Journal of Theoretical and Applied Information Technology
15th July 2025. Vol.103. No.13

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

MULTI-AGENT PATH PLANNING BASED ON IMPROVED ASYNCHRONOUS ACTOR-CRITIC AGENT ALGORITHM.

TEH NORANIS BINTI MOHD ARIS[1], CHEN NINGNING[2], NORWATI MUSTAPHA[3], MASLINA ZOLKEPLI[4]

[1, 2, 3, 4] Department of Computer Science, Faculty of Computer Science and Information Technology,

Universiti Putra Malaysia, 43400 UPM, Serdang, Selangor, Malaysia

E-mail: [1]nuranis@upm.edu.my (corresponding author), [2]gs63185@student.upm.edu.my, [3]norwati@upm.edu.my, [4]masz@upm.edu.my

ABSTRACT

Existing approaches often rely on centralized or reactive planning methods, which can become inefficient or cause deadlocks in complex environments. There is a clear need for a process that handles these challenges while maintaining real-time performance and robustness. We propose a multi-agent path planning method based on the improved Asynchronous Actor-Critic Agent (A3C) algorithm. First, we enhance the Actor-Critic neural network within the A3C framework by integrating it with the VGG network to develop a fully decentralized strategy. Additionally, we improved the reinforcement learning rewards and penalties, allowing agents to perform real-time reactive path planning with partially visible information, exhibiting implicit coordination. Second, we use the sequence information input characteristics and long-term memory capabilities of Long Short Term Memory (LSTM) neural networks, allowing the neural network to gain "long-term memory" and enhance network learning speed. The LSTM neural network replaces the deep neural networks (DNN) in the original A3C network. We tested this approach in various environments with different sizes, agent quantities, and obstacle densities. Finally, we quantified the results based on average planning length, average planning time, and success rates over 100 tests. The experimental results show that the proposed method significantly improves the success rate and efficiency of multi-agent path planning in noisy and uncertain environments.

Keywords: *Multi-Agent Path planning, A3C, VGGnet, LSTM*

1. INTRODUCTION

With the rapid development of Deep Reinforcement Learning (DRL) in multi-agent systems, various practical application challenges and solutions in the direction of Multi-Agent Deep Reinforcement Learning (MADRL) have emerged. Path planning in a collision-free environment is crucial for many agents to complete tasks quickly and efficiently, and using deep reinforcement learning for multi-agent path planning is a new research field in artificial intelligence. Multi-agent path planning training methods can be divided into three categories:

(i) Centralized: The centralized training method assumes that the actions of all agents are determined by a central server that knows the intentions of all agents. The centralized method has three limitations. First, as the number of agents increases, the computational complexity of centralized control and scheduling becomes high. Second, due to communication issues, the centralized method cannot scale to large systems. Third, centralized systems are prone to failures in the central server, communication between agents, or the sensors on any single agent, for example, the optimal centralized planning algorithm CBS [1]. However, as the number of agents increases, the central processor consumes a large amount of resources, severely affecting the efficiency of MAPF.

(ii) Distributed: The distributed path planning algorithm adopts a decentralized approach, eliminating the need for a central processor, and making it more scalable and better suited for MAPF environments. For example, the classic distributed low-level obstacle avoidance algorithm ORCA [2]. Yang Hailan et al. [3] proposed a multi-agent path planning algorithm based on memristor reinforcement learning, which improves the probability transition function and pheromone of the ant colony algorithm, and suggests using

memristors as neural network synapses to enhance the DQN (deep Q-network) algorithm, effectively solving path planning issues in unknown environments. Chen et al. [4] proposed an improved deep deterministic policy gradient algorithm, which accelerates algorithm convergence by building a dual neural network and experience replay matrix, and uses batch priority sampling to train the neural network, addressing MAPF path planning in complex environments. Compared to centralized path planning algorithms, it is more adaptable to maps and better at handling dynamics. Meanwhile, distributed computing can address the dimensional explosion problem. However, conventional neural networks struggle with real-time path planning, often resulting in collisions, deadlocks, and congestion among agents. To address MAPF collisions, deadlocks, and congestion issues, Zhou et al. [5] proposed a path planning algorithm combining reservation tables and traffic rules, which prevents agent collisions by setting traffic rules and maintaining reservation tables storing all agents' positions at specific times. A dynamically weighted map is created using the reservation table, allowing agents to avoid congested areas based on map weights. Digani et al. [6] used a hierarchical map method, dividing the map into a topological structure layer and a grid map layer, and cutting the map into different regions. The segmented regions are treated as nodes in the topological structure, and the D* algorithm is used for path planning based on the edge weights of the topology. Lou et al. [7] designed a strategy to resolve multi-agent path conflicts and congestion by detecting whether overlapping nodes or overlapping time exist on the routes of agents. If conflicts are detected, agents are guided to replan their routes. Although the above algorithms can solve multi-agent collisions, deadlocks, and congestion problems, both weighted map creation and map segmentation come at the cost of multi-agent operational efficiency [8]. Furthermore, the hierarchical map method involves manual segmentation, making it less adaptable to reflect agent congestion issues.

(iii) Parameter sharing: Path planning is distributed, but the learning process is centralized. This method can leverage each agent's capabilities and compensate for the shortcomings of centralized path planning, adapting to location-based planning environments. It does not require knowledge of the dynamic model of the environment, has no communication requirements, and can be used in cooperative and competitive settings. However, it requires a central coordinator, which is not fully autonomous, and its scalability is poor. As a result, it does not support the training of a large number of agents, and the training time is relatively long. For example, the MADDPG algorithm [9], uses centralized training and distributed execution. Tao [10] proposed a decentralized partially observable multi-agent path planning method based on evolutionary reinforcement learning (MAPPER) to learn efficient local planning strategies in mixed dynamic environments. This method uses image-based representations to describe the behavior of dynamic obstacles and trains strategies in mixed dynamic environments without the homogeneity assumption. However, in three-dimensional environments, the training time and complexity significantly increase, and the method lacks generalization capability.

In summary, multi-agent path planning, as well as problems such as collisions, deadlocks, and congestion, have corresponding studies. However, the aforementioned multi-agent path planning algorithms often overlook these problems, while the path planning algorithms that aim to address these issues do so at the expense of agent operational efficiency. The lack of integrated research into both aspects prevents a fundamental improvement in MAPF efficiency.

Therefore, this paper designs a framework based on VGG16-LSTM, using an LSTM neural network to replace the original fully connected layers, enabling the network to have long-term memory capabilities. It proposes the A3C algorithm as the core algorithm of the framework, constructing different dynamic and static obstacle environments and designing corresponding action selection strategies and reward functions. This framework achieves multi-agent collaborative control, real-time path planning, and real-time obstacle avoidance through the A3C algorithm, thereby improving the operational efficiency of multi-agents.

2. THEORETICAL FOUNDATION

2.1 Reinforcement Learning (RL)

2.1.1 Introduction to Reinforcement Learning

Through interaction with the environment, an agent observes the potential impacts of its behavior and learns to adjust its actions for better responses to rewards. In reinforcement learning, machine learning algorithms control an autonomous agent that observes the environmental state and takes actions at each time step. After the agent takes action, the environment transitions to a new state based on the

current state and the selected action's result. The state is statistical information about the environment, containing all the information the agent needs (such as the positions of actuators and sensors), so the agent can choose the optimal action.

The sequence of actions taken by the agent is influenced by the rewards provided by the environment. A reward is a scalar that the environment provides to the agent when transitioning from one state to another. The agent's goal is to learn a policy π that maximizes the expected return. The policy returns actions based on the agent's current state. The interaction between the agent and the environment generates information. When the environment provides positive rewards, the agent is more likely to execute the current action the next time it encounters the same situation, whereas negative rewards will make the agent avoid executing the current action.

Formally, RL can be described as a Markov Decision Process (MDP). There are various methods to solve reinforcement learning problems, including value function-based methods, policy search methods, and a hybrid actor-critic method that uses both value function and policy search [11][12][13][14].

2.1.2 A3C Algorithm

Commonly used methods in reinforcement learning are usually Q-learning algorithms, and the concept of DRL emerged with the successful combination of deep learning and Q-learning, specifically with the advent of DQN[15][16][17]. However, DQN, a value function-based reinforcement learning method, is more suitable for solving discrete low-dimensional problems and cannot handle continuous high-dimensional problems. Q-learning is an off-policy method that cannot perform online updates. Since machine learning requires samples to be independently and identically distributed, DQN employs an experience replay mechanism, storing the agent's data in an experience replay unit, and the data can be batch processed or randomly sampled from different time steps to reduce non-stationarity and remove update correlations. However, this mechanism leads to higher memory consumption and increased computational load for each real interaction [18][19]. Due to these limitations, the Asynchronous Advantage Actor-Critic (A3C) algorithm emerged.

The A3C algorithm is applicable to both discrete and continuous high-dimensional problems. It uses parallel asynchronous execution of multiple agents instead of the experience replay mechanism in DQN to achieve decorrelation and improve algorithm stability. Simply put, the asynchronous concept is realized by using a multi-threaded parallel mechanism, where each sub-thread corresponds to an agent, explores the environment using different strategies in the same state, and obtains the corresponding parameter gradients. Once the required number of update steps is reached, the parameters are pushed to the main thread to update the current optimal parameters. A3C employs the Temporal Difference (TD) algorithm, which is a further improvement of the TD(λ) algorithm based on n-step TD learning and integrates weighting methods. N-step TD learning updates the current value function using n-step value function estimates, with the update formula shown as equation (1):

$$v(s_t) \leftarrow v(s_t) + \alpha\left(G_t^{(n)} - v(s_t)\right) \qquad (1)$$

Represents the error signal which, is updated in the direction of the error signal to correct errors. From the above equation, we can use the n-step value function to estimate the current value function. When solving problems, it is necessary to carefully consider which value estimation method to use to better approach the true value. To solve this problem, the n-step estimates are combined with a geometric weighting method to form the $TD(\lambda)$ algorithm, which involves multiplying $G_t^{(n)}$ by a weighting factor $(1-\lambda)\lambda^{n-1}$ to obtain G_t^{λ} as shown in equation (2), with the update formula shown in equation (3):

$$G_t^{\lambda} = (1-\lambda)G_t^{(1)} + (1-\lambda)\lambda G_t^{(2)} + \cdots + (1-\lambda)\lambda^{n-1}G_t^{(n)} \qquad (2)$$

$$v(s_t) \leftarrow v(s_t) + \alpha\left(G_t^{(\lambda)} - v(s_t)\right) \qquad (3)$$

λ is a constant between 0 and 1, with $1-\lambda$ acting as the weight. As time increases, $1-\lambda \leftarrow 0$, and the geometric weighting without memory results in low costs, making it possible to achieve $TD(\lambda)$ at the same cost as $TD(0)$. Specifically, when $\lambda = 0$, only the current state value is updated, making it equivalent to $TD(0)$. When $\lambda = 1$, the state value function update is equivalent to Monte Carlo (MC).
Another key aspect of A3C is its use of the Actor-Critic framework, which combines the advantages of Actor-only and Critic-only methods, achieving better convergence and single-step updates while applying them to continuous problems.
A3C approximates the Policy Gradient (PG) method within the Actor-Critic framework. The main idea is to combine the Policy Gradient (Actor)

and value function approximation (Critic) methods, abandoning the use of returns to evaluate the true value function and instead using the Critic algorithm, i.e., function approximators, to evaluate the value function, as shown in equation (4):

$$\nabla_\theta J(\theta) \approx \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s,a) Q_w(s,a)] \quad (4)$$

w is the parameter for updating the action value function in the Critic, and θ is the parameter for updating the policy in the Actor.

A3C (as shown in Figure 1) maintains a policy $\pi(a_t|s_t;\theta)$ and an estimate of a value function $V(s_t;\theta_v)$. Like the n-step Q-learning variant, the Actor-Critic variant also operates in the forward view and uses the same n-step return combination to update the policy and value function. The policy and value function is updated after each t_{max} operation or upon reaching a terminal state.

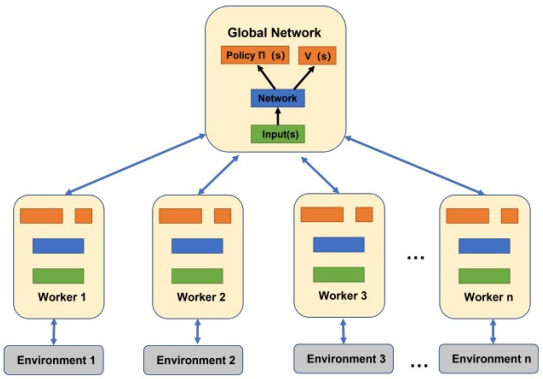

Figure 1: A3C algorithm framework diagram

The updates executed by the algorithm can be regarded as

$$\nabla_{\theta'} \log \pi(a_t|s_t;\theta') A(s_t,a_t;\theta,\theta_v) \quad (5)$$

$A(s_t,a_t;\theta,\theta_v)$ is an estimate of the advantage function. The advantage function is defined as:

$$\sum_{t=0}^{k-1} \gamma^i r_{t+1} + \gamma^k V(s_{t+k};\theta_v) - V(s_t;\theta_v) \quad (6)$$

k varies in different states, and its upper bound is t_{max}. Through parallel Actor and cumulative updates, the stability of training can be effectively improved, which is similar to value-based methods. Although the parameters θ for the policy and θ_v for the value function are usually separate, in practice, they often share some parameters, making the training more stable and reliable. Generally, a convolutional neural network has a softmax output for the policy $\pi(a_t|s_t;\theta)$ and a linear output for the value function $V(s_t;\theta_v)$, with all non-output layers shared. Adding the entropy of the policy π to the objective function can effectively suppress linear outputs, thus preventing premature convergence to suboptimal deterministic policies. This technique was originally proposed in the literature [36] and found particularly useful in tasks requiring hierarchical behaviors. The gradient of the complete objective function, including the entropy regularization term, concerning the policy parameters is:

$$\nabla_{\theta'} \log \pi(a_t|s_t;\theta')(R_t - V(s_t;\theta_v)) + \beta \nabla_{\theta'} H(\pi(s_t;\theta')) \quad (7)$$

H is the entropy. The hyperparameter β controls the strength of the entropy regularization term.

2.2 LSTM Neural Network

When faced with tasks like predicting the next scene in a video, speech recognition, and language translation, Deep Neural Networks (DNNs) and Convolutional Neural Networks (CNNs) often underperform. This is because the training samples for these tasks are sequences of varying lengths and continuous information, which are difficult to split into isolated words or single video frames, whereas DNNs and CNNs are more suitable for training on fixed-size samples.

Recurrent Neural Networks (RNNs) excel in training and predicting continuous sequence inputs due to their looping mechanism, which connects neurons of consecutive time sequences using a weight matrix, strengthening unit connections. Thus, the output is related not only to the current input but also to the previous output, providing the neural network with memory capabilities.

Although RNNs handle time series problems effectively, they face issues like the gradient vanishing problem, where significant short-term input changes can greatly affect their state. The gradient vanishing in RNNs occurs because the derivative of the activation function tanh is between 0 and 1. During backpropagation, if the parameter W is initialized to a value less than 1, the product of multiple (tanh function derivatives $* W$) results in a very small partial derivative, leading to gradient vanishing, causing slow parameter updates, or even stopping updates altogether. This indicates that RNNs have long-term dependency effects, meaning a unit in the RNN is mainly influenced by its neighboring units, and over prolonged periods, results in smaller memory values.

Journal of Theoretical and Applied Information Technology
15th July 2025. Vol.103. No.13

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

The Long Short-Term Memory (LSTM) variant effectively addresses the long-distance dependency problem in RNNs by adding a cell state C_t to store long-term states An LSTM memory cell is shown in Figure 2. The key mechanisms for LSTM's long-term memory function are the forget gate, input gate, and output gate. The forget gate, shown in the blue dashed box in Figure 2, determines how much "memory" from the previous node's cell state can be retained. The input gate, composed of the green and orange dashed boxes, includes the candidate gate in the orange dashed box. It determines how much of the current input can be remembered. The output gate, in the yellow dashed box in Figure 2, controls the proportion of "historical" information to combine with the current input. The gates are single-layer fully connected networks that control outputs by manipulating activation values and the products of the vectors they wish to control.

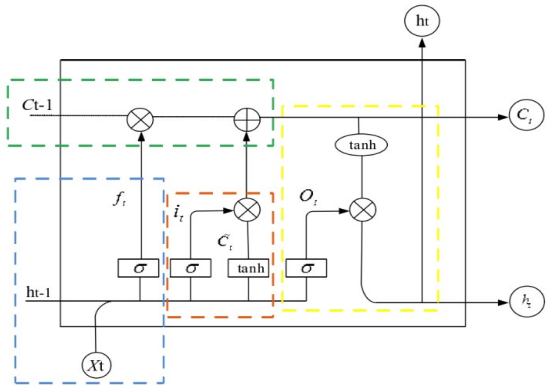

Figure 2: LSTM Memory Module

3. MULTI-AGENT PATH PLANNING BASED ON IMPROVED A3C ALGORITHM

3.1 Policy Representation based on improved A3C Algorithm

The purpose of this section is to transform the Multi-Agent Path Finding (MAPF) problem into the Reinforcement Learning (RL) framework.

3.1.1 Agent State Space

In a partially observable discrete grid environment, the agent can only observe the state of the environment within a limited field of view (FOV) centered on itself. Within the limited FOV, the available information is divided into different channels to simplify the agent's learning task. Specifically, each observation consists of two-dimensional matrices representing obstacles, other agents' positions, the agent's target position (if within the FOV), and the target positions of other observable agents (as shown in Figure 3). When the agent approaches the edge of the environment, obstacles are added to all positions outside the boundary of the environment.

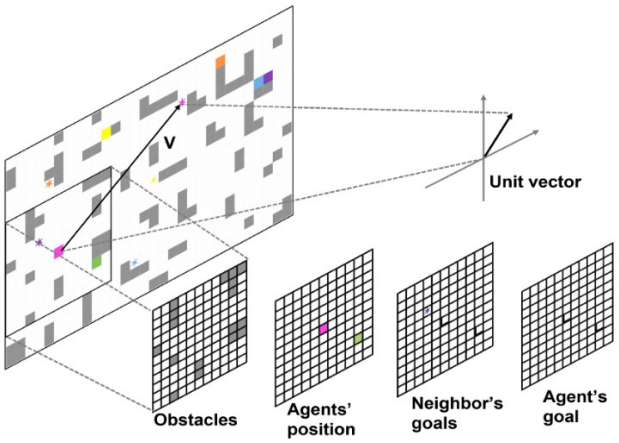

Figure 3: Observation Space for Agents

3.1.2 Agent Action Space

In the grid environment, the agent takes discrete actions: moving one cell in one of the four basic directions or remaining stationary. At each time step, some actions may be invalid, such as moving into obstacles or other agents. During training, actions are sampled only from valid actions, and an additional loss function helps learn this information. Compared to giving negative rewards to agents who choose invalid actions, this method can achieve more stable training. Additionally, to prevent convergence to oscillating policies, agents are prohibited from returning to the position they occupied in the previous time step during training (agents can remain stationary for multiple consecutive time steps). This is necessary to encourage exploration and the learning of effective strategies. If the agent chooses an invalid move during training, it will remain stationary at that time step.

3.1.3 Reward Function Design

The reward function (as shown in Table 1) follows the same intuition used by most reward functions in grid worlds: the agent is penalized for each time step it does not stay on the target, leading to a strategy to reach the target as quickly as possible.

Table 1: reward setting grid environment

Movements	Reward
Movement (up/down/left/right)	-0.3
collision	-2
No movement (in/out of target position)	0.0/-0.5
Finish a round	20

Journal of Theoretical and Applied Information Technology
15th July 2025. Vol.103. No.13

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

3.2 Improvement to the Actor-Critic Network and Introducing LSTM in A3C

In the A3C algorithm, a deep neural network approximates the agent's policy, mapping current observations of the environment to the next action. This network has multiple outputs: one for the actual policy and the other for training. The advantage of this algorithm is that it does not require explicit agent communication, allowing multiple agents to learn a common strategy and achieve better training results in a shared environment.

The 7-layer convolutional network shown in Figure 5 is an improvement on the Actor-Critic network, inspired by VGGnet [20][21][22]. In VGG, three 3×3 convolutional kernels can replace a 7×7 convolutional kernel, and two 3×3 convolutional kernels can replace a 5×5 convolutional kernel, effectively increasing the depth and efficiency of the neural network while maintaining the same receptive field, thereby better meeting the complexity of modeling. This is because a 5×5 convolutional kernel can be considered as performing a fully connected operation over a 5×5 area, while using two 3×3 convolutional kernels can convolve different parts of this area separately and then concatenate them, effectively realizing a fully connected operation over a 5×5 area. This method reduces the number of parameters that need to be learned, improving the model's efficiency and generalization ability, as shown in Figure 4.

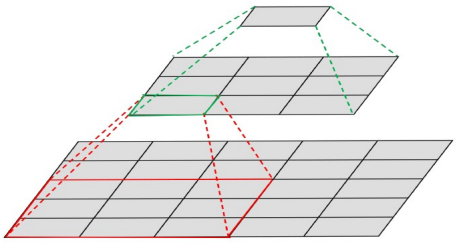

Figure 4: Schematic of two 3×3 convolution kernels replacing a 5×5 convolution

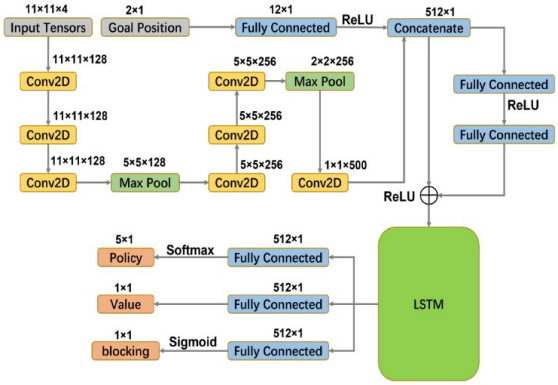

Figure 5: Structure of a neural network consisting of 7 convolutional layers crossed with a maximum pooling layer

In Figure 5, several small 3×3 convolutional kernels are used between each max-pooling layer. Specifically, the two inputs of the neural network (local observations and target direction/distance) are preprocessed independently and then concatenated. The four-channel matrix representing local observations (11×11×4 tensors) passes through three convolutions and two stages of max-pooling, followed by a final convolutional layer. Meanwhile, the target unit vector and magnitude pass through a fully connected layer. Then, the concatenation of these two preprocessed inputs passes through two fully connected layers and is fed into an LSTM network with an output size of 512. The output layer consists of policy neurons with softmax activation, value output, and feature layers, used to train each agent to understand whether it hinders other agents from achieving their goals. To achieve cooperative behavior among agents, we apply penalties to encourage the movement of other agents (referred to as "blocking penalties"). If an agent stays at its target position while blocking another agent from reaching its goal, it receives a severe penalty (a penalty of -2 in practice, as shown in Table 1). The "blocking" output in the network is trained to predict when an agent is blocking other agents, implicitly explaining to the agent for incurring additional penalties in such situations. During training, policy, value, and blocking outputs are batch updated every step or at the end of an episode. Generally, the value is updated to match the total discounted return ($R_t = \sum_{i=0}^{k} \gamma^i r_{t+i}$)by minimizing:

$$L_V = \sum_{t=0}^{T} (V(o_t;\theta) - R_t)^2 \quad (8)$$

Journal of Theoretical and Applied Information Technology
15th July 2025. Vol.103. No.13

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

When updating the policy, the advantage function is approximated using bootstrapping from the value function:

$$A(o_t, a_t; \theta) = \sum_{i=0}^{k-1} \gamma^i r_{t+i} + \gamma^k V(o_{k+t}; \theta) - V(o_t; \theta) \quad (9)$$

Where is constrained by the batch size T. Additionally, the entropy term $H(\pi(o))$ is added to the policy loss, which has been shown to encourage exploration and prevent premature convergence by penalizing policies that always choose the same action. The policy loss is:

$$L_\pi = \sigma_H \cdot H(\pi(o)) - \sum_{t=0}^{T} \log(P(a_t|\pi, o; \theta) A(o_t, a_t; \theta)) \quad (10)$$

With a small entropy weight σ_H (actually $\sigma_H = 0.01$). Two additional loss functions help guide and stabilize training. First, update the blocking prediction output by minimizing $L_{blocking}$ (the log-likelihood of prediction errors). Second, define the loss function L_{valid} to minimize the log-likelihood of choosing invalid moves.

Since the input handled by reinforcement learning is essentially sequence information, the LSTM network with long-term memory capabilities is chosen to replace the fully connected layers in the original A3C network model, giving the entire network model better long-term memory. The overall model structure of the A3C network combined with LSTM for agent path planning is illustrated.

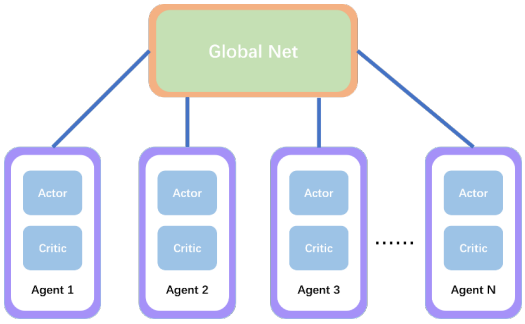

Figure 6: Improved A3C Network Model Structure

As shown in Figure 6, each thread has Actor and Critic networks with the same structure. The global network also has Actor and Critic networks but only serves to store the corresponding parameters. During training, the global network saves the parameters of the best-performing network in the sub-threads and then pushes these parameters to each local network via a synchronization mechanism. Sub-threads compute gradients for the corresponding Actor and Critic networks during training and perform gradient descent in the global network, saving the corresponding network parameters. Each time the sub-threads complete an action, the push and pull operations of network parameters are repeated to ensure consistency and optimal performance.

4. SIMULATION EXPERIMENT AND AN ANALYSIS

The experimental environment in this paper is an improved grid world based on the Asprilo benchmark [23], typically used by classical or learning-based MAPF solvers. Asprilo is a benchmark framework for multi-agent path planning, focusing on typical scenarios in internal logistics and warehouse automation with multiple mobile agents. It provides concise specifications for these problems, along with tools to generate benchmark instances, verify path planning, and visualize instances (as shown in Figure 7) and planning results.

Figure 7: Schematic diagram of grid environment

4.1 Introduction to Baseline Methods

The baseline methods in this experiment include the classic distributed low-level obstacle avoidance ORCA [24][25][26] (Optimal Reciprocal Collision Avoidance) algorithm, the optimal centralized planning CBS [27][28][29] (Conflict-Based Search) algorithm, and a coupled planning algorithm ODrM* [30][31][32] (with expansion factor $\epsilon = 2.0$). The dynamic obstacle avoidance in the ORCA algorithm relies on the analysis of the velocities between the agent, other agents, and obstacles. CBS is a two-layer algorithm, while the ODrM* algorithm performs coupled replanning for all agents in the environment.

4.2 Experimental Design and Result Analysis

In the experiment, at the beginning of each round, the grid environment size is randomly chosen to be 10, 40, or 70, with a doubled probability for size 10. The obstacle density is selected from a triangular distribution between 0% and 50%. Each grid environment ensures at least one path for the agent to reach the target point, with agents, targets, and obstacles placed randomly and uniformly. At each time step, agents' actions are executed in random order, ensuring equal execution priority. Eight agents are placed in each environment during training. At time step 0, all agents in the same environment are synchronized to act in parallel. During training, our method and baseline methods are executed with the same parameter settings. See Table 2 for details.

Table 2: Parameter setting in the experiment.

Parameter	Setting
Max_Episode_Length	256
γ	0.95
EXPERIENCE_BUFFER_SIZE	128
GRID_SIZE	11
ENVIRONMENT_SIZE	(10,70)
OBSTACLE_DENSITY	(0,5)
NUM_META_AGENTS	3
LR_Q	2e-5
DEMONSTRATION_PROB	0.5
optimizer	Nadam
value loss	MSE
imitation loss	Cross Entropy

After training, testing is conducted under various grid environment sizes, obstacle densities, and agent scales. Grid environment sizes are set to 10 and 20, obstacle densities to 0, 0.2, 0.3, and 0.4, and agent scales to 4, 8, 16, 32, and 64. The experiment compares the success rates of different planners, i.e., whether they complete tasks within the specified time steps. Agents plan their paths in grid environments of size 10 and 20, within a maximum of 256 time steps. One hundred tests are conducted in environments with fixed grid sizes and obstacle densities. Performance comparisons with baseline algorithms are evaluated by calculating the average plan lengths, success rates, and average plan times over these 100 tests. The experimental results are shown in Figures 8 to 13.

Figures 8 and 9 show that with fewer agents, the average planning length of the ORCA method is significantly longer than the other three methods. This is understandable because, in the multi-agent dynamic environment obstacle avoidance problem, multiple ORCA methods are used to find a common solution. This involves obtaining a set of velocities that can avoid collisions with multiple agents and obstacles and then selecting a velocity from this set. As the number of agents increases, our method shows a significant increase in average planning length compared to other methods. More agents lead to more frequent blocking situations, significantly increasing the average planning length.

Figures 10 and 11 show that our method performs very well with low obstacle density. With low obstacle density, agents can easily bypass each other. In dense environments, joint actions are necessary for agents to reach their goals (sometimes requiring drastic path changes), where our method can be easily outperformed. Similarly, ORCA's performance is much worse; it cannot prevent deadlocks and performs very poorly in most scenarios involving more than 16 agents and any obstacles due to its completely decoupled reactivity. In a grid environment of size 10, with different obstacle densities, our method almost always shows better success rates than traditional multi-agent path planners. Traditional multi-agent path planners require global information about the environment. In a grid environment of size 10, our FOV is 11, meaning our method also knows the global information in this environment. Finally, since training involves grid environments of different sizes but the agent scale remains unchanged, it is observed that as the number of nearby agents within the FOV increases, the agents' performance worsens. Overall, our method adapts to different team sizes, environment sizes, and obstacle densities, despite allowing agents to access only local information about the environment. In low obstacle density environments, it exhibits the same performance, and in some cases, even outperforms state-of-the-art multi-agent path planners, despite these planners accessing global information of the system.

Multi-agent path planning involves simultaneously planning paths for multiple agents in a given space to ensure they do not collide while minimizing total runtime. The average planning time in the experiment is shown in Figures 12 and 13. Our method performs well in terms of average planning length and success rate across different grid sizes and obstacle densities. However, our method takes longer average planning times compared to the CBS method, the ODrM* method, and the ORCA method.

Journal of Theoretical and Applied Information Technology
15th July 2025. Vol.103. No.13

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

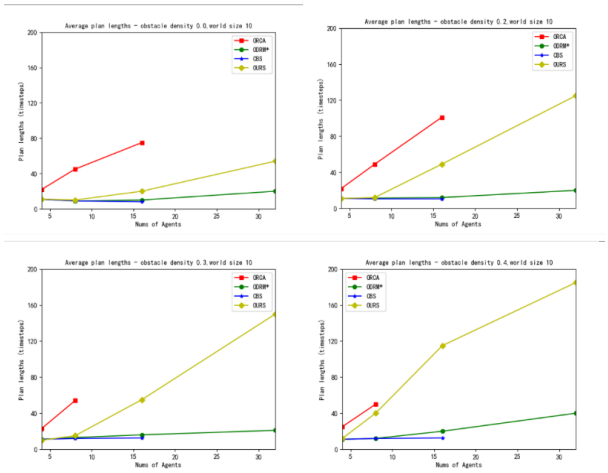

Figure 8: Average planning length in a grid of size 10 with different obstacle densities

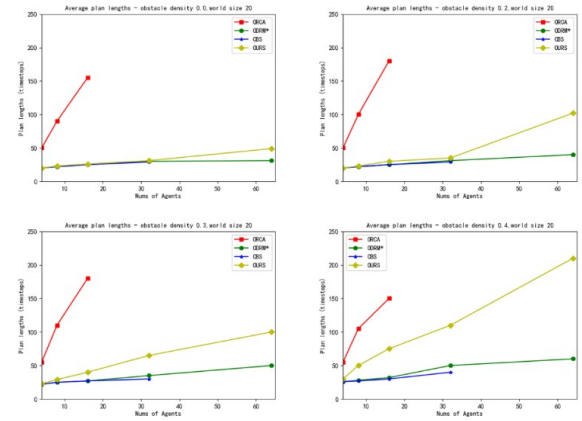

Figure 9: Average planning length in a grid of size 20 with different obstacle densities

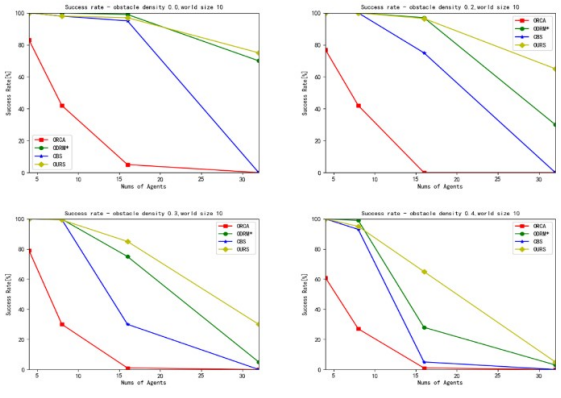

Figure 10: Success rates in a grid of size 10 with different obstacle densities

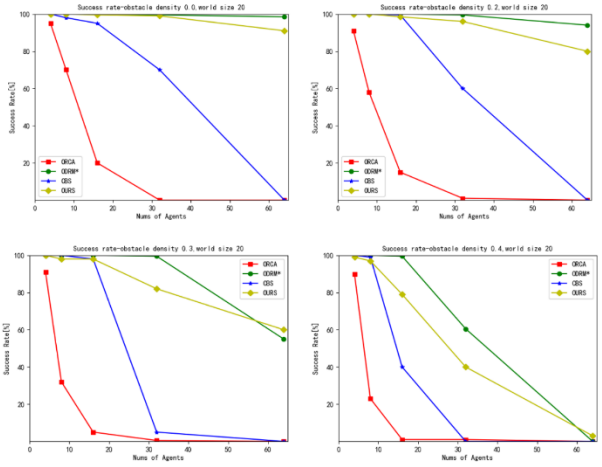

Figure 11: Success rates in a grid of size 20 with different obstacle densities

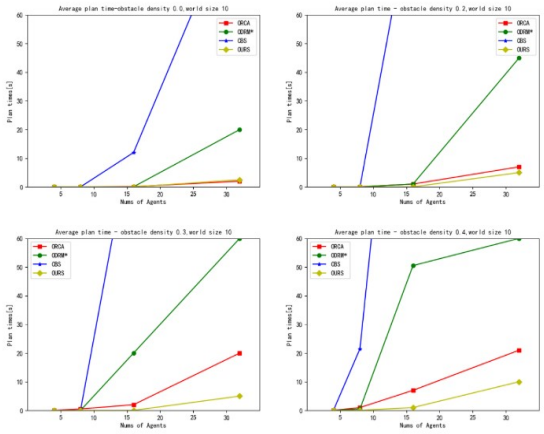

Figure 12: Average planning time in a grid of size 10 with different obstacle densities

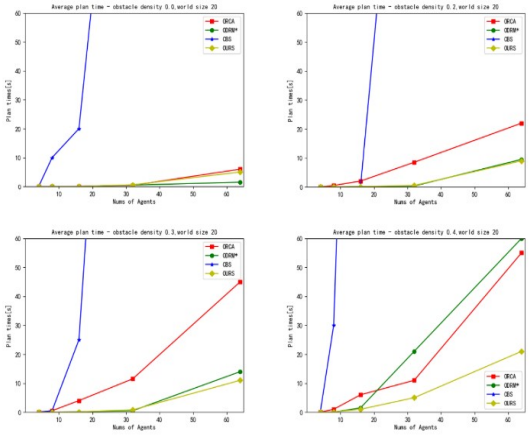

Figure 13: Average planning time in a grid of size 20 with different obstacle densities

5. CONCLUSION

This paper addresses issues in MAPF path planning such as poor real-time performance, weak obstacle recognition, multi-agent collisions, deadlocks, and congestion by designing a VGG16-LSTM-based framework. The A3C algorithm is proposed as the core algorithm of the framework, with different dynamic and static obstacle environments constructed, along with corresponding action selection strategies and reward functions. This framework utilizes the A3C algorithm to achieve multi-agent cooperative control, real-time path planning, and obstacle avoidance. Meanwhile, experiments and analyses are conducted on the proposed algorithm across various scenarios, dimensions, and evaluation metrics, demonstrating that the VGG16-LSTM-A3C-based multi-agent path planning method improves MAPF's real-time performance, enhances agents' obstacle recognition capabilities, and resolves collision, deadlock, and congestion issues encountered during operation, thereby achieving the goal of improving MAPF efficiency.

Based on this research, there are still some shortcomings, and our future research directions will address these shortcomings:

1. Experimental results show a gap in minimizing total runtime compared to other multi-agent path planning methods. Therefore, a new method to reduce the total runtime is necessary.
2. In dense and highly structured environments, effectively solving lifelong multi-agent path planning requires costly coordination among agents and frequent replanning capabilities, posing a significant challenge for existing coupled and decoupled methods.

ACKNOWLEDGMENT

Thank you to the Faculty of Computer Science and Information Technology, Universiti Putra Malaysia for providing financial support under allocation vote 6236500.

REFERENCES:

[1]Yifan Bai, Shruti Kotpalliwar, Christoforos Kanellakis, George Nikolakopoulos, Multi-agent Path Planning Based on Conflict-Based Search (CBS) Variations for Heterogeneous Robots. published *13 February 2025 in Journal of Intelligent & Robotic Systems*, doi.org/10.1007/s10846-025-02229-0.

[2]Matthew D. Houghton, Michael J. Acheson, Andrew P. Patterson, Alex Oshin, Irene M. Gregory, Combined Bernstein Polynomial Optimal Reciprocal Collision Avoidance Differential Dynamic Programming for Trajectory Replanning and Collision Avoidance for UAM Vehicles, published *23 January 2023 in AIAA SCITECH 2023 Forum*, doi.org/10.2514/6.2023-2544.

[3]YANG H L,QI Y Q,RONG D.AGV path planning based on memristor reinforcement learning in warehouse environment. *Computer Engineering and Applications,* 2024,59(17):318-327. doi.org/10.1109/cac51589.2020.9326742.

[4]Yinliang Chen, Liang Liang. SLP-Improved DDPG Path-Planning Algorithm for Mobile Robot in Large-Scale Dynamic Environment. published *28 March 2023 in Sensors*, doi.org/10.3390/s23073521.

[5]Mingjun Zhou. Analysis Of Current Development Status of Global Path Planning for Intelligent Carrier Robots in Warehouse Logistics Scenarios . published 26 June 2024 in *Highlights in Science, Engineering and Technology*. doi.org/10.54097/3zcsde07.

[6]DIGANI V,SABATTINI L,SECCHI C.A probabilistic eulerian traffic model for the coordination of multiple AGVs in auto-matic warehouse. *IEEE Robotics and Automatic Letters*,2024,1. doi.org/10.1049/icp.2024.3708.

[7]Ping Lou, Yutong Zhong, Jiwei Hu, Chuannian Fan, Xiao Chen. Digital-Twin-Driven AGV Scheduling and Routing in Automated Container Terminals. Published 13 June 2023, *Mathematics*. doi.org/10.3390/math11122678.

[8]Changxi Zhu, Mehdi Dastani, Shihan Wang. A survey of multi-agent deep reinforcement learning with communication. Published June 2024 in *Autonomous Agents and Multi-Agent Systems*, doi.org/10.1007/s10458-023-09633-6.

[9]Ziyang Jin, Yijun Wang, Jingying Lv. Edge Computing Task Offloading of Internet of Vehicles Based on Improved MADDPG Algorithm, Published 29 February 2024 in *KSII Transactions on Internet and Information Systems*, doi.org/10.3837/tiis.2024.02.004.

[10]Mengxue Tao, Qiang Li, Junxi Yu. Multi-Objective Dynamic Path Planning with Multi-Agent Deep Reinforcement Learning, Published 27 December 2024 in *Journal of Marine Science and Engineering*, doi.org/10.3390/jmse13010020.

[11]Y. Wang, B. Xiang, S. Huang, and G. Sartoretti, "SCRIMP: Scalable Communication for Reinforcement- and Imitation-Learning-Based Multi-Agent Pathfinding," in *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 1 Oct. 2023, doi: 10.1109/iros55552.2023.10342305.

[12]M. Gergely, "Multi-Agent Deep Reinforcement Learning for Collaborative Task Scheduling," in *Proceedings of the 16th International Conference on Agents and Artificial Intelligence*, 2024, doi: 10.5220/0012434700003636.

[13]S. Wiggins, Y. Meng, R. Kannan, and V. Prasanna, "Characterizing Speed Performance of Multi-Agent Reinforcement Learning," in *Proceedings of the 12th International Conference on Data Science, Technology and Applications*, 2023, doi: 10.5220/0012082200003541.

[14]C. Zhu, M. Dastani, and S. Wang, "A survey of multi-agent deep reinforcement learning with communication," in *Autonomous Agents and Multi-Agent Systems*, Jun. 2024, doi: 10.1007/s10458-023-09633-6.

[15]Z. Chen, "DQN–MADDPG Coordinating the Multi-agent Cooperation," in *Highlights, Science, Engineering and Technology*, 1 Apr. 2023, doi: 10.54097/hset.v39i.6720.

[16]J. A. J. Sujana and L. Namasivaayam, "Pressure, Queue, and Average Speed - Based Multi-Agent DQN for Optimizing Traffic Signal Control," in *2024 International Conference on Advances in Data Engineering and Intelligent Computing Systems (ADICS)*, 18 Apr. 2024, doi: 10.1109/ADICS58448.2024.10533624.

[17]Z. Lu, "Improving the Performance of Deep Q Network in Decision Making Environment: Applying Multi-Head Attention into DQN," in *Proceedings of the 1st International Conference on Engineering Management, Information Technology and Intelligence*, 2024, doi: 10.5220/0012958200004508.

[18]M. S. R. Krishna and S. S. Mangalampalli, "PWSA3C: Prioritized Workflow Scheduler in Cloud Computing Using Asynchronous Advantage Actor Critic (A3C) Algorithm," in *IEEE Access*, 2024, doi: 10.1109/access.2024.3457518.

[19]C. Sun, T. Li, M. Wu, and S. Chu, "Research on adaptive circuit structure optimization in electronic design based on Asynchronous Advantage Actor Critic (A3C) algorithm," in *Journal of Intelligent & Fuzzy Systems*, 6 Nov. 2024, doi: 10.3233/jifs-241935.

[20]S. Prasad and R. Nayaka, "Deep learning based image classification using small VGGNet architecture," in *AIP Conference Proceedings*, 2024, doi: 10.1063/5.0194640.

[21]S. K. Arthi R, "Deep Learning based Brain Stroke Detection using Improved VGGNet," in *WSEAS Transactions on Biology and Biomedicine*, 12 Oct. 2023, doi: 10.37394/23208.2023.20.21.

[22]D. Upadhyay, M. Manwal, V. Kukreja, and R. Sharma, "Deep Learning-based VGGNet, GoogleNet, and DenseNet121 Models for Cervical Cancer Prediction," in *2024 5th International Conference for Emerging Technology (INCET)*, 24 May 2024, doi: 10.1109/incet61516.2024.10593267.

[23]R. Stern, N. Sturtevant, A. Felner, S. Koenig, H. Ma, T. Walker, J. Li, D. Atzmon, L. Cohen, T. K. Kumar, R. Barták, and E. Boyarski, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *Proceedings of the International Symposium on Combinatorial Search*, 1 Sep. 2021, doi: 10.1609/socs.v10i1.18510.

[24]M. D. Houghton, M. J. Acheson, A. P. Patterson, A. Oshin, and I. M. Gregory, "Combined Bernstein Polynomial Optimal Reciprocal Collision Avoidance Differential Dynamic Programming for Trajectory Replanning and Collision Avoidance for UAM Vehicles," in *AIAA SCITECH 2023 Forum*, 23 Jan. 2023, doi: 10.2514/6.2023-2544.

[25]T. Itzhaki and V. Shaferman, "Bounded and Linear Quadratic Optimal Low-Thrust Collision Avoidance in Circular Orbits," in *AIAA SCITECH 2024 Forum*, 8 Jan. 2024, doi: 10.2514/6.2024-0094.

[26]Z. Liu, C. Yao, W. Na, C. Liu, and Q. Chen, "MAPPO-Based Optimal Reciprocal Collision Avoidance for Autonomous Mobile Robots in Crowds," in *2023 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 1 Oct. 2023, doi: 10.1109/smc53992.2023.10394332.

[27]G. Pooja and S. Bhosale, "Conflict-Based Search for Optimal Multi-Agent Pathfinding," in *International Journal of Advanced Research in Science, Communication and Technology*, 30 Jan. 2023, doi: 10.48175/ijarsct-8161.

[28]J. Ryu, Y. Kwon, S. Yoon, and K. Lee, "Conflict Area Prediction for Boosting Search-Based Multi-Agent Pathfinding Algorithms," in *2024 IEEE International Conference on*

Robotics and Automation (ICRA), 13 May 2024, doi: 10.1109/icra57147.2024.10610843.

[29]K. Okumura, "LaCAM: Search-Based Algorithm for Quick Multi-Agent Pathfinding," in *Proceedings of the AAAI Conference on Artificial Intelligence*, 26 Jun. 2023, doi: 10.1609/aaai.v37i10.26377.

[30]C. Henkel and M. Toussaint, "Optimized directed roadmap graph for multi-agent path finding using stochastic gradient descent," in *Proceedings of the 35th Annual ACM Symposium on Applied Computing*, 30 Mar. 2020, doi: 10.1145/3341105.3373916.

[31]C. Ferner, G. Wagner, and H. Choset, "ODrM* optimal multirobot path planning in low dimensional search spaces," in *2013 IEEE International Conference on Robotics and Automation*, May 2013, doi: 10.1109/icra.2013.6631119.

[32]S. Ardizzoni, L. Consolini, M. Locatelli, and I. Saccani, "Constrained Motion Planning and Multi-Agent Path Finding on directed graphs," in *Automatica*, Jul. 2024, doi: 10.1016/j.automatica.2024.111593.