ECE276B Project 1: Dynamic Programming

Wilson Liao
Department of Electrical and Computer Engineering
University of California, San Diego
w4liao@ucsd.edu

I. Introduction

Path planning is a critical problem in robotics that involves finding an optimal path from a starting position to a goal position while avoiding obstacles in the environment. The goal is to generate a sequence of robot configurations that will move the robot from the starting position to the goal position, while avoiding collisions with any obstacles in the environment. Typically, path planning problems challenging because of the following reasons: (1) High Dimensionality: Path planning involves searching through a large space of possible configurations of the robot, which can have a very high dimensionality. (2) Uncertainty: The environment in which the robot operates is often uncertain, and the robot must be able to adapt to changes in the environment in real-time. (3) Computational Complexity: Path planning algorithms must be able to handle large amounts of data and perform complex calculations in real-time.

Nevertheless, popular path planning algorithms are used in a wide range of applications, including autonomous vehicles, industrial automation, and robotic surgery.

In this project, we adopt Dynamic Programming and Markov Decision Process (MDP) to solve the path planning problem in a Door & Key environment, which consists agent , goal location, walls, key, doors (require key to open if it's lock). The control input (actions) for the agent are as follows: Move Forward (MF), Turn Left (TL), Turn Right (TR), Pick Key (PK) and Unlock Door (UD). We need to find a sequence of actions that minimize the path cost.

II. Problem Formulation

A. Markov Decision Process

We formulate the path planning problem as a MDP. Thus we need to define the state space X, control space U, motion model f, the initial state x_0, the planning horizon T, the stage and terminal costs l, q, and any other elements necessary to make this a well defined problem.

The state space and the control space of the agent can be defined as below:

$$X = \begin{bmatrix} \text{Position} \\ \text{Orientation} \\ \text{Door status} \\ \text{Key status} \end{bmatrix} \tag{1}$$

$$U = \begin{bmatrix} \text{MF} & \text{TL} & \text{TR} & \text{PK} & \text{UD} \end{bmatrix} \tag{2}$$

where position can be any accessible entity in the given 2D grid, orientation is the heading direction $\in \{[1,0],[-1,0],[0,1],[0,-1]\}$, door status indicates whether the doors are open/locked, key status indicates whether the agent carries a key.

However, we need to keep track of too many states in each time step based on this formulation. Thus, in order to reduce the complexity of this problem, we made the following assumptions:

- Only cost for MF control input (i.e. TL, TR, PK, UD has 0 cost)
- Stage cost for each state are zero (i.e. stage cost only depends on control input)

According to the assumption, we can simplify the state space into only the positions in the grid, while the control space remains the same:

$$X \in \{x, y\} \tag{3}$$
$$x \in \{0, 1, ...h-1\} \tag{4}$$
$$y \in \{0, 1, ...w-1\} \tag{5}$$

where h and w are the height and width of the grid respectively.

The initial distribution can be written as follows:

$$p_0(x) = \begin{cases} \frac{1}{hw} & \text{if } x \text{ is in the grid} \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

The remaining part of the MDP problem are shown in the next section for simplicity.

B. Dynamic Programming

A general method find the shortest path is to use dynamic programming. We can consider a DSP problem with vertices V, edges E, edge weights C, start node $s \in V$ and terminal node $\tau \in V$, time horizon $T = |V| - 1$.

The following are the definition for equivalence of DOC and DSP problems. The state space and control space are then re-defined as:

$$X = V \tag{7}$$
$$U = V \tag{8}$$

The motion model can be defined as:

$$x_{t+1} = f(x_t, u_t) = \begin{cases} x_t & \text{if } x_t = \tau \\ u_t & \text{otherwise} \end{cases} \tag{9}$$

The state cost and terminal cost can be defined as:

$$l(x,u) = \begin{cases} 0 & \text{if } x_t = \tau \\ c_{x,u} & \text{otherwise} \end{cases} \tag{10}$$

$$q(x) = \begin{cases} 0 & \text{if } x_t = \tau \\ \infty & \text{otherwise} \end{cases} \tag{11}$$

As the assumption we made in the previous section, the cost $c_{x,u}$ can be written as:

$$c_{x,u} = \begin{cases} 1 & \text{if } u \text{ is MF} \\ 0 & \text{otherwise} \end{cases} \tag{12}$$

Here, we adopt forward dynamic programming for the DSP problem, the following is the pseudo code of the algorithm.

Algorithm 1 Forward Dynamic Programming

```
**Input**: vertices $V$, start $s \in V$, goal $\tau \in V$, and costs $c_{ij}$ for $i, j \in V$
$T = |V| - 1$
$V_0^F(s) = V_1^F(s) = ...V_T^F(s) = 0$
$V_0^F(j) = \infty, \forall j \in V \setminus \{s\}$
$V_1^F(j) = c_{s,j}, \forall j \in V \setminus \{s\}$ for t=2,...,T do
    $V_t^F(j) = min_{i \in V}(c_{i,j} + V_{t-1}^F(i)), \forall j \in V \setminus \{s\}$
    if $V_t^F(i) = V_{t-1}^F(i), \forall i \in V \setminus \{s\}$ then
        break
```

III. Technical Approach

A. Forward Dynamic Programming

In the original Forward dynamic programming algorithm, it check all the nodes to update the optimal cost-to-arrive $V_t^F(j)$ in each time step. However, in our approach we only check the neighbors of the nodes we've checked in the previous time step, since only these nodes are accessible in the current time step. We use a queue to store these need-to-check nodes to perform a Breadth first search (BFS) during updating the cost-to-arrive $V_t^F(j)$. Thus, the update step in forward dynamic programming can be re-written as:

$$V_t^F(j) = min_{i \in Q}(c_{i,j} + V_{t-1}^F(i)) \tag{13}$$

where Q is the queue.

We implement the cost-to-arrive V_t^F as a 2D matrix, where each position stores the latest optimal cost-to-arrive value from start s to its position j in time-step t.

Besides, in order to get the optimal path, we have another 2D matrix called parent, where each position stores the previous node's position that get to its position to yield the optimal cost-to-arrive value. We update the parent matrix whenever we update the cost-to-arrive V_t^F matrix. By tracking from the end node, we can get the path from the parent matrix in a reverse order.

B. Key Door problem

Since the optimal path might not includes passing the door, we divided the problem into 2 cases: (1) Go direct to goal, (2) Go get key and then to goal.

For case 1, we need to find:

- $C_{dir} = Cost(agent\_pos, goal\_pos)$

where $Cost(s, e)$ is the optimal cost of s to e.

For case 2, we need to find:

- $C_{key} = Cost(agent\_pos, key\_pos)$
- $C_{key2goal} = Cost(key\_pos, goal\_pos)$

After the agent gets the key, we can assume that all the doors are unlocked, since the agent is able to take the unlock action when it meets the door.

We then check the following condition to determine which path should we take.

$$C_{dir} > C_{key} + C_{key2goal} \tag{14}$$

If the condition is true, we know that we should get the key first, else there's a better path without getting a key.

C. Retrieve control input

Once we select a better score from the condition above, and get the path from its parent matrix, we need to transform the path into a series of control input. Now, we consider the initial agent orientation and the proper action it should take to get to the next position in the given path.

IV. Results

The following are the optimal control input for each environment:

For env 5x5-normal, the optimal control sequence is: ['TR', 'TR', 'PK', 'TR', 'UD', 'MF', 'MF', 'TR']

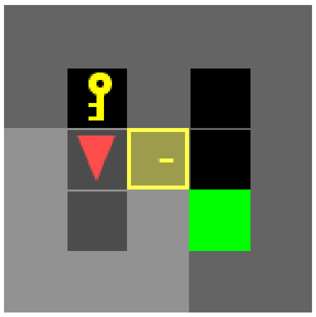

Fig. 1. 5x5-normal

For env 6x6-direct, the optimal control sequence is: ['MF', 'MF', 'TR', 'MF']

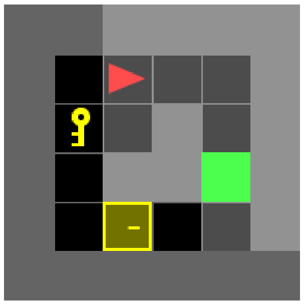

Fig. 2. 6x6-direct

For env 6x6-normal, the optimal control sequence is: ['TL', 'MF', 'PK', 'TL', 'MF', 'TL', 'MF', 'TR', 'UD', 'MF', 'MF', 'TR']

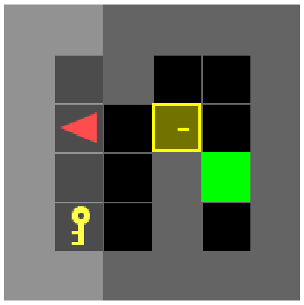

Fig. 3. 6x6-normal

For env 6x6-shortcut, the optimal control sequence is: ['PK', 'TR', 'TR', 'UD', 'MF']

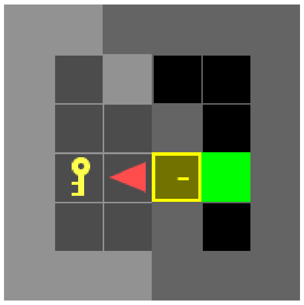

Fig. 4. 6x6-shortcut

For env 8x8-direct, the optimal control sequence is: ['MF', 'TL', 'MF', 'MF', 'MF', 'TL']

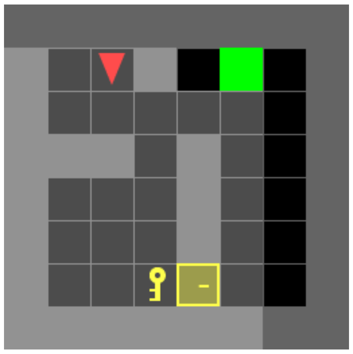

Fig. 5. 8x8-direct

For env 8x8-normal, the optimal control sequence is: ['TR', 'MF', 'TL', 'MF', 'TR', 'MF', 'MF', 'MF', 'PK', 'TR', 'TR', 'MF', 'MF', 'MF', 'TR', 'UD', 'MF', 'MF', 'MF', 'TR', 'MF', 'MF']

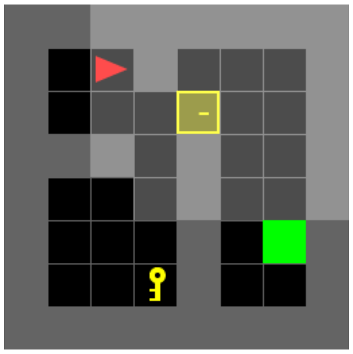

Fig. 6. 8x8-normal

For env 8x8-shortcut, the optimal control sequence is: ['TR', 'MF', 'TR', 'PK', 'TL', 'UD', 'MF']

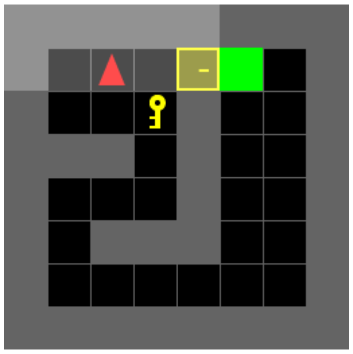

Fig. 7. 8x8-shortcut

From the results above, we can verify the control sequence by hand that these are the optimal control inputs.