ECE 276B: Planning & Learning in Robotics Project 1

Lin-Hsi Tsao (PID: A59015911)
Department of Electrical and Computer Engineering
University of California, San Diego
ltsao@ucsd.edu

Acknowledge: Wilson Liao (w4liao@ucsd.edu)

I. Introduction

Path planning is a crucial aspect of any robotic system, including household robots that must navigate between different locations to perform various tasks. To achieve this, the robot's software must be capable of making informed decisions based on an environment map created using techniques. One approach is to divide the robot's position and orientation into discrete states, each with an associated reward, where the highest reward corresponds to the final goal pose. The control actions the robot takes at a given state to move to another state constitute the control policy, and these transitions can be represented in a connected graph known as the Markov Decision Process (MDP) graph. By using algorithms like value iteration and label correction, we can determine the path with the lowest cost or highest reward through the MDP graph.

In this project, we work with a mini-grid environment containing an agent capable of performing actions such as Move Forward, Turn Left, Turn Right, Pick Key, and Unlock Door. The main objective is to reach the goal in the shortest path possible, and sometimes it is more efficient to reach the goal via the Door. However, there may also be instances where using the door is necessary, and other cases where it is not required. As a result, we must consider all possible scenarios and select the path that provides the highest reward or lowest cost.

II. Problem Statement

We solve the path planning problem using Markov Decision Process(MDP) and Dynamic programming(DP). In this section, we look into the problem formulation in further depth.

A Markov Decision Process (MDP) is a Markov Reward Process with controlled transitions defined by a tuple (X, u, p_0, p_f, T, l, q, γ). The definitions of the notation are as followed:

1) $X = \begin{bmatrix} AgentLocation \\ AgentOrientation \\ DoorStatus \\ KeyStatus \\ GoalStatus \end{bmatrix}$,
where X is a discrete set of states.

2) $U = \begin{bmatrix} MF & TL & TR & PK & UD \end{bmatrix}$,
where u is a discrete set of controls.

3) p_0 is a prior pmf defined on X. In this case, we have already known the starting point x_0 of the agent.

4) $p_f(\cdot|x_t, u_t)$ is a conditional pmf/pdf defined on X for given $x_t \in X$ and $u_t \in U$.

5) T is a finite time horizon, where T = number of grids in the case.

6) $l(x, u) = \begin{cases} 1, & u = MF \\ 0, & u \notin U \backslash MF \end{cases}$
where $l(x, u)$ is a function specifying the cost/reward of applying control $u \in U$ in state $x \in X$.

7) $q(x) = 0$, where $q(x)$ is a terminal cost/reward of being in state x at time T.

8) $\gamma = 0$, where γ is a discount factor.

The goal is to minimize the cost function generated by the agent moving to the goal from the starting point.

III. Technical Approach

In this section, the algorithm solving the problem would be expressed. The algorithm could be stated into three parts.

A. Decide if the key is required or not

The process of determining the shortest route involves four steps, which yield the following expenses:

1) C_{direct}: the cost of reaching the objective from the agent's starting point
2) C_{Key}: the cost of reaching the key from the agent's starting point.
3) $C_{key2Door}$: the cost of reaching the door from the key position.
4) $C_{Door2Goal}$: the cost of reaching the objective from the door position.

We check the following condition.

$$C_{direct} > C_{Key} + C_{key2Door} + C_{Door2Goal} \tag{1}$$

If the equation 1 is true, obtaining the key becomes imperative to attain the shortest path to the goal. Conversely, if equation 1 is false, it means that a shorter and more direct path from the initial Agent location to the goal already exists, rendering the key unnecessary to achieve the shortest path towards the goal.

B. Traverse the shortest path

1) If equation 1 is True, we transform the key position into the objective and establish the related cost grid. This helps us determine the complete cost of traversing from the initial Agent position to the key position. Utilizing the agent's orientation and location, we guide the agent to the key position by examining the four adjacent cells of a given Agent position and selecting the cell with the lowest cost. We then leverage the Agent's orientation to transition to the cell with the lowest cost among the four possible surrounding cells, generating the appropriate sequence and recording them in a list S1. We continue this process until we arrive at a cell neighboring the key position cell. There would be at most four candidate positions. Once we align ourselves with the key, we use the relevant motion command to pick it up and store it in S1.
2) After picking up the key, we recalculate the cost grid, but this time we could directly set the final goal position as the goal position since the door would be unlocked after the agent gets the key. Hence, we could assume that the every door could be passed by the agent. Then, we utilize the same method as discussed in the previous paragraph to generate the optimal sequence, which we store in a list S2. Since there will be at most 4 different sequences S2, we choose the sequence with the lowest cost as our final motion sequence.
   Finally we combine S1 and S2 to get the final action sequence i.e. Seq=[S1, S2].

IV. Results

This section shows the results of the above algorithm. The agent reached the final goal position using the shortest path possible. The image caption states the current problem scenario, and the optimal motion list is shown below it. Since I have not come up with an optimal policy for the random environments, the results are all for the known environments.

A. Known environments

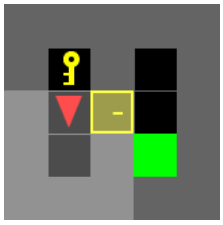

Fig. 1. Test case: doorkey-5x5-normal
The optimal motion: ['TR', 'TR', 'PK', 'TR', 'UD', 'MF', 'MF', 'TR', 'MF']

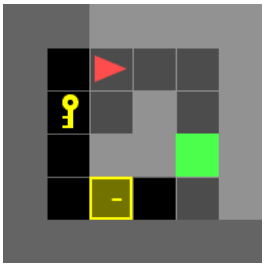

Fig. 2. Test case: doorkey-6x6-direct
The optimal motion: ['MF', 'MF', 'TR', 'MF', 'MF']

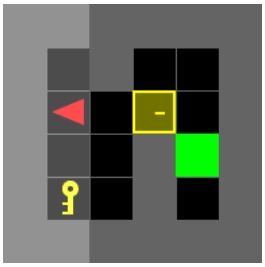

Fig. 3. Test case: doorkey-6x6-normal
The optimal motion: ['TL', 'MF', 'PK', 'TR', 'TR', 'MF', 'TR', 'MF', 'UD', 'MF', 'MF', 'TR', 'MF']

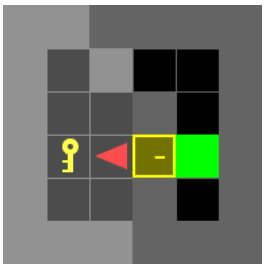

Fig. 4. Test case: doorkey-6x6-shortcut
The optimal motion: ['PK', 'TR', 'TR', 'UD', 'MF', 'MF']

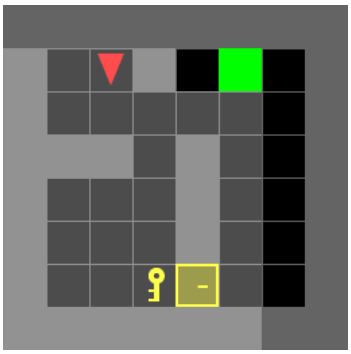

Fig. 5. Test case: doorkey-8x8-direct
The optimal motion: ['MF', 'TL', 'MF', 'MF', 'TL', 'MF', 'TR', 'MF']

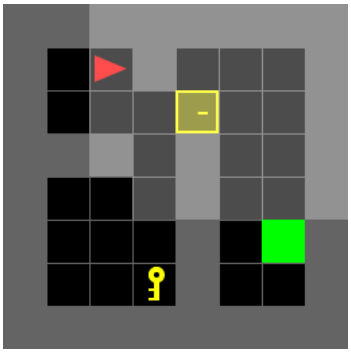

Fig. 6. Test case: doorkey-8x8-normal
The optimal motion: ['TR', 'MF', 'TL', 'MF', 'TR', 'MF', 'MF', 'MF', 'PK', 'TR', 'TR', 'MF', 'MF', 'MF', 'TR', 'UD', 'MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'MF']

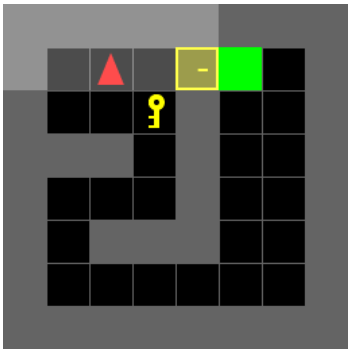

Fig. 7. Test case: doorkey-8x8-shortcut
The optimal motion: ['TR', 'MF', 'TR', 'PK', 'TL', 'UD', 'MF', 'MF']