Enhancing Pathfinding Agents with Pheromone

Mingkun Yu

***Abstract*— Ant Colony Optimization is a metaheuristic strategy where agents leave values in external memory called pheromone which is later used to guide the search. In this report we try to use ACO-style pheromone strategy to enhance naive feedforward agents to navigate through partially observable 2D maze. We experiment on several different strategies, and report the result.**

***Index Terms*— Any Colony Optimization, Deep Reinforcement Learning, Partially Observable Markov Decision Process**

I. INTRODUCTION

It is very common to use maze navigation as a benchmark task in reinforcement learning. A 2D maze, in its most straightforward format, is a plane graph with M*N grids. Each grid is either walkable or not for agents, and an agent could move up, down, left or right 1 grid each time if the grid it tries to move to is walkable. There are destinations scattered in the maze, and the goal of the agents is to move to these destinations as soon as possible. One common extension to this setup is to introduce partial observability, where agents can only see the landscapes of the maze within a predefined vision range. Another common extension is to have multiple agents and multiple goals, and set up the reward to be cooperative or competitive.
Although seems simple, maze-running is a hard problem for reinforcement learning, especially for agents with no memory. For example, it is hard for these agents to explore long tunnels efficiently since their observation will remain the same throughout the whole exploration. What's more, having multiple destinations where the agent is required to visit all of them will also cause trouble since the agent can not remember if a specific destination has been accessed or not.
To solve these problems, one typical solution is to introduce memory, either internal or external. Internal memory is carried by an agent, is typically not accessible by other agents, and do not scale with the size of the maze. An external memory on the other side, is placed on the maze, is accessible by all agents, and scale only with the size of the maze, but not the size of the parameters of the agent. In this project, we will be looking at an external memory, specifically pheromone-style memory which was first introduced in Ant Colony Optimization[1].

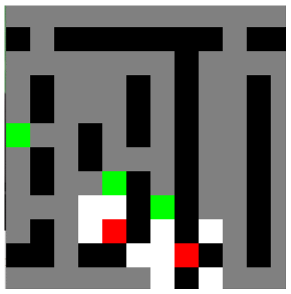


Fig. 1: An example of the maze with 2 agents and 3 destinations. Each black tile represents an obstacle, each red tile represents an agent and each green tile represents a destination. White tiles are walkable tiles that within sight range of agents, while grey tiles are walkable tiles out of sight range.

Ant Colony Optimization(ACO), originally appeared in 1997, is a meta-heuristic search algorithm inspired by the behavior of ants. Ants leave pheromones, which can be treated as a form of external memory, on the ground to help them remember specific information and to communicate with other ants. Inspired by this, ACO uses a number of agents that search and construct solutions independently to the problem that it tries to solve and leave "pheromones" at the same time that is used to guide the further search. In this project, we adopt the idea of using pheromone from ACO and apply it on maze running to enhance deep reinforcement learning agents. This project is built upon OpenAI gym and baselines, and the source code can be found at *https://github.com/RikonYu/898Project.*

II. BACKGROUND

A. Memory Enhanced Deep Reinforcement Learning

It is well known that current reinforcement learning algorithms are not capable of solving partially observable mazes by themselves if the agent has no memory. Difficulty on tasks that only require short-term memorization can be levitated by feeding consecutive frames(e.g., past 24 frames and the current one) into the agent, but this would not work for tasks that require relatively long term memorization, for example, maze running. To tackle this problem, recent research has been focusing on introducing memory, either internal or external. Also, these techniques can be put into two categories according to whether they write to their memory or not. The approach that combines naive reinforcement learning techniques with recurrent neural network[2] can be viewed as a form of mutable internal

memory. There are also works that incorporate a separated but still differentiable memory scheme into their neural network architectures, such as Memory Augmented Control Network[3].
Using static external memory is considered more of a trick than an algorithm in reinforcement learning, and one of the first research that uses mutable external memory is Neural Turing Machine(NTM)[4]. It uses an architecture that mimics both a modern Von Neumann computer and a Turing Machine while being end-to-end differentiable at the same time. Extensions of NTM includes Differential Neural Computer[5], which uses a combination of NTM and attention mechanism, and Reinforcement Learning Neural Turing Machine[6], which extends the model by adding more components and modifying the neural network controller to use LSTM. There are also works that focus on designing more complicated memory model instead of the way memory interacts with other components such as Neural Map[7].

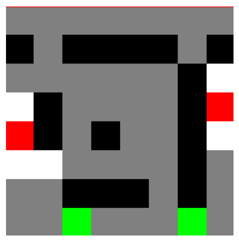
B. Ant Colony Optimization and Deep Learning

It is not new to use metaheuristics alongside deep learning for various purposes, including training the parameters directly, tuning the hyperparameters, evolving the structure, and so on. For example, Blum and Socha [8] constructs one-layer MLP with agents via Gaussian sampling, and exchanging Gaussian parameters as their pheromone. Li et al.[9] places agents on nodes of their neural network, let agents walk on the network and leave pheromones that are later used by backpropagation. Salama et al.[10] uses ACO to tune the number of neurons in a 3-layer MLP network for classification tasks. These parameter-training works are seen more often in the early stages of machine learning when neural networks are shallow and small. Thus they become less and less popular in the era of deep learning. In this project, however, we are merely using the idea of leaving pheromone traces as external memory instead of manifesting the network with ACO.

III. Experiment Setup

In this project, we use an 8x8 maze with a fixed landscape, while the initial position of agents and destinations are randomly generated. One episode consists of 128-time units, and the episode ends if agents have found all destinations by walking on them, or time runs out. Each grid in the maze has either obstacle or plain ground. At each time unit, all agents can move up, down, left or right 1 grid if the grid they are trying to move to contains no obstacles. An agent gains a reward of 1 if it moves on a destination, and -$\frac{1}{128}$ otherwise. It is guaranteed that all agents and destinations spawn at non-overlapping positions, although agents can cross each other with no collision during the exploration. It is also guaranteed that all destinations are always accessible by all agents. For simplicity, we will be dealing with single-agent single destination(SASD) case most of the times. All experiments conducted below are dealing with SASD unless otherwise specified. All agents have the same vision, and their vision are unobstructed, that is, they can see behind obstacles. Therefore, the observation of an agent is everything within the 2*vision+1 x 2*vision +1 square centering at itself. It can see terrain, destinations, other agents and trace within this range, but it can not see whether a destination has been accessed or not, not can it access the internal state of other agents. In multiagent case, all agents share the same policy, same pheromone laying strategy and same reward. They all have access to the public pheromone laid on the ground, but can not distinguish the source of it. Fig.2 contains an example of a maze setup.

As for the structure of deep Q learning, we adopt the CNN-mlp architecture used for Atari 2600 games by changing all the strides to 1 to fit the size of the maze. Below is the structure of the network.


(a)

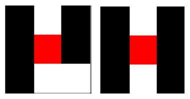

(b)

Fig. 2: An example of the maze layout. Left: the current global state of the maze. Black tiles are obstacles. White and grey tiles are walkable blocks. Red tiles are agents, and green tiles are destinations. A tile being white means that it is within sight of an agent. A grey tile means that it is out of sight. Right: observations of the agents. The 3x3 square at left is the observation of the left agent and the right one of from the right agent. The boundary of the maze is treated as an obstacle, and agents can not walk out of it.

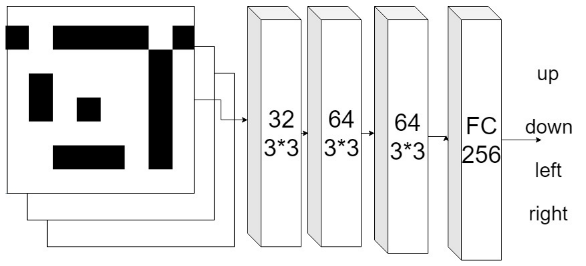


Fig. 3: The structure of the neural network. We stack obstacles, terrain, other agents, destinations and pheromone within the sight of an agent into a 3D "image "as its input. The input is then fed into a convolution layer with 3*3 kernels and 32 channels. The output is fed into a convolution layer with 3*3 kernels and 64 channels, then another convolution layer with 3*3 kernels and 64 layers. Next the output flattened and fed into a fully connected layer with 256 units. Finally the output is fed into an output layer with 4 units, who are the Q values for each action taken at the current location of the agent.

IV. Methodology

From a theoretical point of view, the value of pheromone left by each agent on the ground has 2 possible sources: using existing values from the network or generated for this specific purpose. In this project, we will focus on using existing values.

There are also a various number of reinforcement learning algorithms, including Deep Q Learning family([11],[11]), Policy Optimization family([12],[13]), Actor-Critic family([14]), and so on. To easily track and interpret the state, we will be focusing on deep q learning in this project.

A. Painting Q values

One of the most straightforward ideas would be directly using Q values as the pheromone. An agent at (x,y) paint Q((x,y),up), Q((x,y),down), Q((x,y), left) and Q((x,y),right) on ground. Intuitively, this would help the agent to remember global Q values and pass it through pheromone. The result is shown in Fig.4.

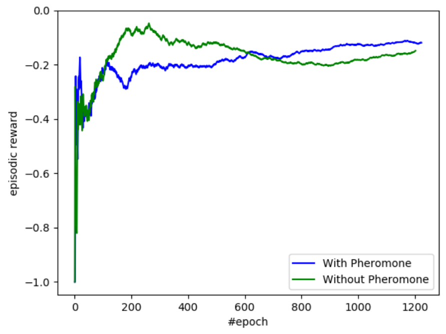


Fig. 4: Average episodic reward of simply use Q values as pheromone vs. not using pheromone at all during training stage on random generated starting position and destination.

As we can see, this idea does not work very well. A likely cause is that the agent would learn to pick up pheromone and translate it into its Q value via simple transformations (e.g., linear function). This would work well if the destination is fixed, and the agent has visited the destination enough times. If the location of the destination is random however, it is expected to have worse performance. The result is shown as follows.

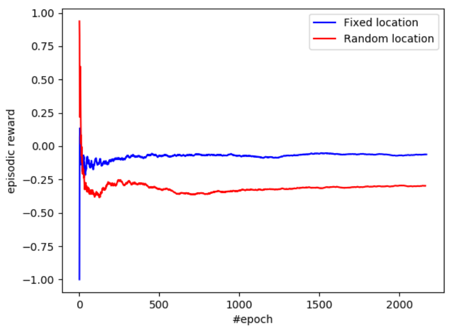


Fig. 5: Average episodic reward for using Q value as pheromone on randomly located destination vs. fixed destination. In the fixed case the destination is always located at the bottom right corner while the agent always spawns on the top left corner.

Therefore, we would need to set up a strategy for agents to paint pheromone such that it could use the pheromone to recognize *The structure of the maze* instead of the location of the destination.

B. Painting Constant

When an agent fails to find the destination, its most common behavior is to wander back and forth in a very small space, most likely a corridor. Therefore, painting a constant on ground would probably help the agent to recognize these corridors and avoid them. Fig.6 is the result of the experiment.

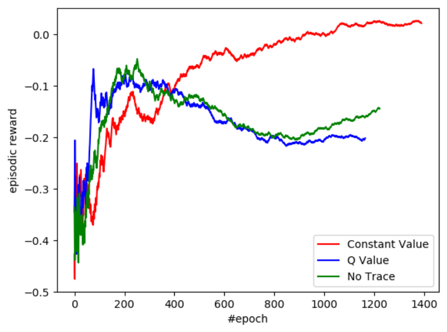


Fig. 6: Average Episodic reward of using nonzero constant value vs. using Q value vs. using no pheromone at all on randomly located destination.

Although the new strategy obtains higher reward, the agent will still fail if itself or destination spawns in one of the corridors. This is caused by the fact that it cannot remember which direction it came from. But in a corridor, it has to remember the direction of the entrance in order to get out after exploring it. To tackle this problem, we will need to make pheromone to include directional information. One straightforward idea is to leave pheromone only for the direction where the agent came from. That is, assume an agent starts at (x,y) and moves downward to (x+1,y), instead of leaving trace for ((x,y),up),((x,y),down),((x,y),left),((x,y),right), it now only leaves trace for ((x,y),down), and the rest 3 pheromone values will be left unchanged.

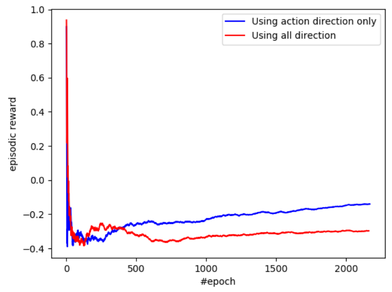


Fig. 7: Average Episodic reward of leaving constant pheromone value on all direction vs. leaving constant pheromone value on the direction of previous action only.

V. Discussion

This project shows that with simple strategies pheromone can provide enhancement to naive DQN when navigating through a partially observable maze. We have shown that simply using existing values in agent's network as pheromone is beneficial for navigation through partial observability. Since agents do not need to distinguish between each other, this strategy would probably scale well for multiagent case.

While there is clear improvement from naive approach, there are still many limitations of this project. First, it is now commonly believed that strategies that are learned end-to-end are better than those that are manually crafted. Therefore, it would be nice to build an end-to-end approach that could learn the pheromone placing strategy automatically. Unfortunately, such approach will go beyond naive DQN and would require algorithms that support continuous action(since writing pheromone on ground is a form of action with continuous value), such as the Policy Optimization family. Training the agent with policy optimization would require more time than this project allowed, and would probably be done in the future if possible.

In addition to the pheromone strategy, reinforcement learning itself also has unstable performance. For example, when leaving constant value as pheromone when initial pheromone is zero across the whole map, in theory this constant value should not matter. But in practice it actually has significant impact on the learned outcome. This is likely to be caused by the initialization of convolution layers, who have positive value at start.

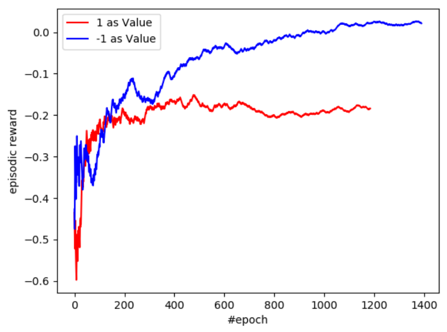


Fig. 8: Average episodic reward for using -1 and 1 as constant pheromone value. In theory they should converge to a similar behavior, but in practice they don't.

Therefore, we need to train the model more times to mitigate this problem, but unfortunately the time required for this is also beyond this project.

References

[1] Marco Dorigo and Luca Maria Gambardella, "Ant colony system: a cooperative learning approach to the traveling salesman problem," *IEEE Transactions on evolutionary computation*, vol. 1, no. 1, pp. 53–66, 1997.

[2] Thomas E Portegys, "A maze learning comparison of elman, long short-term memory, and mona neural networks," *Neural Networks*, vol. 23, no. 2, pp. 306–313, 2010.

[3] Arbaaz Khan, Clark Zhang, Nikolay Atanasov, Konstantinos Karydis, Vijay Kumar, and Daniel D Lee, "Memory augmented control networks," *arXiv preprint arXiv:1709.05706*, 2017.

[4] Alex Graves, Greg Wayne, and Ivo Danihelka, "Neural turing machines," *arXiv preprint arXiv:1410.5401*, 2014.

[5] Alex Graves, Greg Wayne, Malcolm Reynolds, Tim Harley, Ivo Danihelka, Agnieszka Grabska-Barwińska, Sergio Gómez Colmenarejo, Edward Grefenstette, Tiago Ramalho, John Agapiou, et al., "Hybrid computing using a neural network with dynamic external memory," *Nature*, vol. 538, no. 7626, pp. 471, 2016.

[6] Wojciech Zaremba and Ilya Sutskever, "Reinforcement learning neural turing machines-revised," *arXiv preprint arXiv:1505.00521*, 2015.

[7] Emilio Parisotto and Ruslan Salakhutdinov, "Neural map: Structured memory for deep reinforcement learning," *arXiv preprint arXiv:1702.08360*, 2017.

[8] Christian Blum and Krzysztof Socha, "Training feed-forward neural networks with ant colony optimization: An application to pattern classification," in *Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*. IEEE, 2005, pp. 6–pp.

[9] Jeng-Bin Li and Yun-Kung Chung, "A novel back-propagation neural network training algorithm designed by an ant colony optimization," in *2005 IEEE/PES Transmission & Distribution Conference & Exposition: Asia and Pacific*. IEEE, 2005, pp. 1–5.

[10] Khalid Salama and Ashraf M Abdelbar, "A novel ant colony algorithm for building neural network topologies," in *International Conference on Swarm Intelligence*. Springer, 2014, pp. 1–12.

[11] Shixiang Gu, Timothy Lillicrap, Ilya Sutskever, and Sergey Levine, "Continuous deep q-learning with model-based acceleration," in *International Conference on Machine Learning*, 2016, pp. 2829–2838.

[12] John Schulman, Sergey Levine, Pieter Abbeel, Michael I Jordan, and Philipp Moritz, "Trust region policy optimization.," in *Icml*, 2015, vol. 37, pp. 1889–1897.

[13] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.

[14] Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, Alex Graves, Timothy Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," in *International conference on machine learning*, 2016, pp. 1928–1937.