, and B. De Schutter, 'A Comprehensive Survey of Multiagent Reinforcement Learning', IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), vol. 38, no. 2, pp. 156–172, Mar. 2008, doi: 10.1109/TSMCC.2007.913919.

[15] V. Mnih et al., 'Human-level control through deep reinforcement learning', Nature, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[16] O. Vinyals et al., 'Grandmaster level in StarCraft II using multi-agent reinforcement learning', Nature, vol. 575, no. 7782, Art. no. 7782, Nov. 2019, doi: 10.1038/s41586-019-1724-z.

[17] J. Lee, R. Kim, S.-W. Yi, and J. Kang, 'MAPS: Multi-agent Reinforcement Learning-based Portfolio Management System', in Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, Jul. 2020, pp. 4520–4526. doi: 10.24963/ijcai.2020/623.

[18] J. Lussange, I. Lazarevich, S. Bourgeois-Gironde, S. Palminteri, and B. Gutkin, 'Modelling Stock Markets by Multi-agent Reinforcement Learning', Comput Econ, vol. 57, no. 1, pp. 113–147, Jan. 2021, doi: 10.1007/s10614-020-10038-w.

[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, 'Proximal Policy Optimization Algorithms'. arXiv, Aug. 28, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347

[20] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, 'Prioritized Experience Replay'. arXiv, Feb. 25, 2016. [Online]. Available: http://arxiv.org/abs/1511.05952

[21] C. Watkins, 'Learning From Delayed Rewards', Jan. 1989.

[22] V. Konda and J. Tsitsiklis, 'Actor-Critic Algorithms', in Advances in Neural Information Processing Systems, MIT Press, 1999. [Online]. Available: https://papers.nips.cc/paper/1999/hash/6449f44a102fde848669bdd9eb6b76fa-Abstract.html

[23] R. A. Howard, Dynamic programming and Markov processes. in Dynamic programming and Markov processes. Oxford, England: John Wiley, 1960, pp. viii, 136.