Implementation of PPO for Multi-Agent Path Finding with Dynamic Obstacles

Jinwoo Park, Jeonghwan Kim, Tae Kyung Park

***Abstract*—Multi-agent path finding (MAPF) is a fundamental problem in the field of robotics where multiple agents have to navigate to their own goals without collisions in a confined space. This problem is apparent in many real world applications such as automated warehouses. We propose a reinforcement learning based method for learning decentralized policy for MAPF. Specifically, we propose a synchronized proximal policy optimization (PPO) algorithm to obtain a stable yet efficient training. Synchronization of weight of agents in the same environment along with PPO loss helped mitigating data inefficiency issue exhibited in our baseline MAPPER [1] and further stabilized the training. The effectiveness of our algorithm was verified against the baseline using average rewards, collision and success rates.**

***Index Terms*—multi-agent path finding, reinforcement learning, proximal policy optimization, decentralized policy**

I. Introduction

Multi-agent path finding (MAPF) aims to generate collision-free paths for multiple agents under constraints such as partial observation, dense population, highly structured and dynamic environment. MAPF can be classified into two categories: centralized and decentralized. Centralized method runs optimization using full information about the global environment. Such method requires high network connection with immense computing power because it has to process information coming from multiple working agents in densely populated environments. In order to mitigate these issues, decentralized methods have gained popularity where each agent makes its own decision based on its local observation. This project focuses on the decentralized MAPF where agents move toward their goals without full observation of the environment with dynamic obstacles.

Prior to reinforcement learning (RL), conflict-based search (CBS) [2] methods were dominant in the MAPF literature. CBS has been addressed for its disadvantage regarding the exponential runtime to the number of conflicts. We therefore propose a reinforcement learning based method for solving the decentralized MAPF, namely Multi-agent path planning with proximal policy optimization (MAPFO). Our work is a direct succession of the A2C based agent, MAPPER [1] which we chose as our baseline. We stabilize the training with synchronized agents and clipped surrogate loss [3] to achieve performance gain.

In this project, we verify that our method of synchronized PPO shows more stable learning and gains higher rewards and success rates compared to the baseline MAPPER.

J. Park is with the Institute for Robotics and Intelligent Machines, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: jinwoop@gatech.edu).

J. Kim is with the School of Electrical and Computer Engineering, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: jkim3662@gatech.edu).

T. Park is with the School of Computer Science, Georgia Institute of Technology, Atlanta, GA 30332 USA (e-mail: tpark97@gatech.edu).

II. Related Work

A. MAPF with Reinforcement Learning (RL)

Many research efforts have been devoted to solve MAPF with RL. PRIMAL [4] and its successor PRIMAL2 [5] combine RL with imitation learning to solve this problem. Our baseline MAPPER [1] implements A2C with evolutionary update to learn a policy that is robust to environments with dynamic obstacles. It is still challenging to use RL for MAPF due to its nature of long-horizon episode with sparse rewards. We follow a method proposed by MAPPER [1] to overcome this issue by filling the gaps for sparse rewards with sub-tasks and trying to follow the path generated by A*.

B. Proximal Policy Optimization (PPO)

It is shown in MAPPO [6] that on-policy reinforcement learning in multi-agent settings can achieve sample efficiency compared to that of the off-policy methods, and require minimal hyperparameter tuning without any domain-specific algorithmic modifications or architectures. They corroborated that PPO-based methods indeed have outstanding performance on various cooperative Multi-agent reinforcement learning (MARL) environments. From the baseline MAPPER, we modified vanilla A2C loss to PPO loss and changed separated policy to synchronized policy to mitigate issues caused by abrupt evolutionary update.

III. Method

Our environment is derived from the baseline MAPPER [1]. The environment (dimensions of 20x24, 32x32, and 73x51) contain agents, dynamic and static obstacles, and goal positions. An example of training environment of this paper is depicted in Figure 1.

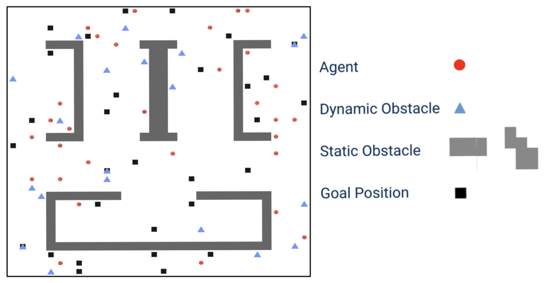


Fig. 1: Example of an environment.

A. Training environment

Each agent has a field of view (FOV) of 15x15 grid with it being in the center. An agent receives receives observations: 1) its previous positions, 2) position of other agents and dynamic obstacles, 3) the weighted observed trajectories of other agents within its FOV. In addition, each agent computes a waypoint vector towards its goal.

The agent chooses an action from 9 discrete actions (i.e., N, NE, E, SE, S, SW, W, NW, or stay) where it can move to the neighboring grid or stay in the current grid.

We calculate the reward (derived from MAPPER [1]) for each step as

$$R = r_s + r_c + r_o + \lambda\, r_f + r_g, \tag{1}$$

where the value of each term is shown in Table I, $\mathbf{p}_a \in \mathbb{R}^2$, and $\mathbb{S} = \{\mathbf{p}_{start}, \cdots, \mathbf{p}_{goal}\}$.

Rewards	Value
step penalty r_s	-0.1 (move) or -0.5 (wait)
collision penalty r_c	-5
oscillation penalty r_o	-0.3
off-route penalty weight λ	0.3
off-route penalty r_f	$-min_{\mathbf{p}\in S}\lvert\lvert\mathbf{p}_a - \mathbf{p}\rvert\rvert_2$
goal-reaching reward r_g	30

TABLE I: Reward design

The maximum number of steps (*max_step*) that an agent can take in an episode is different for each agent. The *max_step* of an agent used in the training is the four times the length of trajectory computed by the A*. This is to give enough steps for the agent to explore and learn new policies while precluding the exploitation of bad policies. An episode terminates when all agents reach their goals or time step reaches the largest *max_step* of all agents. Each agent stops rollout after reaching its goal.

B. Algorithm

The network architecture of our actor-critic agent follows the baseline MAPPER where the actor and critic network shares the observation encoding part of the network. We use CNN layers and MLP layers to encode local 2D observation and waypoint vector respectively. The outputs of the CNN and MLP layers are concatenated before fed into additional MLP layers to create a single observation encoding. The observation encoding is fed into an actor network and a critic network to output action probability and state-value.

We synchronize the policy for all agents in the environment. Each agent performs rollout based on single synchronized global policy so that data collected from all agents count towards the synchronized policy. We use clipped surrogate loss [3] to stabilize the training by following suggestions of MAPPO [6]. Our method distinguishes from vanilla A2C and PPO in that our method synchronizes the weights of multiple agents acting in the same environment while the vanilla A2C and PPO methods synchronize the weights of multiple workers performing rollouts in separate environments. Our algorithm MAPFO is depicted in Algorithm 1.

Algorithm 1 MAPFO

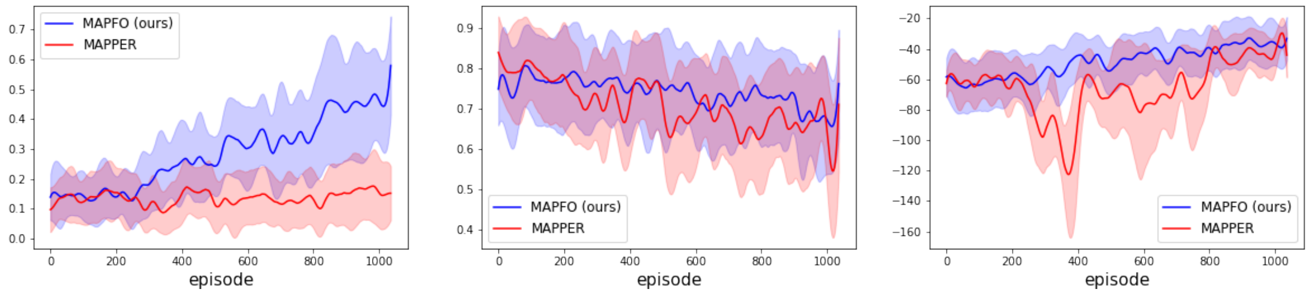
```
Initialize global agent weight $\Theta$ for actor-critic $\pi_\Theta$, $V_\Theta$ shared by $N$ agents
repeat
  // Data collection
  for $t = 1, \ldots, max\_step$ do
    // Rollout $N$ agents based on synchronized policy $\pi_\Theta$
    for $i = 1, \ldots, N$ do
      $a_t^i \sim \pi_\Theta(o_t^i; \Theta)$
    end for
    Execute action $a_t = [a_t^1, \ldots, a_t^N]$ and observe $(r_t^1, o_{t+1}^1), \ldots, (r_t^N, o_{t+1}^N)$
  end for
  Compute reward-to-go $\hat{R}$ for each agent
  Compute advantage estimate $\hat{A}$ for each agent using $V_\Theta$
  $D = \{\}$ // Trajectory data for all agents
  for $t = 1, \ldots, max\_step$ do
    for $i = 1, \ldots, N$ do
      $D = D \cup \{(o_t^i, a_t^i, r_t^i, o_{t+1}^i, \hat{A}_t^i, \hat{R}_t^i\})$
    end for
  end for
  // Parameter Update
  for $epoch = 1, \ldots, n_{epochs}$ do
    for mini-batch $k = 1, \ldots, K$ do
      $d_k \leftarrow$ mini-batch from $D$
      Adam update $\Theta$ on clipped surrogate $L(\Theta)$ with $d_k$
    end for
  end for
until Converges
```

IV. Experiments and Results

We experiment our algorithm on various environments by changing number of agents and obstacles. We compare success rate, collision and average reward of our method with the baseline MAPPER [1], which is a modification of Advantage Actor Critic (A2C) combined with evolutionary selection. Each agent in the baseline has its own neural network policy function which is updated with its own rollout. After designated number of gradient updates, the evolutionary selection step proceeds. The best performing agent is selected based on the accumulated rewards of the rollout. The selected agent's parameter is copied to each agent with probability proportional to the difference between rewards of the agent and the best performing agent.

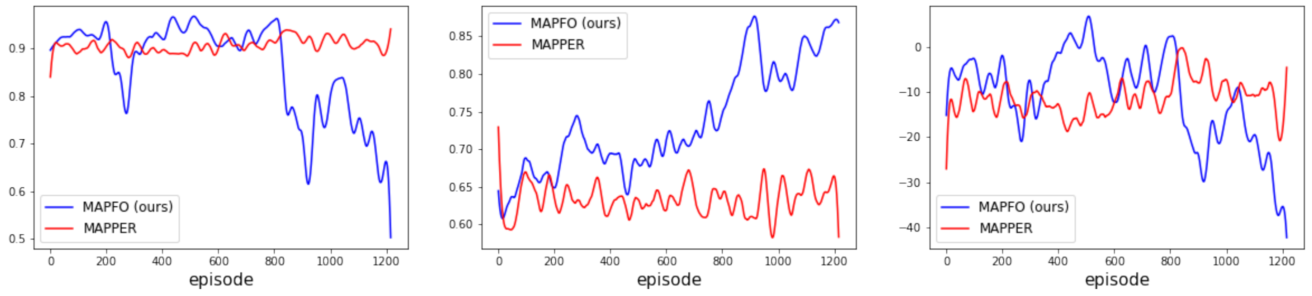
1) Stability: Figure 2 shows training result for environments with 8 agents and 10 dynamic obstacles. From success rate graph on the left, we can see that the proposed algorithm–MAPFO–performs significantly better than the MAPPER in this environment. The graph on the middle shows the collision rate of agents, which is the ratio of agents who had collision during its episode rollout. Compared to MAPPER, our method shows stable training behavior with the average reward increasing gradually. One cause of the instability of the baseline is due to the evolutionary update of unsynchronized agents, leading to drastic changes in the agents' policy. The synchronous update of our method provides better stability.

2) Curriculum Training: When increasing the number of agents and obstacles, the highly congested environment hinders agents from finding paths to their goals. Inspired by the



(a) Success Rate (b) Collision Rate (c) Average Reward

Fig. 2: Comparison between MAPFO and baseline MAPPER on 8 agents with 10 dynamic obstacles environment



(a) Success Rate (b) Collision Rate (c) Average Reward

Fig. 3: Comparison between MAPFO and baseline MAPPER on 30 agents with 10 dynamic obstacles environment. The agents are initialized with weights pretrained from 4 agents with 10 obstacles environment

curriculum learning paradigm, we first train the policy in a less congested environment with fewer agents and then train the agents in more congested environment. Figure 3 shows a result of this approach, where agents are trained to perform path finding in environments with 30 agents and 10 dynamic obstacles. Here, the both MAPFO and MAPPER initialize their training with the converged weights obtained from 4 agent and 10 obstacle scenario.

Although our method managed to gain high success rate in the beginning, it failed to converge. After 800 episodes, it began to perform poorly with high collision rate while the MAPPER performed consistently. We further discuss this phenomenon in the discussion section.

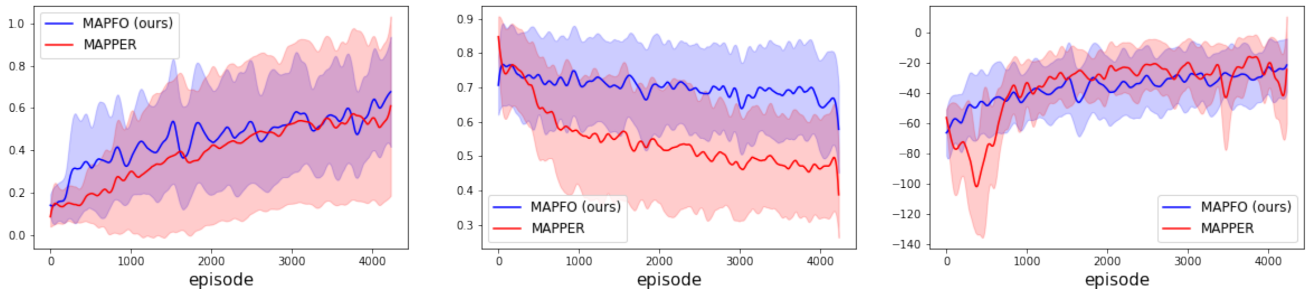
3) Robustness: We test the robustness of our method by modifying the learning rates. We train our agents on environments with 8 agents and 10 obstacles, with three different learning rates (0.0001, 0.0003, 0.0009) and draw mean and variance of the success rate, collision rate, and average reward on Figure 4. We can see from the success rate graph that the performance variance of our method is smaller than that of MAPPER. Although the collision rate of the baseline MAPPER is higher than that of ours, we found out that this was due to some agents in MAPPER refusing to move, remaining stationary during the entire rollout.

# agents	# obs	alg.	success rate	avg. reward	# collision
4	10	A2C	0.93 (± 0.12)	-26.2 (± 63.5)	13.1 (± 17.8)
		PPO	**1.00** (± 0.00)	**-21.0** (± 28.1)	**4.2** (± 2.1)
8	10	A2C	0.94 (± 0.07)	-72.6 (± 43.9)	7.9 (± 6.9)
		PPO	**0.99** (± 0.04)	**-22.6** (± 10.6)	**6.0** (± 1.9)
10	10	A2C	0.85 (± 0.11)	-103.4 (± 75.4)	11.5 (± 8.0)
		PPO	**0.96** (± 0.07)	**-39.4** (± 18.8)	**7.6** (± 4.2)
30	10	A2C	**0.97** (± 0.04)	**-24.5** (± 19.4)	11.9 (± 5.9)
		PPO	0.95 (± 0.04)	-33.7 (± 35.7)	**11.1** (± 5.7)
30	70	A2C	0.98 (± 0.02)	-9.8 (± 14.1)	**14.7** (± 2.8)
		PPO	**0.99** (± 0.02)	**-8.6** (± 13.3)	16.0 (± 3.9)

TABLE II: Test time statistics in different environment settings. The best performing (highest success rate) agent during training was chosen for the evaluation. Agents in 30 agent environments are trained from agents initialized with weights pretrained in the environment with 4 agents 10 dynamic obstacles (first row).

V. Discussion

The statistical results of MAPPER and MAPFO for various scenarios are tabulated on Table II. They are tested using the weights with highest success rate during the training sequence. The mean and standard deviations of three performance metrics are obtained through running 10 episodes for each environment settings (different start and goal locations). We observed that our algorithm show robust learning curves and better performance compared to baseline, which corroborates that our hypothesis – synchronized PPO learns more stably (lower variance and positive learning slope) than A2C with



(a) Success Rate (b) Collision Rate (c) Average Reward

Fig. 4: Comparing robustness to learning rate between MAPFO and baseline MAPPER in 8 agents with 10 obstacles environment.

evolutionary update– is valid. However, we also observed a frailty of our algorithm in curriculum training which we discuss in the following subsection.

A. Training

We found that our MAPFO algorithm fails to train well with the full-batch update. Hence we used a mini-batch of size 32 to enhance the performance of our algorithm. The mini-batched update helped sparse goal-reaching rewards take more effect during training and helped come out of the local minima, making the training successful.

B. Limitations

We initially observed that our method cannot train fresh in highly congested environment with high number of agents and obstacles, specifically neither MAPPER nor our method for scenarios with 20 agents and above struggled to learn. Thus we implemented curriculum learning strategy. Despite our method outperformed the baseline MAPPER in the freshly training settings, we found in Figure 3 that it failed in the curriculum setting. Although the success rate and average reward improved in the first couple hundred episodes, it increased the collision rate, driving the agent's policy to fall in a pitfall and perform poorly. Our method was not robust enough to withstand this drastic change in environment, while MAPPER survived with help of evolutionary update. We believe this issue can be solved by giving a better curriculum or by implementing automated approach to gradually increase the difficulty of the learning without expert supervision.

VI. Conclusion

This paper proposes a deep reinforcement learning approach for solving decentralized multi-agent path finding problem where agents find paths to the goals using local observation in dynamic environments.

We propose an algorithm where agents in the environment have the same policy and all trajectories contributes to the final policy hence increasing the sample efficiency. Combined with the clipped surrogate loss in PPO, we stabilize our training and mitigate the brittleness to hyperparameters.

We verified from our results that our method is more stable and less brittle to the learning rates. For highly congested settings with increased number of agents, we verified that giving a curriculum: train the agents first in low-congestion environment, helps the training. Although we have obtained an agent that outperforms the fully trained MAPPER by early stopping, our method soon diverged afterwards as the collision rate increased.

As a future work, other techniques such as decaying learning rate, Generative Advantage Estimator [7] or value and feature normalizations could further improve the performance of our PPO based method.

VII. Acknowledgment

We genuinely appreciate Zuxin Liu for providing the source code of MAPPER.

References

[1] Z. Liu, B. Chen *et al.*, "Mapper: Multi-agent path planning with evolutionary reinforcement learning in mixed dynamic environments," in *2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2020, pp. 11 748–11 754.

[2] S. K. T. Huang and B. Dilkina, "Learning to resolve conflicts for multi-agent path finding with conflict-based search," *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[3] J. Schulman, F. Wolski *et al.*, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[4] G. Sartoretti, J. Kerr *et al.*, "Primal: Pathfinding via reinforcement and imitation multi-agent learning," *IEEE Robotics and Automation Letters*, vol. 4, no. 3, pp. 2378–2385, 2019.

[5] M. Damani, Z. Luo *et al.*, "Primal$_2$: Pathfinding via reinforcement and imitation multi-agent learning - lifelong," *IEEE Robotics and Automation Letters*, vol. 6, no. 2, pp. 2666–2673, 2021.

[6] C. Yu, A. Velu *et al.*, "The surprising effectiveness of ppo in cooperative, multi-agent games," 2021.

[7] J. Schulman, P. Moritz *et al.*, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.