Finding the Shortest Path Using Reinforcement Learning

LINGFENG ZHANG, WENHAN HAO, & TIANMING QIU

Group: applied-rl17 / E3

1 Consice Introduction of Project

This project focuses on finding a shortest path in a map with many cross roads. The agent has to make an optimal decision to turn into a correct or best direction at each road crossing, namely the state in terms of Markov Decision Process.

2 Markov Decision Process Description

Obviously, the states in this model are the different road crossings. State transition, namely the transition of different road crossing, is adopted as a stochastic process. The modelling of this issue satisfies the Markov Decision Process (MDP). First, it has the Markov property. And the transition from road crossing C_t to next crossing C_{t+1} is independent of all the previous states and actions. No matter which road crossing the Epuck was in or which actions it takes before the road crossing C_t, the probability that it transits from current state to next state would be never influenced. In another words, it can be represented by: $P(s_{t+1}) = P(s_t, a_t)$. The current state captures all that is relevant about the world in order to predict what the next state will be. Secondly, it is clear to see that the state transition is related to action. Epuck can choose to turn right, turn left, go ahead or backward, and the consequence will be dependent on the chosen actions at current state.

A Markov decision process is a discrete-time state transition system. It can be described formally with 5 components as a 5-tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$ as below:

$\mathcal{S}$**: States** First, it has a finite set of states. These states will play the role of outcomes in the decision theoretic approach, as well as providing whatever information is necessary for choosing actions.

$\mathcal{A}$**: Action** A small finite set includes turn left, turn right, go forward and backward.

$\mathcal{S}$**: Transition probability** The transition probabilities describe the dynamics of the world. They play the role of the next-state function in a problem-solving search, except that every state is thought to be a possible consequence of taking an action in a state. So, we specify, for each state s_t and action a_t, the probability that the next state will be s_{t+1}. Now we just model the transition probability as the simplest one: All of them equal to one. That means if and only if Epuck takes a specific action (e.g. turn right) at the current action, it will transit into a definitely deterministic state.

r**: Reward** In this model we will assign a reward '+10' at the destination state and '0' at the other states. Also at the 'blind alley' it will get a very poor reward such as '-100'.

γ γ is a discout factor here we select as 0.9.

3 Algorithm

In recent years, reinforcement learning has achieved many impressive results, including playing video games, and training advanced manipulation skills. The goal of this experiment is to realize artificial agents, which can be trained and can find the shortest path in intricate traffic environment. Here we consider using An off-policy TD control algorithm: Q-learning to achieve our goal.

We define a discrete-time finite-horizon discounted Markov decision process (MDP) by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, r, \gamma)$. The agent interacts with an environment over a time of discrete time steps. At each time step, the agent receives a state from $\mathcal{S}$ and select an action from $\mathcal{A}$ according to its policy π, where π is a mapping from states $\mathcal{S}$ to action $\mathcal{A}$. In return, the agent receives the next state and a scalar reward r. The process continues until the agent reaches a terminal state. The total return is the accumulated reward with discount factor γ. The action value $Q^\pi(s, a)$ is the expected return for selecting action a in state s and following policy π. The optimal value function

$$Q^*(s, a) = \max_\pi Q^\pi(S, a)$$

gives the maximum, action value for state s and action a achievable by any policy.

In Q-learning, the action value $Q(s, a)$ will be updated toward one-step return as below:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Each time obtaining a reward r can directly affects the value of the state action pair s, a that led to the reward. The pseudocode of Q-learning algorithm is shown below:

Q-learning: An on-policy TD control algorithm

Initialize $Q(s,a)$, $\forall s \in \mathcal{S}$, $\forall a \in \mathcal{A}$, arbitrarily, and $Q(\text{terminal-state},\cdot) = 0$
Repeat (for each episode):
    Initialize S
    Repeat (for each step of episode):
        Choose A from S using policy derived from Q (e.g., ϵ-greedy)
        Take action A, observe R, S'
        $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
        $S \leftarrow S'$
    Until S is terminal

At each step, the agent will choose an action A according to current policy, and get the reward R. But the action value Q will be updated using another policy, here we choose the action which can maximize the next action value. That's why Q-learning is an off-policy algorithm, the policy we use to choose action and the policy we use to evaluate the value function are different. Here we prefer to the off-policy rather than on-policy algorithm, because off-policy algorithms are usually more efficient than on-policy.

4 Simulation

The goal of our simulation is to simulate the real traffic network. The e-puck will take on a role of explorer to find the shortest path between two points of the map.

To make our tasks tractable, we consider a grid-based representation with discrete states and action. The navigational task for our mobile robots could be projected into this presentation by employing a number of actions such as "turn left", "turn right", and "drive forward" that only use a low-level controller that takes care of accelerating, moving, and stopping. The different crossroad in the simulated map will be considered as states. We will perform our experiment in the simulator v-rap with our test map, which has 10 states.

In reinforcement learning, the desired behavior is implicitly specified by the reward function. The goal of our reinforcement learning algorithm is to find the optimized trajectory, which can maximize the accumulated long-term reward. In our experiment, we assign a reward '+10' at the destination state and '-1' at the other states. Also at the 'blind alley' it will get a very poor reward such as '-10' in order to let agent avoid those states. At the begin of

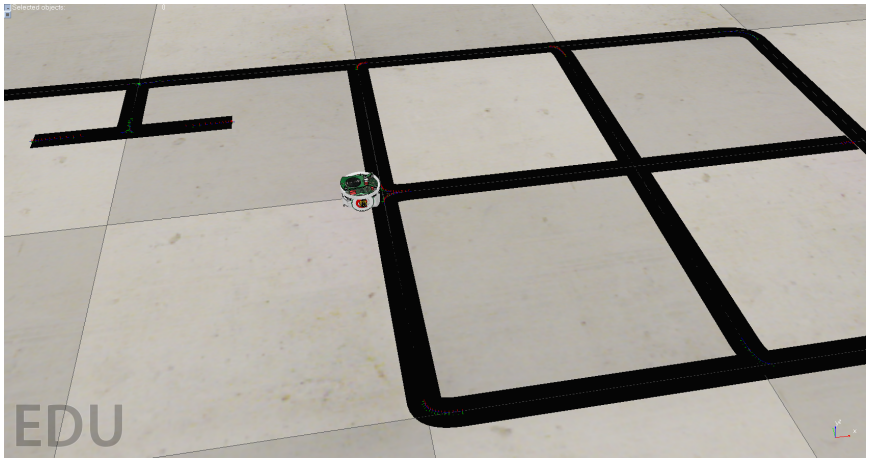

Figure 1: An example of simulation map

each simulation, two points will set up as the start point and destination, and the e-puck robot will be placed in the start point of the test map, and start to explore the test map. From one state to adjacent state, the robot will drive along the line we drawn on the ground. The under sensor on e-puck will scan the ground all the time in order to keep robot following the line and detect the label we use to distinguish state. Everytime the robot reaches a new state, the Q-value will be updated with the reward we received and a new action will be selected according to the current policy. Once e-puck reaches the terminal state(destination), it will be replaced in the same start point and start a new exploration. Through trial and error, finally the action value will converge at a point, which is called optimal action value, then we can find the optimal policy for this problem. Figure 1

5 Milestones

to be written to be written

References