ECE276B Project 1: Dynamic Programming

Awies Mohammad Mulla
MS ECE
ISRC
A59016119
amulla@ucsd.edu

***Abstract*—**

I. Introduction

We are implementing a dynamic programming algorithm to find the optimal path for a robot to traverse a grid. The problem is formulated as a Markov Decision Process (MDP). The actions of the robot comprises of simple actions: forward step, turn right, turn left and special actions: picking-up the key and unlocking the door. Since, the status of the door could be both open and close, our optimal policy will differ accordingly (whether to pick up the key for opening the door or not) and it is this fact which make this problem more challenging compared to solving a simple 2D grid.
The problem of Optimal Control is an essential area of research in robotics which is linked to every field of robotics. From autonomous cars in the form of motion planning to robotic arms in the form of trajectory planning, the problem of optimal control is ubiquitous. The current project is form of an optimal control (motion planning) problem on a much smaller scale. It can be explained as a robot trying to find the optimal path to reach a goal state from a start state. The robot has to avoid obstacles and depending on the state of the door robot has to pick up the key which is somewhat similar to traversing through a waypoint.
We are solving the mentioned problem using Forward Dynamic Programming. We have formulated the problem as a Markov Decision Process (MDP) in the form of non-directional graph in which the nodes consists of the state of the robot mentioned in the Problem Formulation and the edges representing the costs of traversing between nodes. The connectivity of the nodes is further described in the Formulation and the cost on edges here has been taken as equal for all edges. Using Dynamic Programming we are traversing through the graph to find the optimal path (to each of the nodes progressively).

II. Problem Formulation

The problem is formulated as a MDP i.e., the next state of the system is only dependent on the current state and the input provided at that timestep. The state of the system consists of pose of the agent (position and direction), status of the key with the agent (whether the agent has picked up the key or not) and the status of the door (whether the door is open or closed). The input to the system is the action taken by the agent at that timestep. The actions are simple actions: forward step, turn right, turn left and special actions: picking-up the key and unlocking the door. The elements of the MDP are defined below:

- **State Space**: The state space (χ) consists of all the possible states (x) of the system. The state of the system is defined as $x = [x_1, x_2, x_3, x_4, x_5, x_6]^T$. The elements of the state vector are defined as follows:

$$\begin{aligned} x_1 &= \text{X-coordinate of the agent in the grid} \\ x_2 &= \text{Y-coordinate of the agent in the grid} \\ x_3 &= \text{Direction of the agent in the grid along x-axis} \\ x_4 &= \text{Direction of the agent in the grid along y-axis} \\ x_5 &= \text{Status of the key with the agent (0: No, 1: Yes)} \\ x_6 &= \text{Status of the door (0: Closed, 1: Open)} \end{aligned}$$

  where $x_1, x_2 \in \mathbb{W}$; $\mathbb{W}$ is the set of all the possible coordinates in the grid and $x_3, x_4 \in \{-1, 0, 1\}$.
- **Control Space**: The control space ($\mathbb{U}$) consists of all the possible actions ($u \in$ Forward (MF), Turn Right (TR), Turn Left (TL), Pick Key (PK), Unlock Door (UD)) that the agent can take. We can defined the actions as follows (to suit our motion model):

$$\begin{aligned} u &= [1\ 0\ 0\ 0\ 0]\text{Forward Step} \\ u &= [1\ 0\ 0\ -1\ -1]\text{Turn Right} \\ u &= [0\ 0\ 0\ 1\ -1]\text{Turn Left} \\ u &= [0\ 1\ 0\ 0\ 0]\text{Pick-up Key} \\ u &= [0\ 0\ 1\ 0\ 0]\text{Unlock Door} \end{aligned}$$

- **Motion model**: The motion model (f) describes the dynamics of the system. The motion model is defined as follows:

$$x_{t+1} = x_t + Bu_t$$

  where x_t is the state of the system at time t and u_t is the action taken by the agent at time t. we get above motion model since we know no noise is present in the system. The matrix B is defined as follows:

$$B = \begin{bmatrix} x_3 & 0 & 0 & 0 & 0 & 0 \\ 0 & x_4 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & x_4 & x_3 \\ 0 & 0 & 0 & 0 & x_3 & x_4 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

- **Initial State**: The initial state of the system comprise of agent position and direction in the format mentioned above along with the status of the key and the door (both 0).
- **Planning Horizon**: The planning horizon (T) is the maximum number of timesteps for which the agent has to plan the optimal path. Here it would be simply be taken as total number of nodes in the graph, for instance in the case of 5x5 grid it would be (5x5x4x2=200; 2 is for status of key).
- **Cost Function**: The cost function (c) is the function which defines the cost of taking an action at a particular state. The cost function is defined as $c = 1$ for all the actions.

III. Technical Approach

The problem is solved using Forward Dynamic Programming. To obtain optimal action sequence we are first calculating the cost to reach each state (and goal state). The algorithm to calculate costs-to-go is described below:

- Initialize the cost to reach each state as ∞ except the start state which is 0.
- Initialize an open list (this will contain the state as well as action to be taken at that state) which contains the states to be explored.
- Now insert the start state with possible actions at that state (TR, TL, MF, PK, UD) into the open list.
- Loop till the open list is empty or planning horizon is reached.
- During each loop, pop the first element from the open list and calculate the cost to reach the state which we will obtain on applying the popped input to the system with popped state using following equation:

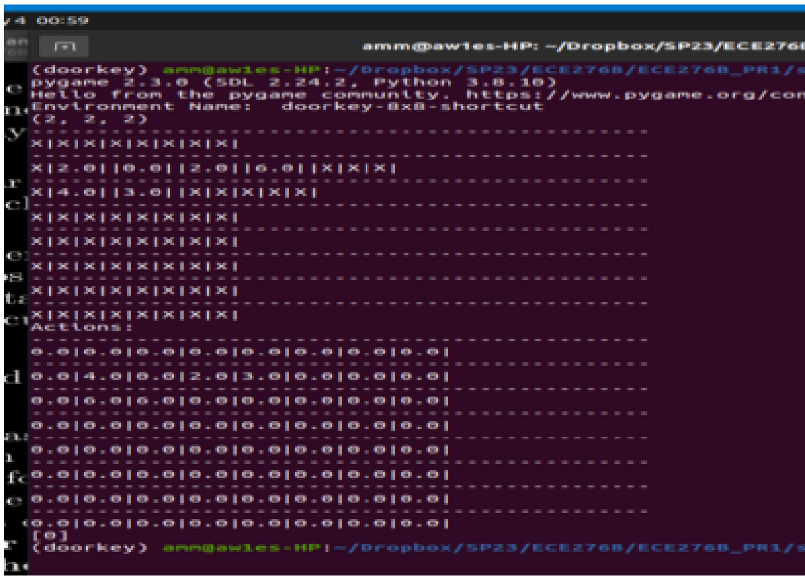
  $$V_{t+1}(x_{t+1}) = \min\{c(x_t, u_t) + V_t(x_t), V_t(x_t)\}$$

  where, $V_t(x_t)$ is the cost to reach the state x_t at time t and $V_{t+1}(x_{t+1})$ is the cost to reach the state x_{t+1} at time $t+1$. The cost function $c(x_t, u_t)$ is defined above.
- If the cost-to-go of a particular state is updated, then insert the state with possible actions at that state into the open list. If it is not updated i.e., the cost becomes stationary ($V_{t+1}(x_{t+1}) = V_t(x_t)$) then do not insert the state into the open list as this will be final cost-to-go for that state.
- We will loop till the open list is empty or planning horizon is reached.

After finding the cost-to-go for each state, we will find the optimal action sequence by backtracking from the goal state to the start state, using the cost table. This algorithm essentially requires the connectivity of the graph to be known beforehand (which was used to formulated the Motion model of the MDP).

IV. Results

A. Grids



```
4 00:59
amm@aw1es-HP: ~/Dropbox/SP23/ECE276
(doorkey) amm@aw1es-HP:~/Dropbox/SP23/ECE276B/ECE276B_PR1/s
pygame 2.3.0 (SDL 2.24.2, Python 3.8.10)
Hello from the pygame community. https://www.pygame.org/con
Environment Name:   doorkey-8x8-shortcut
(2, 2, 2)
-----------------------------------------------
x|x|x|x|x|x|x|x|
-----------------------------------------------
x|2.0||0.0||2.0||6.0||x|x|x|
-----------------------------------------------
x|4.0||3.0||x|x|x|x|x|
-----------------------------------------------
x|x|x|x|x|x|x|x|
-----------------------------------------------
x|x|x|x|x|x|x|x|
-----------------------------------------------
x|x|x|x|x|x|x|x|
-----------------------------------------------
x|x|x|x|x|x|x|x|
-----------------------------------------------
x|x|x|x|x|x|x|x|
Actions:
-----------------------------------------------
0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|
-----------------------------------------------
0.0|4.0|0.0|2.0|3.0|0.0|0.0|0.0|
-----------------------------------------------
0.0|6.0|6.0|0.0|0.0|0.0|0.0|0.0|
-----------------------------------------------
0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|
-----------------------------------------------
0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|
-----------------------------------------------
0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|
-----------------------------------------------
0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|
-----------------------------------------------
0.0|0.0|0.0|0.0|0.0|0.0|0.0|0.0|
[0]
(doorkey) amm@aw1es-HP:~/Dropbox/SP23/ECE276B/ECE276B_PR1/s
```

Fig. 1. 8x8 Grid

B. Analysis

- We were able to find the optimal cost-to-go for each state in the grid for all the grids in the known environment.
- Due to lack of time we were not able to implement the algorithm for the unknown environment.
- The results mentioned above are the images of the cost-to-go grids with minimum costs at each coordinate (among all directions).
- We could have been able to extract the optimal action sequence from the cost-to-go grid, but due to lack of time we were not able to implement it.
- The algorithm would be straightforward depending upon difference in the cost-to-go values and states.