Utilization of a solution to the Drunken Philosophers Problem in a Deep Reinforcement Learning based Lifelong Path Planning Policy

Ezra Altabet, ealtabet@umich.edu

***Abstract*— Multi-agent asynchronous path execution is a challenging problem. Recent Work by Yunus Emre Sahin demonstrated that that this problem can be remapped into the well known Drunken Philosophers Problem (DrPP), and known solutions can readily be applied to multi-agent path execution problems under certain path conditions. Yunus' solution to the DrPP allows for collision avoidance, fairness, and deadlock-freeness. This paper aims to expand on his work by training a Lifelong Path Planning Policy to take advantage of the benefits provided by his solution to the DrPP. Initial results demonstrate promise, but further training and testing is needed. The code for this project is available at `https://github.com/Ealt1/ROB590-W23`.**

I. Introduction

The goal of the paper upon which this work was built was that given a collection of paths in a multi-agent system, one path for each agent, to devise a distributed protocol so that all robots guaranteed to reach their targets in the face asynchrony and avoid all collision along the way. This is called the Multi-Robot Path Execution (MRPE) problem. The goal was such that execution policies generated could be used with any suitably abstracted continuous-dynamics and low-level control policies that allow stopping when needed. Moreover their execution policies needed to be able to interfere with higher-level decision-making modules that lead to asynchrony by temporarily stopping the individual robots for emergencies, maintenance, or other applications. The authors realized that the MRPE problem could be recast as a drinking philosophers problem (DrPP) an extension of the well-known dining philosophers problem. DrPP is a resource allocation problem for distributed and concurrent systems, and by partitioning the workspace into a set of discrete cells and treating each cell as a shared resource, drinking sessions can be constructed such that the MRPE can be solved using any DrPP algorithm. In the multi-robot setting, this allows for collision avoidance, fairness, and deadlock-freeness [1]. The details of Yunus' DrPP algorithm are beyond the scope of this paper, but interested readers may find his work available at `https://arxiv.org/pdf/2001.00440.pdf`

II. Personal Contributions

In order for there to be a DrPP solution to an MRPE, the initial paths must satisfy three conditions:

1) Initial drinking sessions are disjoint for each robot.
2) Final cells of each robot are not do not overlap.
3) There exists at least one free cell in each agent's path.

Although mapping from MRPE to DrPP has the benefits of collision avoidance, fairness, and deadlock-freeness, it cannot be immediately integrated to be used by any path planner since all paths collectively must meet the above conditions. Furthermore, even if all paths of a multi-agent system did meet the above constraints, a path that is optimal for a single-robot individually may not continue to be optimal when integrated into the MRPE. Therefore, the first goal of this project was to design an optimized multi-agent path planning policy that could automatically generate paths that meet the above constraints while also minimizing individual excursion time of a given agent, and minimizing computational resources. This seeks to achieve a balance of safety guarantees and performance.

The second goal of this project aims to extend the DrPP solution to the lifelong planning scenario. While in simulation there is an end once all agents have completed their tasks, in the real world, a robot will need to go to a new destination once it has completed its task at its previous one. Therefore the path planner must be excursion and resource efficient not just over one excursion, but over multiple excursions.

III. Methods

This project aimed to achieve both goals by developing a multi-agent decentralized path planner via deep reinforcement learning. Because multi-agent deep reinforcement learning is fairly new, package options were fairly limited. I developed a multi-agent environment using PettingZoo and trained the environment using RLlib. While the original code is in Matlab, because PettingZoo is a python package, and the at time of this paper, Matlab only supports very limited multi-agent reinforcement learning, the final code is a python script utilizing the original Matlab script as a callable library to define and move agents in python. Further, at the time of writing, there are no environment packages that directly support asynchronous environments, and thus a parallel environment (ParallelEnv) was used. Therefore RLlib was used as it was the only reinforcement learning package that supported PettingZoo ParallelEnv.

A. The PettingZoo ParallelEnv

In order to utilize available deep reinforcement learning packages, the model was integrated into a PettingZoo ParallelEnv. PettingZoo is the multi-agent equivalent of OpenAI's Gymnasium environments, and ParallelEnv means that each agent produces their actions simultaneously instead of in a turn-based fashion. (To mimic an asynchronous environment a "no action" action was added, and will be discussed in greater detail.) The environment is outlined as follows:

1) Reset: The reset phase occurs at the beginning of the simulation. It initializes empty paths for each agent and other needed parameters. When simulation ends due to either truncations or terminations, the reset method can be called to reinitialize the environment.

2) Step: The step method will occur repeatedly until all truncations or terminations occur. This method accepts the actions of all agents as inputs and will alter the environment based on the commanded actions. Here the step method is divided into two phases: path planning and path execution.

- Path Planning: Prior to path planning, each agent will be given a list of valid actions, called an action mask. Action masks, when feasible (i.e. the action space is sufficiently small) are preferable to simply punishing invalid actions because they reduce the chances of the training algorithm sinking into local minima [2]. However, because action masking works by zeroing out the logits of invalid actions, in large action spaces it is often prohibitively expensive to attempt action masking by computing the validity of every action in the action space. Thus, while it is possible to have a whole path be made to be a single action, the action mask of the space would be too large to compute. Thus, to utilize action masking, the action space was modified to be an iteration of six possible actions:
  - 0: Stay in place
  - 1: Move left one square
  - 2: Move up one square
  - 3: Move right one square
  - 4: Move Down one square
  - 5: Pass

  The determination of valid actions is as follows: If the agent is not currently in motion, passing is not a valid action and the script determines the location that the first four possible actions will cause the agent to enter. If the Manhattan distance between that location and the target location is less than the number of moves remaining and the location is in bounds then the action is valid. If the agent is currently in motion, passing is the only valid action. It is important that the initial maximum number of allowed moves must be greater than the largest possible Manhattan distance between two points on the board, otherwise a scenario where no actions are possible may occur and an error will result. The selected action here will not actually cause the agent to immediately move, but will simply be added to the queue of actions to take in the next phase. Once an action has been taken, if all paths of all agents have been filled, then the script will continue to Path Execution phase. Otherwise, it will skip the Path Execution phase and allow a new set of actions to occur, repeating until all paths of all agent have been fully determined.
- Path Execution: If all paths have been fully determined, compute all drinking sessions for all agents. If any conflicts are registered such that DrPP will fail, provide a harsh punishment for affected agents and truncate all agents. (In the case of training, this will trigger an environment reset so that further learning my take place.) Otherwise, allow agents to move along their paths using the DrPP solution until either a robot has achieved its target position or a robot has reached its time limit for execution (this latter condition ensures a performance threshold to achieve and also handles agent stalling.) If the an agent reaches its goal, provide a reward (described below), set the current location of the agent to be the initial location of the agent's new trajectory and randomly give the agent a new target location that is not a current or target location of any other agents. If an agent stalls, provide punishment for that agent and truncate all agents. If the simulation runs for some large time threshold (in this case one thousand time steps), provide a large reward and terminate all agents. (During training this will also trigger an environment reset.)

3) Rewards: The following rewards are provided after each step method execution:

- If only path planning occurred and not path execution, provide a reward of 0.
- If path execution occurred but failed either due to agent conflict or time limit elapsing for an agent, provide a harsh penalty (in this case -100000000) to each affected agent.
- If path execution occurred and an agent successfully completed a path, award the agent a reward minus the time taken to complete the trajectory. This is meant to promote agent "survival" while rewarding more efficient paths. (In this case 100 minus the time elapsed for that agent during excursion.
- If an agent survives for a considerable amount of time (in this case, 1000 time steps) provide a large reward (in this case 10000) to the agent.

4) Observations: The observations for a given agent after the reset method and each step method is as follows:

- The action mask of that agent
- The paths of all agents
- The initial locations of each agent for their path
- The current location of each agent, either the most recent location in their path queue if still planning or their current location on the map if still moving.
- The target locations of each agent
- A Boolean indicating if all agents are executing paths or if an agent is still planning

5) Truncations and Terminations: Truncations occur if an agent failed for any reason. In this case if any agent needs to be truncated it will trigger truncations for all agents. A truncation will occur if a DrPP conflict occurred or a time limit elapsed for an agent.

Terminations occur when the agent has completed its purpose and is no longer needed. In this case, terminations only occur if an agent is able to make it past the total time threshold. This value was abitrarily set to 1000 time steps, but can be set lower to be a more realizable goal or set

higher to promote longer performance if needed.

An example of a cycle of this lifelong environment can be found in the Appendix with Figure 4.

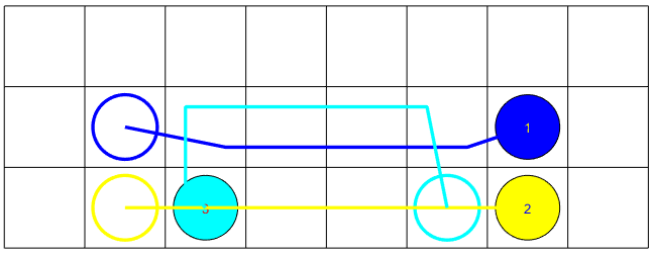

Fig. 1: Three agents on a 3x8 map. Solid circles are locations of each agent, hollow circles are their target locations, and the connecting lines are commanded paths of each agent between their current and target locations.

B. RLlib training

Once the PettingZoo environment was completed RLlib was used to to train the agents with Proximal Policy Optimization (PPO). Torch was used as the training framework as it has lower computational demand compared to Tensorflow. The training model was a fully connected neural network that handles action masking for an agent by setting the logits of invalid actions to zero. This ensures that its policy can only choose from valid actions. If any truncations or terminations are triggered this triggers an environment reset. The environment was trained using five rollout workers and two evaluation workers with an evaluation interval of two iterations. Due to time limitations, the policy was trained on fifty iterations at four thousand steps per iteration.

IV. Results

Although time limitations restricted the amount of training, initial results are promising. As can be seen in figure 2 increased training does yield additional average rewards. Since fewer steps also implies more randomness of actions, it can also be concluded that the currently trained algorithm performs better than random actions.

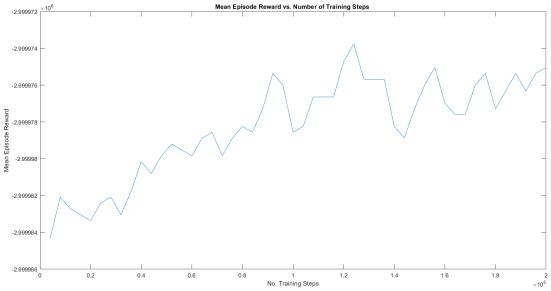

Fig. 2: Mean reward achieved over all agents for a given training iteration. (1 training iteration is 4000 steps.)

It is also clear as to the source of these rewards. As seen, the shape of average rewards in figure 2 is virtually identical to the shape of mean episode length in figure 3. This suggests that increased rewards throughout training are a result of the agents getting better at ”surviving” longer, not necessarily from becoming more efficient. This is makes sense since lasting for more iterations would result in the biggest initial reward increase. It is possible, however, that further training may induce more efficient path planning in late stages of training as increased survival yields diminishing rewards returns.

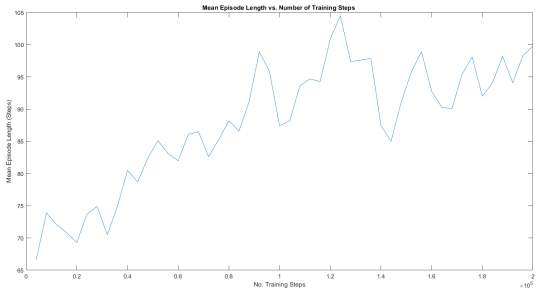

Fig. 3: Mean length of episode for a given training iteration. (1 training iteration is 4000 steps.)

V. Future Work

Although the work here is preliminary, initial results show promise using deep reinforcement as an approach to develop an optimized path planning policy for the Lifelong DrPP setting. However, there are a handful of ways by which training can be be improved. These include:

- Increased training. Due to time constraints, only a limited amount of training could be performed. However, initial results suggest the algorithm would benefit from simply training longer
- Randomization of the agent dynamics. It is possible that the trainer is implicitly learning the dynamics of the system rather than simply the properties of the DrPP solution. To address this, the reset method should be modified such that agent dynamics are randomized prior to a trial run.
- Adjust Network Build of RL Model. The current algorithm utilizes the default fully connected Torch network. Customizing this network to take advantage of properties specific to this scenario may promote learning.
- Rather than having a very large punishment and ending the simulation when the most recently made path conflicts with those of other agents and ending the episode, it may be better to give that agent a smaller punishment and have it try calculating its path again. This may limit the frequency of episodes ending as a result of agent conflicts and may allow the training algorithm to focus on path efficiency rather than agent survival.
- Reset to random map size and number of agents. The current policy was trained exclusively on three agents in a three by eight map. However, this approach may limit generalizability of the trained policy to other

numbers of agents or different size maps. To address this issue, future training may benefit by randomizing the number of agents and the map size during a reset so the algorithm can be exposed to a diversity of scenarios.
- Adding obstacles to the calculation of valid actions. This project calculated a valid action by determining if it was possible to achieve the target goal given that action and the remaining number of moves available to that agent. This calculation simply computed the Manhattan distance and compared it the remaining number of moves. However, more complex computation would likely be needed to evaluate a possible path given the remaining moves as well as the number of obstacles. This may be simplified by removing the constraint on the number of moves but then this might lead to poor convergence to the target location of the agent.
- Adding a module for local bottle sharing. Currently every time an agent begins a new path, all bottles between all agents are redistributed. However, this can be computationally expensive. It may be beneficial to incorporate a module to check which agents have overlapping paths, and only redistribute bottles amongst those agents.
- Anticipation. In the DrPP solution used by this project, if two agents have the same goal then the collective of paths is invalid. However in reality it may often be the case that two agents have the same destination, but given that they are intended to arrive at different times, this is a nonissue. It may be worth investigation into a relaxed version of the DrPP solution that takes this anticipation into account.
- Heterogeneity. In this project all agents were identical. It may be worth exploring scenarios with a heterogeneous collection of agents, for example, agents that have differing valid locations or different dynamics from each other.

VI. Appendix

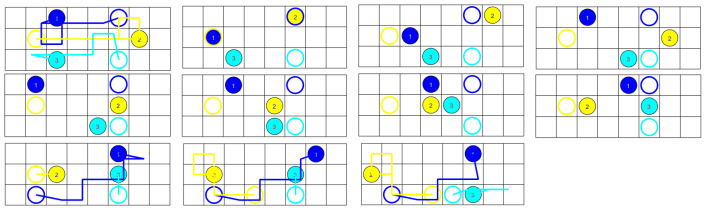

Fig. 4: Demonstration of the environment via random actions. Initial conditions of the first paths are on the top right. In the ninth panel the dark blue agent reaches its target. A new goal is generated for this agent and the agent randomly generates a path between its current location and the new target location. In the following panels, yellow and light blue agents also reach their goals. New target locations are randomly generated and the agents randomly generate paths from their current locations to the newly assigned target locations.

Acknowledgement

I would like to acknowledge the contributions of several others in the development of this report. Dr. Necmiye Ozay has worked with throughout the project so that I may better understand the problem and has been a sounding board for many of my ideas throughout the project. Yunus Emre Sahin created the original MATLAB code that this project heavily utilizes, and this report in large part is an extension of his work.

I would also like to thank Elliot Tower, one of the maintainers of PettingZoo, for taking the time to help me understand PettingZoo and how it could be adapted for my application.

References

[1] Y. E. Sahin and N. Ozay, "From drinking philosophers to asynchronous path-following robots."

[2] S. Huang and S. Ontañón, "A closer look at invalid action masking in policy gradient algorithms," *CoRR*, vol. abs/2006.14171, 2020. [Online]. Available: https://arxiv.org/abs/2006.14171