Dynamic Programming

Nathan Cusson-Nadeau

I. Introduction

In robotics, it is often (and unsurprisingly) extremely desirable for an agent to be efficient at whatever task it is set out to do. Be this cleaning a floor, performing surgery, or pouring us a cup of coffee, having an optimal sequence of commands to perform the task at hand as efficient as possible is typically ideal. To construct this optimal series of commands, roboticists employ a variety of techniques which seek to find the best solution to a problem given a known environment. One such methodology is called dynamic programming, which seeks to achieve the optimal control sequences, or policy, that will be the most efficient, or in other words, minimize the most costs to the robot.

To employ a dynamic programming algorithm one must have knowledge of all states of the robot's operating environment. This includes it's starting position, the goal, and all relevant objects in the environment with knowledge of how to interact with them. If these prerequisites are met, the roboticist can then construct the algorithm by computing at all possible end states and working backwards in time to find which series of control inputs yields the minimum cost for the robot. Costs are defined by the roboticists and assigned to actions that the robot takes. For instance, if a robot is powered by a battery and is trying to minimize electricity usage while achieving its goal, a higher cost would be assigned to maneuvers which require more power. However, if the minimum time to complete a task is what is considered optimal, maneuvers which take longer to complete (even though they may save electricity) can be assigned higher costs.

In this project, we tackle something similar to what many of the aforementioned roboticists face. Using dynamic programming to construct an optimal control policy, our objective was to navigate an agent through various square grid environments with an optimal control policy which minimized cost.

II. Problem Formulation

Consider an autonomous agent at initial state x_0 within a known map environment M composed of $h \times h$ squares. Squares of M can be: occupied by the agent, walls, doors, a key, and a goal (see Figure 3). The agent's orientation within a square can be described by the cardinal directions: up, down, left, and right. Walls are stationary (cannot move from it's initial square), untraversable and uninteractable, meaning the agent cannot alter the walls or move onto a square occupied by them. The key is stationary and may be picked up by the agent and removed from the environment if the agent is in an adjacent square and is oriented towards the square where the key resides. The agent cannot occupy the same square as the key. Doors are stationary and can either be open or closed. When closed, the agent cannot occupy the door square. To open a door the agent must have obtained the key and, similarly to the key, can only open the door if it is in an adjacent square and is oriented towards the door. Finally, the goal is a stationary square that can be simultaneously occupied by the agent.

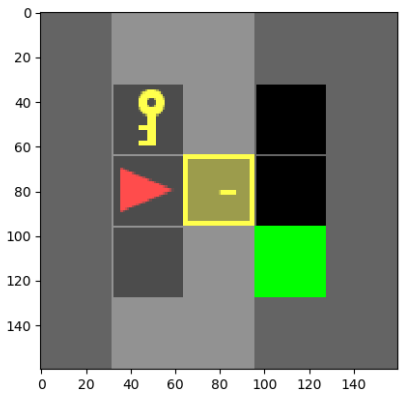

Fig. 1: Example Environment: Agent (Red Triangle), Door (Yellow Square), Goal (Green Square), Key (Yellow Key), Walls (Solid Gray Squares)

The agent's task at hand is to navigate to the goal square $G := [g_x, g_y]$ with the optimal control sequence π_0^* in the finite-time horizon T that minimizes energy consumption for the agent. As such, our goal in this project was to find the optimal control policy π_0^* which accomplishes this.

Additionally, the agent had to traverse two different types of environments with slightly different rules associated between the two. These two environments were tackled in two parts, A and B.

In part A, the agent must traverse seven different predetermined known environments and will need seven different control policies computed to navigate each associated policy environment. The only commonality between all seven environments is that there exists a key and a single closed door.

In part B, the environment will no longer be predetermined and will have elements of randomness as well as 2 doors instead of 1. This randomness occurs in 3 places: the goal location, doors state, and the key position. These random elements have a set of options which will be used when the map is randomly generated:

$$G := \{[5,1],[6,3],[5,6]\}$$

$$K := \{[1,1],[2,3],[1,6]\}$$

$$D_{1,2} := \{0,1\}$$

Here, G represents the set of possible goal coordinates. K similarly represents the set of possible key coordinates. D is the set of door states for door i, D_i, where i indexes door 1 or 2 respectfully. A door state of 0 for a D_i means that door is closed upon the agent spawning in, and vice versa for a door state of 1.

Upon spawning in M for B, the agent will be placed at square [3,5] facing up and receive full knowledge of the initial state x_0.

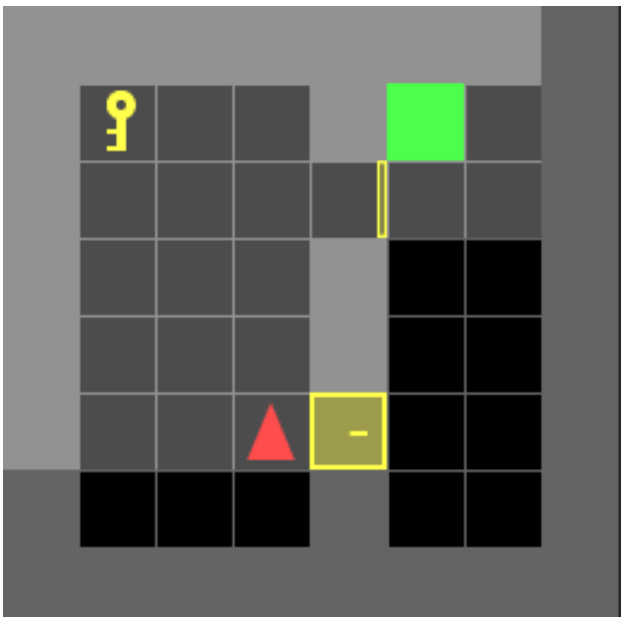

Fig. 2: Example Random Environment: Agent (Red Triangle), Door (Yellow Square), Goal (Green Square), Key (Yellow Key), Walls (Solid Gray Squares)

To tackle both types of problems, we first formulate these problem as a Markov Decision Processes (MDP). This requires the definition of the state-space $\mathcal{X}$, control space $\mathcal{U}$, motion model $f(x,u)$, time horizon T, stage cost $l(x,u)$, and terminal cost $q(x)$ to form the MDP tuple $(\mathcal{X},\mathcal{U},f,T,l,q)$. Part A and B will each require slightly different MDP tuple formulations that will be discussed in the following sections.

A. MDP Formulation Part A - Known Map

The state-space $\mathcal{X}$ of an environment from A can be described by $\mathcal{X} \in \mathbb{R}^{h\times h\times 4\times 2\times 2}$, where an individual state vector $\mathbf{x} \in \mathbb{R}^5$ represents:

$$\mathbf{x}_t := \begin{bmatrix} x_t^1 \\ x_t^2 \\ x_t^3 \\ x_t^4 \\ x_t^5 \end{bmatrix} = \begin{bmatrix} x \\ y \\ direction \\ doorstate \\ keystate \end{bmatrix} \tag{1}$$

$$x = \{0,1,...,h-1\}$$

Control Space		
Command	Representation	Result
MF	Move Forward	move one square in direction agent is facing
TL	Turn Left	rotate agent left 90°
TR	Turn Right	rotate agent right 90°
PK	Pick up Key	obtain key if in adjacent square and facing key
UD	Use Door	unlock door if holding key and in adjacent square facing closed door

TABLE I: Control Space $\mathcal{U}$

Costs	
MF	3
TL	3
TR	3
PK	1
UD	1

TABLE II: Costs of Actions

$$y = \{0,1,...,h-1\}$$

$$direction = \{0,1,2,3\}$$

$$doorstate = \{0,1\}$$

$$keystate = \{0,1\}$$

where x, and y are the current x and y coordinates of the agent respectfully, $direction$ is the orientation of the agent (up, down, left, or right), $door$ represents if the door is open or closed, and $keystate$ represents if the key has been obtained or not.

The agent can perform an action $u \in \mathcal{U}$ at each time step $t \in \{0,...,T\}$ which compose the control space

$$\mathcal{U} := \{MF, TL, TR, PK, UD\} \tag{2}$$

where the actions correspond to "Move Forward","Turn Left", "Turn Right", "Pick up Key", "Unlock Door" respectfully. These actions control the agent as follows:

Taking any of these actions costs energy for the agent and thereby incurs positive costs. The associated stage costs $l(x,u)$ are given by:

$$l(x,u) := \begin{cases} c_{x,u}, \text{valid action} \\ \infty, \text{invalid action} \end{cases} \tag{3}$$

where $c_{x,u}$ is the cost of using action u at the current state $\mathbf{x}$. These values are given in Table II.

The motion of the agent in M can be described by the motion model $f(x,u)$ defined as:

$$f(x,u) := \begin{cases} \text{valid action}, \mathbf{x}_{t+1}, \\ \text{invalid action or } [x_1,x_2] = G, \mathbf{x}_t \end{cases} \tag{4}$$

Invalid actions in part A can be described as any of the following scenarios involving x and u:

- any u while x, y correspond to coordinates in a locked door, on a key, or in a wall.
- $u = 3$ (attempt to pick up key) when the key has already been obtained ($keystate = 1$, x, y are not adjacent to the key's position on the cardinal directions, or $direction$ does not face the key.
- $u = 4$ (attempt to open door) when not holding the key ($keystate = 0$), or door is already open ($doorstate = 1$), or not adjacent and oriented correctly to open the door.

Finally, we define the terminal costs of the agent as:

$$q(x) := \begin{cases} 0, \text{ if } (x_1, x_2) = (g_x, g_y) \\ \infty, \text{ otherwise} \end{cases} \tag{5}$$

Defining invalid controls or final states in l and q to ∞ was done to ensure that actions that should not be possible incur the largest amount of cost. The reason for this will become apparent in the upcoming technical approach section.

B. MDP Formulation Part B - Random Map

The MDP of part B is defined near identically to part A's but with a singular key difference. This difference occurs in the state space $\mathcal{X}$ of B, which is now of the lengthy dimension $\mathbb{R}^{h\times h\times 4\times 2\times 2\times 3\times 3\times 2}$ which corresponds to a state vector $\mathbf{x} \in \mathbb{R}^8$:

$$\mathbf{x}_t := \begin{bmatrix} x_t^1 \\ x_t^2 \\ x_t^3 \\ x_t^4 \\ x_t^5 \\ x_t^6 \\ x_t^7 \\ x_t^8 \end{bmatrix} = \begin{bmatrix} x\_coord \\ y\_coord \\ direction \\ door1state \\ door2state \\ goallocation \\ keylocation \\ keystate \end{bmatrix} \tag{6}$$

where each of the elements correspond to the following sets:

$$x = \{0, 1, ..., 7\}$$
$$y = \{0, 1, ..., 7\}$$
$$direction = \{0, 1, 2, 3\}$$
$$door1state = \{0, 1\}$$
$$door2state = \{0, 1\}$$
$$goallocation = \{0, 1, 2\}$$
$$keylocation = \{0, 1, 2\}$$
$$keystate = \{0, 1\}$$

As evident in this state space representation there is now the inclusion of a goal location and key location state. These states allow for every permutation of the random environment to be encoded so that an optimal policy can be generated which takes all situations into account before-hand.

From these definitions, we arrive at our MDP formulation with tuple $(\mathcal{X}, \mathcal{U}, f, T, l, q)$ for both parts A and B.

III. Technical Approach

By formulating the problem as an MDP it is possible to apply a deterministic dynamic programming algorithm (DPA) which can compute the optimal control sequence π_0^* that minimizes the value function V_0^* over T. The deterministic DPA can be found in algorithm 1.

The algorithm operates by first computing the terminal value function $V_T(\mathbf{x}) = q(x)$ for all $\mathbf{x} \in \mathcal{X}$ at time T. This in essence assigns infinite costs to ending states $\mathbf{x}_T$ which do not share square coordinates with the associated goal G for its environment. We define T as an arbitrarily large time $T = 200$ which will allow the program to find the best policy regardless of the environment. This number was chosen after a bit of trial and error of executing the algorithm. All test cases exited the algorithm in less than 30 steps, so 200 is quite excessive - but this number is a very non-influential and just needs to be large "enough".

Next, we iterate backwards in time, assigning the cost function $Q_t(\mathbf{x}, \mathbf{u})$ 5 different values based on every possible control action u for each state x at every time step t. This is computed by adding the stage cost $l(x, u)$ with the prior value function $V_{t+1}(x')$. The prior value function corresponding to the state is determined by using the motion model $f(x, u) = x'$ by applying the control action.

Next the best value function for the current time and state $V_t(x)$ is computed by equating it to the minimum cost function with respect to control. This is because it would be considered the optimal control for this state and time if it results in the lowest cost. Using this optimal control u we then store it as a policy $\pi_t(x)$. This procedure continues until all states value functions are the same as the prior time step, which implies the best possible value functions have been computed and the optimal policy $\pi_{0:T-1}^*$ was obtained.

Algorithm 1
Deterministic Dynamic Programming

Input: MDP $(\mathcal{X}, \mathcal{U}, f, T, \ell, q)$
$V_T(\mathbf{x}) = q(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}$
for t = T-1 to 0 **do**
  $Q_t(\mathbf{x}, \mathbf{u}) = \ell(\mathbf{x}, \mathbf{u}) + V_{t+1}(\mathbf{x}'), \forall \mathbf{x} \in \mathcal{X}, \mathbf{u} \in \mathcal{U}$
  $V_t(\mathbf{x}) = \min_{u \in \mathcal{U}} Q_t(\mathbf{x}, \mathbf{u}), \forall \mathbf{x} \in \mathcal{X}$
  $\pi_t(x) = \arg\min_{u \in \mathcal{U}} Q_t(\mathbf{x}, \mathbf{u}), \forall \mathbf{x} \in \mathcal{X}$
  if $V_t(\mathbf{x}) = V_{t+1}(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}$ **then**
    break
  end if
end for
return policy $\pi_{0:T-1}$ and value function V_0

IV. Results

A. Results Part A - Known Map

After applying algorithm 1 to each of the seven environments, the optimal policy sequence and associated value function sequence was generated, then tested by applying

5x5 Normal Optimal Policy								
Policy:	TL	PK	TR	UD	MF	MF	TR	MF
Value:	20	17	16	13	12	9	6	3
Time:	187							

TABLE III: 5x5 Normal Environment Policy

6x6 Direct Optimal Policy				
Policy:	TL	TL	MF	MF
Value:	12	9	6	3
Time:	188			

TABLE IV: 6x6 Direct Environment Policy

the generated policy to the agent in real-time. Additionally, the time horizon in which all value functions converged was determined. The following tables present data for a few of the environments tested. For brevity, not all environment sequences are shown here, but they can be found in the "Optimal Policies.txt" file in the associated code submission package. .gif files can also be found in the associated code submission folder which illustrate visually how the agent maneuvered in accordance to these policies in the given environment when tested.

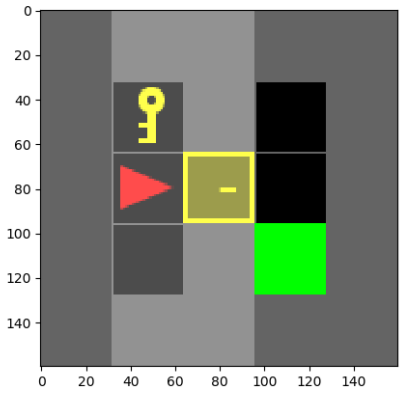

Fig. 3: 5x5 Normal Environment

Assessing these results, we can see that with the associated stage cost $l(x, u)$ and terminal cost $q(x)$, these action sequences are optimal. The value function is reduced to 0 when in the goal, as expected with a terminal cost of 0.

Further experimentation could be done by playing around with the stage costs that could potentially alter the path taken by the agent. For instance, drastically increasing the cost of a MF command without a key and having a minuscule cost

8x8 Shortcut Optimal Policy										
Policy:	MF	TR	PK	TR	MF	TR	MF	UD	MF	MF
Value:	26	23	20	19	16	13	10	7	6	3
Time:	183									

TABLE V: 8x8 Shortcut Environment Policy

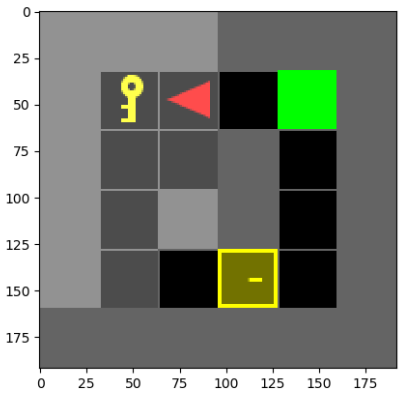

Fig. 4: 6x6 Direct Environment

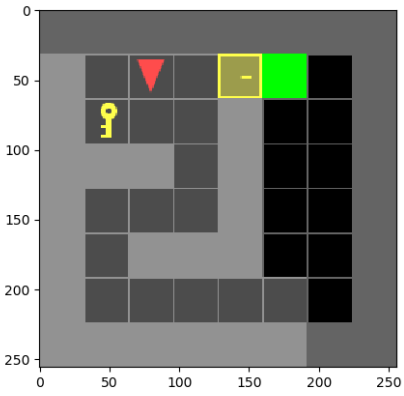

Fig. 5: 8x8 Shortcut Environment

of the MF command when holding a key could result in an interesting situation where the agent would intentionally pursue the key even if the goal is already reachable. With how the stage cost are assigned now, the optimal path will always be the one which requires as little movement as possible, an expected result and often times the desired one. But as mentioned above, this does not always have to be the case and the dynamic programming algorithm should still perform well in this case.

B. Results Part B - Random Map

Using the dynamic programming algorithm to compute a single policy for the random environment proved successful. Multiple test across many permutations of the random map yielded the same time of convergence and a singular optimal policy $\pi^*_{0:T}(x_0)$ that could be used to compute the ideal sequence of actions as a function of the initial state x_0. In the following tables, the optimal policies and value functions are provided, along with their time of convergence ($t = 178$).

Animated .gif files of successful runs can be found in the affiliated code submission for this report. Also, found within

Random Map - 20													
Policy:	MF	MF	TL	PK	TR	MF	TR	UD	MF	MF	MF	TR	MF
Value:	35	32	29	26	25	22	19	16	15	12	9	6	3
Time:	178												

TABLE VI: Random Environment Policy 20

Random Map - 31					
Policy:	TR	MF	MF	TR	MF
Value:	15	12	9	6	3
Time:	178				

TABLE VII: Random Environment Policy 31

is a failed run from the debugging process which provided an interesting result that illustrated the importance of associating terminal costs with the right state. In the bugged run, the terminal costs for the goal positions were set to 0 for all permutations of the states - even the state vectors were not associated with the goal. Because of this, the algorithm routed the agent to the fastest perceived goal every time, even if it were not the true goal.

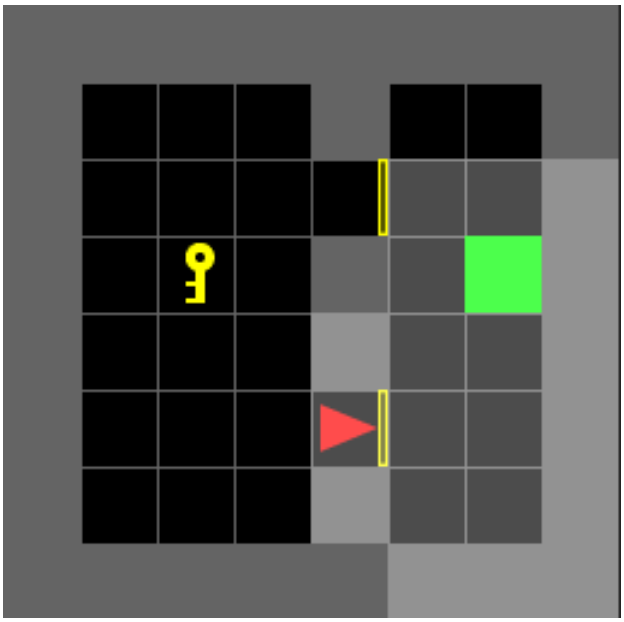

Fig. 6: Random Environment Example