Adjust Planning Strategies to Accommodate Reinforcement Learning Agents

Xuerun Chen*
National University of Defense Technology
*chenxuerun18@nudt.edu.cn

Abstract. In agent control issues, the idea of combining reinforcement learning and planning has attracted much attention. Two methods focus on micro and macro action respectively. Their advantages would show together if there is good cooperation between them. An essential for the cooperation is to find an appropriate boundary, assigning different functions to each method. Such a boundary could be represented by parameters in a planning algorithm. In this paper, we create an optimization strategy for planning parameters, through analysis of the connection of reaction and planning; we also create a non-gradient method for accelerating the optimization. The whole algorithm can find a satisfactory setting of planning parameters, making full use of the reaction capability of specific agents.

Keywords: reinforcement learning, planning, deep learning, agent control, pattern search, graph search.

1. Introduction
The solution of many continuous decision problems can be described as such a process: agent set out from the initial state, then go through a series of the intermediate state and finally reach the goal state. Imagine an agent in a maze, which needs to find some key positions and pass through them one by one to get out.

The agent has two types of behavior: one is the micro action taken at every state, which is similar to muscle activity, called reaction; another is the change of trend in reactions taken over a period of time, which is similar to the thought of human, called planning[1]. For the agent in a maze, the reaction can be its every little moving step and planning can be its every determination of the position it should reach next.

In complicated scene with a high-dimensional data stream, long-term decision process, and sparse supervision signal, an agent trained only to react[2-3] can hardly perform well (See Appendix A for demonstration). However, combining reaction and planning[4-6] can significantly improve its capability.

The essence of such improvement is that agent has limited reaction capability and the introduction of planning releases the agent from reacting in the whole task. If the agent in the maze only knows how to reach a nearby position, with consecutive adjacent positions given by a planner, it still has the ability to reach a specified remote position.

What is the connection between reaction and planning? Improving reaction capability means spending much time training a connecting structure[7,8] before the task. Planning is not required once
the reaction capability reaches a certain level, which is difficult in a complicated scene (See Appendix B for demonstration). Improving planning capability means designing a planner that can provide useful information for the reactor or divide the original task into easier tasks. Planning would consume many more resources during the task.

Considering the features of reaction and planning, there should be a way to make full of their advantages: giving agents enough reaction capability without consuming much resources before the task, and, ensuring well performance without consuming much resources during the task. To achieve this, an evaluation of the reaction capability is necessary, helping to get a compatible planner.

A recent work (SoRB)\(^9\) showed a novel way to handle problems in a complicated scene: agent first samples states as waypoints, next connect waypoints to get a planning graph, then finds the shortest path in the graph, and finally reacts along with waypoints on the shortest path. They found a powerful tool to incorporate planning techniques into RL: distance estimates obtained from RL.

Based on SoRB, this paper analyzes the effect of two planning parameters on planning: the number of waypoints in and the maximum edge length of the planning graph. An online adapting algorithm is then proposed, which can adjust the planning parameters base on the complexity of state place and reaction capability of agent. With this algorithm, the task will be handled with relatively little computational cost and a high success rate.

2. Background

Before using an online adapting algorithm, the agent has obtained a reactive policy trained by RL and got a planner which constructs a planning graph and uses Dijkstra's Algorithm to find the shortest path, dividing the original task by setting subgoals along the path. Two parameters as mentioned above have a significant effect on performance. An optimization strategy is created to find a satisfactory setting of the parameters. A modified pattern search method is created to accelerate optimization.

Goal-ConditionedRL: The state of the agent is determined by its current and goal state: \(s, g \in S\). At every state agent takes a reaction: \(a \in A\). It has a reactive policy: \(\pi \in S \times S \rightarrow A\). The environment of agent has a reward function: \(r \in S \times S \times A \rightarrow R\), and a transition function: \(P \in S \times S \times A \rightarrow S \times S\). The reactive policy is learned by DDPG\(^2\) algorithm. The agent has a function that assesses values of each pair of state and action: \(Q \in S \times S \times A \rightarrow R\). Ideally, \(Q(s, g, a) = E^\pi[\sum_{t=0}^{\infty} \gamma^t r(s_t, g, a_t)]\), where \(\gamma \in (0, 1)\) is a discount factor. By decreasing Bellman Error: \(|Q(s, g, a) - r(s, g, a) - y max_a Q(s, g, a)|\), Q values will approach to the ideal ones. By choosing the reaction of larger Q value: \(\pi(s, g) = max_a Q(s, g, a)\), better performance can be achieved. After alternately optimizing Q and \(\pi\), agent will get better reacting and evaluating capability.

Distance Estimates Obtained from RL: In order to construct a planning graph, the agent must estimate distances between every pair of waypoints without additional information. If the environment has a special setting\(^6\):\(r(s, g, a) = -1\) and \(\gamma = 1\), then DDPG algorithm will learn Q values that have a close connection to the shortest distance between two states and could be used to determine lengths of each edge in the planning graph.

Planning: The idea of combing planning in RL has been around for a long time\(^10,11\). A recent work\(^12\) uses CNN as a planner which conveys useful global information to reactive policy. Another work\(^13\) use hierarchical RL, where high-level controller set goals and low-level controller produce locomotion. Both controllers are trained in an actor-critic process. In this paper, waypoints are filled in state place in advance\(^6\) rather than generated dynamically during the task. Since agent can estimate the distance between two waypoints, it can search a path without additional learning.

Pattern Search: The two planning parameters are optimized according to the testing result of tasks. The optimization has no gradient. After changing parameters, the planning graph should also be changed which could take a long time. Therefore, pattern search is used to reduce optimization time. In the original pattern search method\(^14\), the searching interval is gradually reduced to precisely find the optimal value. In this paper, the searching interval is increased to quickly find the range of optimal value. Adjustments are made to ensure an appropriate termination of the search.
3. Algorithm
The online adapting algorithm has two parts: optimizing planning parameters and pattern search. The optimization strategy is derived from the analysis of the relationship between planning and reaction. Pattern search accelerates, gives a soft convergence circumstance to, and sets a termination condition on the optimization.

3.1 Optimizing Planning Parameters
The two planning parameters are changed according to three different testing results of tasks: agent reaches the goal successfully; the agent finds a path to the goal but cannot reach it; the agent cannot find a path to the goal.

Success: The shortest path is got by visiting waypoints in the planning graph using Dijkstra's Algorithm, which takes more time as the number of waypoints grows. If agent could reach the goal, we could try to set less waypoints to get a quicker reaction.

Cannot Reach: This means there is a pair of adjacent waypoints in the path that agent cannot move from one to another by the reaction. The shortest distance of two states is estimated by Q network trained through DDPG. This distance estimate is efficient but not accurate enough (See Appendix C for demonstration). In SoRB, three Q networks are trained together and distributional Q Values are used to ensure robust distance estimates. If the problem still exists given these, the reaction capability of agent must be overestimated. Therefore, the maximum edge length of the planning graph should decrease so that easier subgoals are set to the agent.

No Path: This means the start and goal state are not connected in the planning graph. It is caused by the sparsity of waypoints or edges. The solution is to add both two parameters which could bring more waypoints and edges into the graph.

Combining the above three situations, we can get an optimization algorithm, shown in Algorithm 1.

**Algorithm 1 Optimizing Planning Parameters Once**
Inputs are a search policy sp, a rollout function rf which tests the agent and returns frequencies of occurrence of each situation, and a threshold number eth ( = 0.05) which set a condition on changing parameters.

```
function UPDATE(sp, rf, eth)
    no_problem ← True
    result ← rf(sp)
    if result.rate_of_cannot_reach > eth then
        the maximum edge length decreases
        no_problem ← False
    else if result.rate_of_no_path > eth then
        both two parameters increase
        no_problem ← False
    end if
    if no_problem then
        the number of waypoints decreases
    end if
end function
```

3.2 Pattern Search
The aim of using pattern search is to quickly determine the range of optimal value, and then narrow this range. The planning parameters may fluctuate to some extent but are close to the optimal value. Each parameter is optimized independently and an extra group of parameters are used to record its
optimization status. The process of pattern search is described below, taking the number of waypoints (denoted by $w$) as an example.

Initially, $w$ is set to a small enough value. The increment of $w$ (denoted by $i$) is larger than the decrement (denoted by $d$). The reason for such a setting is that smaller $w$ is more likely to cause a failure while larger $w$ increase task time which is relatively tolerable. A larger $i$ could avoid $w$ from converging into a dangerous area.

In the beginning, $w$ increases continuously and $i$ increases exponentially which makes $w$ far exceed the optimal value. Then $w$ is set to its last searching value, $i$ is set to its initial value and $w$ will continue increasing. After several such repetitions, the optimal value is determined within a small range. Now we can fix $i$ and try to reduce $w$. When 'No Path' happens, meaning that $w$ may enter the dangerous area, it should increase. Although the agent may perform well in the following tasks, the risk still exists. Therefore, $d$ is reduced simultaneously to restrict an attempt at reducing $w$. As $d$ decrease, $w$ gradually moves away from the dangerous area and fluctuates in a small area. The search is terminated when $d$ is small enough. A clear process is shown in Algorithm 2. To further accelerate the optimization, another search process is provided (See Appendix E for detail).

#### Algorithm 2 Pattern Search

Given number of exponential search times: $n (= 3)$, increment of $w$: $i (= 3)$, decrement of $w$: $d (= 1)$, growth factor of $i$: $\rho (= 2)$, reduction factor of $d$: $\gamma (= 0.9)$, and termination threshold: $tth (= 0.1)$,

where $w$ (initialized to 1) denotes the number of waypoints.

```plaintext
k ← 1
when $w$ should increase :
  $w ← w + k \times i$
  if $n > 0$ then
    $k ← k \times \rho$
  else if $n = 0$ then
    $d ← d \times \gamma$
  end if
when $w$ should decrease :
  if $n > 1$ then
    $n ← n - 1; w ← w - \frac{k}{\rho} \times i; k ← 1$
  else if $n = 1$ then
    $n ← n - 1; w ← w - d; k ← 1$
  else if $n = 0$ then
    $w ← w - d$
  end if
when $d < tth$ :
  End search
```

4. Experiment

The experiments are taken in a 2D environment (See Appendix). First, a satisfactory parameter setting is got using the adapting algorithm. Then, one of the planning parameters is fixed to see the effect of another on the task time and success rate.

4.1. Changing Process of Planning Parameters
We first consider the situation when planning parameters converge. The meaning of parameters below can be found in Algorithm 1 and 2 where one of their setting in the experiments is given.

When \( w \) converges (concentrate on \( w \) since maximum edge length is changed along with \( w \)), it fluctuates around a certain value. On average, it goes up \( tth \) times every time it goes down \( i \) times. When it goes up, 'No Path' happens in at least \( cth \) of the tasks (assume the number is exactly \( cth \)). When it goes down, the frequency of 'No Path' is less than \( cth \) (assume the number is exactly \( \frac{cth}{2} \)). Then we can calculate the success rate of task when \( w \) is around its convergence value:

\[
sr = 1 - \frac{i \times \frac{cth}{2} + tth \times cth}{i + tth}
\]

Notice that \( i \) is much larger than both \( tth \) and \( cth \). Equation 1 can be simplified:

\[
sr \approx 1 - \frac{cth}{2}
\]

This means, using the adapting algorithm, we would finally get a convergent \( w \) value that makes agent success in \( (1 - \frac{cth}{2}) \) of tasks. Such prediction is not accurate enough since it is derived under many assumptions, however, it is useful for understanding the training result.

Figure 1: (Left) The larger \( w \), the higher success rate the agent will achieve. A larger \( i \) would lead to higher success rate because it is more inclined to avoid failure. (Right) For a smaller setting of \( i \), \( w \) gets stable around 400.

Two changing processes of planning parameters are shown. The setting of extra parameters in Fig 1(a) is the same as those in Algorithm 1 and 2 except that \( i \) is set to 10. In Fig 1(b), the setting is totally the same. Reaction capability also has an influence on the convergence value of \( w \) (See Appendix D for detail).

The randomness comes from two parts: waypoints and tasks are randomly sampled. In each iteration, there are 40 different waypoints settings, in each of which 5 different tasks are given.

4.2. Comparison of Different Planning Parameters Settings

In Fig 1(b) we get a satisfactory setting of planning parameters: \( w=406.3 \) and \( e=5.1 \), where \( e \) denotes the maximum edge length of the planning graph. Taking this setting as the center, we now compare task time and success rate in different parameter settings. We first fix \( e \) to 5 and change \( w \). Then we fix \( w \) to 400 and change \( e \). The results are shown in Fig 2(a) and 2(b).
The distances of each pair of waypoints are cached before the task so that the time complexity of searching the next waypoint is reduced from $o(w^2)$ to $o(w)$. This makes task time grows almost linearly with $w$. Failed tasks are not counted when calculating the average task time.

The experiments show that with an appropriate parameter setting, the pattern search can quickly find a satisfactory setting of two planning parameters. This method can extend to more sophisticated problems where there are more than three planning parameters to optimize without gradient, as long as the optimization strategy (similar to Algorithm 1) is given.

![Figure 2](https://example.com/figure2.png)

(a) different $w$  
(b) different $e$

Figure 2: (Left) Success rate improves little when $w$ and itself exceed certain values. The $sr$ derived from Equation 1 is close to the certain value, so is the converged $w$. (Right) Larger $e$ is more likely to cause occurrence of ‘Cannot Reach’, but provide shorter paths, leading to less task time. Smaller $e$ is more likely to cause ‘No Path’, and hide shortcuts, causing more task time and lower success rate.

5. Discussion and Future Work
Combining planning and reaction could help to handle complicated tasks which have a high-dimensional data stream, long term decision process, and sparse supervision signal. A good planning algorithm could make full use of limited reaction capability which is usually obtained by deep reinforcement learning. A specific planning algorithm has parameters that need changing to accommodate the reaction capability of agent. The optimization direction of these parameters could not derive from calculating gradients. Therefore, we need to Fig out the relationship between planning and reaction to create optimization strategy. After determining the strategy, improved pattern search method can be used to greatly accelerate optimization.

In this paper, the planning method creates a memory of state place where agent can get useful instructions during tasks. In the future, we could design a planner that creates and remove waypoints repeatedly, to form a more efficient memory of the environment. We could also try to simultaneously improve planning and reaction, which might bring us powerful agents with excellent reactions (see Fig 4) in the whole environment. Furthermore, agent needs to explore the environment when there is no waypoint initially.

Acknowledgement
The code is available on https://github.com/chenxuerun/APS

References
[1] Richard S S, Doina P, and Satinder S 1999 Between mdps and semi-mdps: a framework for
temporal abstraction in reinforcement learning Artificial Intelligence 112 pp181 – 211
[2] Volodymyr M, Koray K, David S, Andrei A R, Joel V, Marc G B, Alex G, Martin R, Andreas K F, Georg O, Stig P, Charles B, Amir S, Ioannis A, Helen K, Dharshan K, Daan W, Shane L, and Demis 2015 Human-level control through deep reinforcement learning Nature 518 pp529–533
[3] Timothy P L, Jonathan J H, Alexander P, Nicolas H, Tom E, Yuval T, David S, and Daan W 2016 Continuous control with deep reinforcement learning International Conference on Learning Representations (ICLR)
[4] Aleksandra F, Oscar R, Marek F, Kenneth O, Anthony F, James D, and Lydia T 2018 Prm-rl: long-range robotic navigation tasks by combining reinforcement learning and sampling-based planning. IEEE International Conference on Robotics and Automation (ICRA) pp 5113–5120
[5] Nikolay S, Alexey D, and Vladlen K 2018 Semi-parametric topological memory for navigation International Conference on Learning Representations (ICLR)
[6] Benjamin E, Ruslan S, and Sergey L 2019 Search on the replay buffer: bridging planning and reinforcement learning International Conference on Learning Representations (ICLR)
[7] David E R, Geoffrey E H, and Ronald J W 1986 Learning representations by back-propagating errors Nature 323 pp533–536
[8] Geoffrey E H and Ruslan R S 2006 Reducing the dimensionality of data with neural networks Science 313 pp504–507
[9] Leslie P K 1993 Learning to achieve goals IJCAI
[10] Peter D, Geoffrey E H, Hanson S J, Cowan J D and Giles C L 1993 Feudal reinforcement learning. Advances in Neural Information Processing Systems 5 pp271–278
[11] Richard S S, Doina P and Satinder S 1999 Between mdps and semi-mdps: a framework for temporal abstraction in reinforcement learning Artificial Intelligence 112 pp181 – 211
[12] Aviv T, Yi W, Garrett T, Sergey L and Pieter A 2016 Value iteration networks Proceedings of the 30th International Conference on Neural Information Processing Systems pp2154–2162
[13] Binpeng X, Glen B, Kangkang Y and Michiel V D P 2017 Deeploco: dynamic locomotion skills using hierarchical deep reinforcement learning ACM Trans Graph 36
[14] Robert H and Jeeves T A 1961 “direct search” solution of numerical and statistical problems. J. ACM 8 pp212–229
[15] Marc G B, Will D and Rémi M 2017 A distributional perspective on reinforcement learning. Proceedings of the 34th International Conference on Machine Learning 70 pp449–458
[16] Jason Y, Jeff C, Yoshua B and Hod L How transferable are features in deep neural networks? Proceedings of the 27th International Conference on Neural Information Processing Systems 2 pp3320–3328
[17] Max J, Volodymyr M, Wojciech M C, Tom S, Joel Z L, David S and Koray K 2016 Reinforcement learning with unsupervised auxiliary tasks. CoRR, abs/1611.05397
[18] Rouhollah R, Pooya A, Aman B and Ladislaus B 2016 Learning real manipulation tasks from virtual demonstrations using lstm