Dynamic Programming

Renu Krishna Gutta, PID: A59018210, ECE 276B, Project-1, 05/01/2023

1. INTRODUCTION

The goal of this project is to find the optimal control policy that would take the robot to a goal. We consider the robot is in a 2D grid environment which has walls, doors and a key for unlocking the doors. The aim of the robot is to traverse the environment in the least cost way to reach the goal. This has many real-life applications such as autonomous vehicles and other path finding and obstacle avoidance problems. To find the optimal control policy, we will first formulate this as a Markov Decision Process (MDP). In an MDP, a robot interacts with an environment over a series of discrete time steps, making decisions at each time step that affect the state of the robot and spends some cost in return. The goal is to find a policy that minimizes the expected total cost over time. The duration is finite here and hence we call this problem as the Finite-horizon Optimal Control. In this project, we will try to solve this problem using Dynamic Programming.

2. PROBLEM FORMULATION

2.1. State-Space (χ)

Define the state with following attributes:

State χ_A for Part-A

Attribute	Value range
Robot position (pos)	$(i,j) \in \mathbb{Z}^2, 0 \le i,j < grid\ size(m \times n)$
Orientation (ori)	$(m,n) \in \{UP: (0,-1), LEFT: (-1,0), DOWN: (0,1), RIGHT: (1,0)\}$
Having key $(keystatus)$	$\{No: 0, Yes: 1\}$
Door open $(doorstatus)$	$\{closed: 0, open: 1\}$

State χ_B for Part-B

Attribute	Value range
Robot position (pos)	$(i,j) \in \mathbb{Z}^2, 0 \le i,j < grid\ size(m \times n)$
Orientation (ori)	$(m,n) \in \{UP: (0,-1), LEFT: (-1,0), DOWN: (0,1), RIGHT: (1,0)\}$
Key positions $(keypos)$	$(i_1,i_2) \in \{(1,1),(2,3),(1,6)\}$
Having key $(keystatus)$	$\{No: 0, Yes: 1\}$
Two doors status$(doorstatus)$	Closed-Closed: 0 Closed-Open: 1 Open-Closed: 2 Open-Open: 3

2.2. Control-Space ($\mathcal{U}$)

Set of actions:

Move forward	MF
Turn left	TL
Turn right	TR
Pick key	PK
Unlock Door	UD

2.3. Motion Model (f)

$$x \in \chi, u \in \mathcal{U}$$
$$x_{t+1} = f(x_t, u_t)$$

Function $f(.)$ will be described in the technical approach section.

2.4. Initial state (x_0)

The initial values of all the state attributes, which are available in the provided environment grid data.

2.5. Planning horizon (T)

$$T = number\ of\ states - 1$$

For part A:

$$number\ of\ states = m \times n \times 4 \times 2 \times 2$$

For part B:

$$number\ of\ states = m \times n \times 4 \times 3 \times 2 \times 4$$

2.6. Stage (l) and Terminal costs (q)

$$l(x_t,u_t) = \begin{cases} 1, & if\ action\ u_t\ is\ allowed \\ \infty, & if\ action\ u_t\ is\ not\ allowed \end{cases}$$

$$q(x_T) = \begin{cases} 1, & if\ x_t\ has\ pos = goal \\ \infty, & otherwise \end{cases}$$

2.7. *Markov Decision Process: Finite-horizon Optimal Control*

The finite-horizon optimal control problem in an MDP $(\chi, \mathcal{U}, \mathrm{p}(x_0), f(.), T, l, q)$ with initial state x_t at time t

Part A:
To find optimal policy $\pi_{t:T-1}$;

$$\min_{\pi_{t:T-1}} V_t^{\pi}(x_t) \coloneqq q(x_T) + \sum_{\tau=t}^{T-1} l\big(x_\tau, \pi_\tau(x_\tau)\big)$$
$$such\ that,\ \ x_{\tau+1} = f(x_\tau, u_\tau), \qquad \tau = t, \dots, T-1$$
$$x_\tau \in \chi, \qquad \pi_\tau(x_\tau) \in \mathcal{U}$$

Part B:
Since there are three possible goal (terminal) locations, let them be $\left[x_T^{(1)}, x_T^{(2)}, x_T^{(3)}\right]$
Optimal policy: $\pi_{t:T-1} \coloneqq \left[\pi_{t:T-1}^{(1)}, \pi_{t:T-1}^{(2)}, \pi_{t:T-1}^{(3)}\right]$

For $k = 1, 2, 3$:

$$\min_{\pi_{t:T-1}^{(k)}} V_t^{\pi}(x_t) \coloneqq q\big(x_T^{(k)}\big) + \sum_{\tau=t}^{T-1} l\left(x_\tau, \pi_\tau^{(k)}(x_\tau)\right)$$

3. TECHNICAL APPROACH

3.1. *Part A*

The environment grids are available as .env files. Once loaded we get a *gym-environment* data structure which contains all the details of the grid. Let's denote the data structure as ***env***

3.1.1. Initialize states:

- From the *env* structure, we get the grid size.
- Initialize a 2D list of grid size: ***state_grid***
- Each cell of the *state_grid*, contains a list of 16 states.
  - Explanation: Each cell in the *state_grid* corresponds to the spatial location in the *env* grid.
  - Each spatial location will have 4 x 2 x 2 number of states – owing to the possibilities of 4 orientations, have-key/do-not-have-key and door-locked/unlocked.
  - These 16 states for one spatial location and stored as a list in that location in state*_grid*
- All the states have their *pos, ori, keystatus, doorstatus* in a sequential manner, so that when given just these attribute values, we can calculate the index of that particular state in the *state_grid*.

3.1.2. Define motion model:

- **Goal:** To write the function, $x_{t+1} = f(x_t, u_t)$
- Instead of generating a new state x_{t+1}, this function will actually point to the pre-generated states in the *state_grid*.
- For each state, we compute its front position: $frontpos = pos + ori$
- If $u_t == MF$:
  - If $frontpos$ is neither a wall nor locked door: return state such that $x.pos_{t+1} = frontpos$ and rest all attributes are same.
    Else: return *None*.
- If $u_t == TL$:
  - Return the state with orientation turned left from current orientation and rest all attributes same as the current state.
- If $u_t == TR$:
  - Return the state with orientation turned right from current orientation and rest all attributes same as the current state.
- If $u_t == PK$:
  - If $frontpos$ has a key, $keystatus = 0$ (not having key) and the $doorstatus$ is 0 (closed), pick the key, i.e., return the state with $keystatus = 1$ with other attributes equal.
  - Else: return *None*
- If $u_t == UD$:
  - If $frontpos$ has a door, $keystatus = 1$ (having key) and the $doorstatus$ is 0 (closed), unlock the door, i.e., return the state with $doorstatus = 1$ with other attributes equal.
  - Else: return *None*

3.1.3. Compute stage, terminal costs, and initialize value function:

Stage cost:
- **Objective:** To compute $l(x, u) \quad \forall\, x \in \chi, u \in \mathcal{U}$ and update the state attribute, $stage\_cost$, which is a list of 5 elements corresponding to the each of the $u \in \mathcal{U}$
- For each state, we first compute its front position: $frontpos = pos + ori$
- If $u == MF$:
  - If $frontpos$ is neither a wall nor locked door: append $stage\_cost = 1$
    Else: append $stage\_cost = \infty$
- If $u == TL$:
  - append $stage\_cost = 1$
- If $u == TR$:
  - append $stage\_cost = 1$
- If $u == PK$:
  - If $frontpos$ has a key, $keystatus = 0$ (not having key) and the $doorstatus$ is 0 (closed): append $stage\_cost = 1$
  - Else: append $stage\_cost = \infty$
- If $u == UD$:
  - If $frontpos$ has a door, $keystatus = 1$ (having key) and the $doorstatus$ is 0 (closed): append $stage\_cost = 1$
  - Else: append $stage\_cost = \infty$

Terminal cost:
- **Objective:** To compute $q(x) \quad \forall\, x \in \chi$ and update the state attribute, $terminal\_cost$, which is a single value.
- Get goal position $goal\_pos$ from *env* data structure.
- For each $x \in \chi$,

If $x.pos == goal\_pos$: set $terminal\_cost = 1$
Else: set $terminal\_cost = \infty$

- Note that for one $goal\_pos$, there will be 16 possible states. Reaching any of these states will satisfy the main problem.

Value function:

- **Objective:** To initialize value function for each state which will be iteratively updated while applying the Dynamic Programming algorithm.
- After we finish running the algorithm, this value function of each state will give the minimum cost of reaching the goal starting from that state.
- For each $x \in \chi$,

If $x.pos == goal\_pos$: initialize $value\_function = 1$
Else: initialize $value\_function = \infty$

3.1.4. Update children and parents

- **Objective:** Using motion model, we find the children and parents of each state and their corresponding actions.
- We define the attributes $parents, children$ in the state variable.
- For each state, run motion model. The output state will be a *child* of the input state and the input state will be a *parent* of the output state.
  i.e., let $x_{t+1} = f(x_t, u_t)$. Then:
  $$x_t.children.append(x_{t+1}, u_t)$$
  $$x_{t+1}.parents.append(x_t, u_t)$$

3.1.5. Optimal control policy: Dynamic Programming

- **Goal:** To obtain optimal path for the robot to reach the goal with minimum cost.
- Ater performing all the previous steps, we have the updated ***state_grid*** with all the attributes required to perform Dynamic Programming.
- We are performing backward dynamic programming.
- For each state, we store the optimal action that needs to taken at that state to reach the goal

```
# Algorithm
goal_states ← states with pos=goal_pos
Initialize: current_children ← goal_states
            next_children ← []

# Backward dynamic programming
FOR t in T-1:0
  FOR child in current_children
    FOR (parent,act) in child.parents
      Q = parent.stage_cost + child.value_func
      IF Q < parent.value_func
        parent.value_func = Q
        parent.optimal_action = act
        next_children.append(parent)
      ENDIF
    ENDFOR
  ENDFOR
  current_children ← next_children
  next_children ← []
ENDFOR

# Get optimal control sequence
optimal_seq ← []
start_state ← get_initial_state(env)
X ← start_state
WHILE X is not goal
  optimal_seq.append(X.optimal_action)
  X ← motion_model(X, X.optimal_action)
END

OUTPUT: optimal_seq
```

3.2. Part B

In Part B, the state space changes as described in sec.2.1. Since the key can be in any of the three positions {(1,1), (2,3), (1,6)}, we introduce the key position as an attribute in the state.
Also, the $goal\_pos \in \{(5,1),(6,3),(5,6)\}$. So, **the optimal control policy must generate action sequences for each of these goal positions.**

3.2.1. Initialize states

- There are 36 different random environments all having the same size, wall and doors' locations.
- That is, grid size: 8x8, vertical wall at column 4 with two doors at (4,2) and (4,5)
- Just like in Part A, we define a 2D ***state_grid*** of size 8x8.
- Here each element will have 96 states all with the same spatial position.
  - Explanation: For each location, there are possibilities of 4 orientations, 3 key positions, have-key/do-not-have-key and 4 {open,close}x{open,close} states of the doors.
  - Hence, 4 x 3 x 2 x 4 = 96.
- Therefore, the attributes are - *pos, ori, keypos, keystatus, doorstatus.*

3.2.2. Define motion model:

- The core logic is similar to Part A with slight modifications as per the new extended state space.
- For each state, we compute its front position: $frontpos = pos + ori$
- If $u_t == MF$:
  - If $frontpos$ is neither a wall nor locked door: return state such that $x.pos_{t+1} = frontpos$ and rest all attributes are same.
    Else: return *None*.
- If $u_t == TL$:
  - Return the state with orientation turned left from current orientation and rest all attributes same as the current state.
- If $u_t == TR$:
  - Return the state with orientation turned right from current orientation and rest all attributes same as the current state.
- If $u_t == PK$:
  - If $frontpos = keypos$ (key position), $keystatus = 0$ (not having key) and the $doorstatus \neq (open, open)$, pick the key,

        - i.e., return the state with $keystatus = 1$ with other attributes equal.
        - Else: return $None$
- If $u_t == UD$:
    - If $frontpos$ is one of the door locations i, $keystatus = 1$ (having key) and the $doorstatus[i] = closed$ for that particular door: unlock that door, i.e., return the state with $doorstatus[i] = open$ with other attributes equal.
    - Else: return $None$

3.2.3. *Compute stage, terminal costs and initialize value function*

Stage cost:

- For each state, we first compute its front position: $frontpos = pos + ori$
- If $u == MF$:
    - If $frontpos$ is neither a wall nor locked door: append $stage\_cost = 1$
      Else: append $stage\_cost = \infty$
- If $u == TL$:
    - append $stage\_cost = 1$
- If $u == TR$:
    - append $stage\_cost = 1$
- If $u == PK$:
    - If $frontpos = keypos$ (key position), $keystatus = 0$ (not having key) and the $doorstatus \neq (open, open)$: append $stage\_cost = 1$
    - Else: append $stage\_cost = \infty$
- If $u == UD$:
    - If $frontpos$ is one of the door locations i, $keystatus = 1$ (having key) and the $doorstatus[i] = closed$ for that particular door: append $stage\_cost = 1$
    - Else: append $stage\_cost = \infty$

Terminal cost:

- Given the goal position $goal\_pos$.
- For each $x \in \chi$,
  If $x.pos == goal\_pos$: set $terminal\_cost = 1$
  Else: set $terminal\_cost = \infty$
- Note that for one $goal\_pos$, there will be 96 possible states. Reaching any of these states will satisfy the main problem.

Value function:

- After we finish running the algorithm, this value function of each state will give the minimum cost of reaching the goal starting from that state.
- For each $x \in \chi$,
  If $x.pos == goal\_pos$: initialize $value\_function = 1$
  Else: initialize $value\_function = \infty$

3.2.4. *Optimal Control Policy: Dynamic Programming*

- **Objective:** To provide a single optimal control policy.
- There are three possible goal positions and we do not know the exact goal position at the time of control policy generation.
- Hence, for each state we must find three different optimal actions so that each of those actions will take the robot from that state to the corresponding goal in minimum cost.
- Hence, we treat it as three independent backward dynamic programming problems.
- Refer to sec.3.1.5 for the dynamic programming algorithm.
- Therefore, for each state, we will have three different optimal actions. Each action will take us to each of the three goals.
- <u>Note about code implementation:</u>
  Dynamic programming is supposed to be implemented only once where all the three optimal actions are found in parallel for each state. For better code readability and error avoidance, I implemented the algorithm in a sequential manner, where I first find optimal policy with respect to $goal\_1$ for all the states, followed by $goal\_2$ and $goal\_3$

- <u>Obtain the optimal action:</u>
    - Load the random environment.
    - Get the $goal\_pos$ from the environment.
    - Find the $initial\_state$ by reading all the required attributes from the environment.
    - Pass the $goal\_pos$ and $initial\_state$ into the optimal control policy to get the optimal action sequence for the particular environment.

4. RESULTS

Note: To view the animation .gif files, please ensure that the directories – *'./known_opt_gifs'* and *'./random_opt_gifs'* are in the same directory as this report.

4.1. Part A

(Click on the hyperlinks below to view the animation)

Environment (.gif)	**Optimal Path**
doorkey-5x5-normal	TR, TR, PK, TR, UD, MF, MF, TR, MF
doorkey-6x6-direct	MF, MF, TR, MF, MF
doorkey-6x6-normal	TL, MF, PK, TR, TR, MF, TR, MF, UD, MF, MF, TR, MF
doorkey-6x6-shortcut	PK, TR, TR, UD, MF, MF
doorkey-8x8-direct	MF, TL, MF, MF, MF, TL, MF
doorkey-8x8-normal	TR, MF, TL, MF, TR, MF, MF, MF, PK, TR, TR, MF, MF, MF, TR, UD, MF, MF, MF, TR, MF, MF, MF
doorkey-8x8-shortcut	TR, MF, TR, PK, TL, UD, MF, MF

4.2. Part B

(Click on the hyperlinks below to view the animation)

Environment (.gif)	Optimal Path
DoorKey-8x8-1	MF, MF, MF, TR, MF, MF, TL, MF
DoorKey-8x8-2	MF, MF, MF, TR, MF, MF, TL, MF
DoorKey-8x8-3	TR, MF, MF, TL, MF, MF, MF, MF
DoorKey-8x8-4	MF, MF, MF, TL, MF, MF, TR, PK, TR, MF, MF, UD, MF, MF, TL, MF
DoorKey-8x8-5	TR, MF, MF, MF, TL, MF, MF
DoorKey-8x8-6	
DoorKey-8x8-7	TR, MF, MF, MF, TL, MF, MF
DoorKey-8x8-8	MF, MF, MF, TL, MF, MF, TR, PK, TR, MF, MF, UD, MF, MF, MF, TR, MF
DoorKey-8x8-9	TR, MF, MF, TR, MF
DoorKey-8x8-10	MF, MF, MF, TR, MF, MF, TR, MF, MF, MF, MF
DoorKey-8x8-11	TR, MF, MF, TR, MF
DoorKey-8x8-12	MF, MF, MF, MF, TL, MF, PK, TL, MF, MF, MF, MF, TL, MF, UD, MF, MF, TR, MF
DoorKey-8x8-13	MF, MF, MF, TR, MF, MF, TL, MF
DoorKey-8x8-14	MF, MF, MF, TR, MF, MF, TL, MF
DoorKey-8x8-15	TR, MF, MF, TL, MF, MF, MF, MF
DoorKey-8x8-16	MF, MF, TL, PK, TR, MF, TR, UD, MF, MF, TL, MF
DoorKey-8x8-17	TR, MF, MF, MF, TL, MF, MF
DoorKey-8x8-18	MF, MF, MF, TR, MF, MF, MF, TR, MF
DoorKey-8x8-19	TR, MF, MF, MF, TL, MF, MF
DoorKey-8x8-20	MF, MF, TL, PK, TR, MF, TR, UD, MF, MF, MF, TR, MF
DoorKey-8x8-21	TR, MF, MF, TR, MF
DoorKey-8x8-22	MF, MF, MF, TR, MF, MF, TR, MF, MF, MF, MF
DoorKey-8x8-23	TR, MF, MF, TR, MF
DoorKey-8x8-24	MF, MF, TL, PK, TL, MF, MF, TL, UD, MF, MF, TR, MF
DoorKey-8x8-25	MF, MF, MF, TR, MF, MF, TL, MF
DoorKey-8x8-26	MF, MF, MF, TR, MF, MF, TL, MF
DoorKey-8x8-27	TR, MF, MF, TL, MF, MF, MF, MF
DoorKey-8x8-28	TL, MF, MF, TL, PK, TL, MF, MF, UD, MF, MF, TL, MF, MF, MF, MF
DoorKey-8x8-29	TR, MF, MF, MF, TL, MF, MF
DoorKey-8x8-30	MF, MF, MF, TR, MF, MF, MF, TR, MF
DoorKey-8x8-31	TR, MF, MF, MF, TL, MF, MF
DoorKey-8x8-32	TL, MF, MF, TL, PK, TL, MF, MF, UD, MF, MF, MF, TL, MF, MF
DoorKey-8x8-33	TR, MF, MF, TR, MF
DoorKey-8x8-34	MF, MF, MF, TR, MF, MF, TR, MF, MF, MF, MF
DoorKey-8x8-35	TR, MF, MF, TR, MF
DoorKey-8x8-36	TL, MF, MF, TL, PK, TL, MF, MF, UD, MF, MF, TR, MF

4.3. Discussion

- This problem requires backward dynamic programming because we are interested in knowing the shortest distance from every state in the state-space to the goal state.
- We use forward dynamic programming when we want to find shortest distance from the start state to every other state in the state-space.
- The conventional dynamic programming algorithm (DPA) involves computing the Q-function of every state *x* and for every control *u*. But here we store the children and parents for each state along with the corresponding actions. So out search space for finding the minimum cost action reduces a lot, thereby speeding up the algorithm.
- For part B, since the goal position can be one of the three possibilities, we must maintain, three different terminal costs and value functions. For each state, each of its value functions is updated independently. Since the start point of backward-DPA, which is the goal, is different in case, we must run the algorithm thrice separately.
- Once we generate the optimal control policy for part B, we will pass the initial state of the environment as well as the goal position to determine the optimal action sequence from that policy.
- **Alternate approach for Part B:** The multiple goal positions case in part B could have been handled in a single DPA procedure by putting goal position and corresponding *terminal_cost* into the state attributes. So, when you choose *goal_1: terminal_cost={1,inf,inf}*, for *goal_2: terminal_cost={inf,1,inf}*, and for *goal_3: terminal_cost={inf,inf,1}*

ACKNOWLEDGEMENT

I would like to thank Prof. Nikolay Atanasov and TA – Zhirui Dai for guiding me with this project.

REFERENCES

[1] https://natanaso.github.io/ece276b/
[2] https://www.gymlibrary.dev/

APPENDIX

Few examples of .gif animations are provided in the following pages.

Part A: Known environment - examples

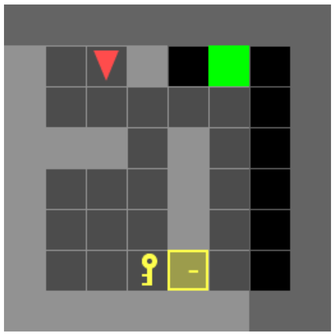

Figure 1: doorkey-8x8-direct

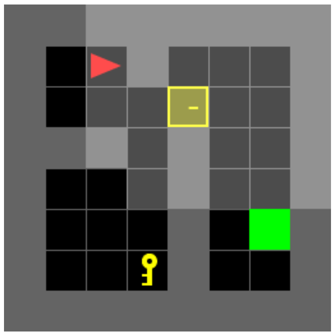

Figure 2: doorkey-8x8-normal

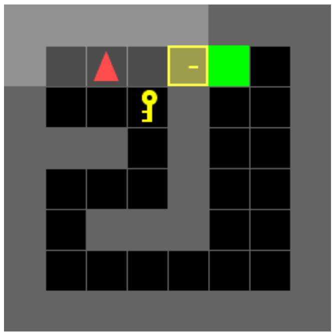

Figure 3: doorkey-8x8-shortcut

Part B: Random environment - examples

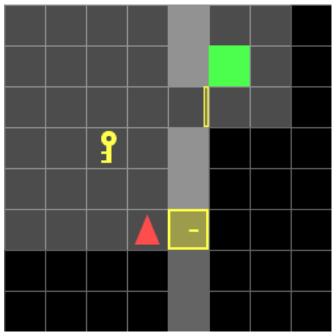

Figure 4: DoorKey-8x8-14

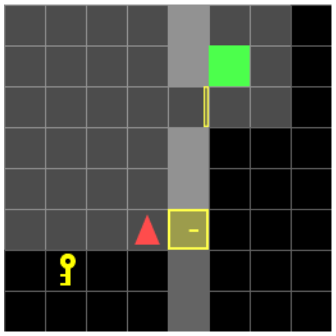

Figure 5: DoorKey-8x8-26

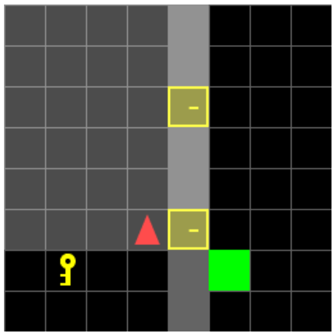

Figure 6: DoorKey-8x8-36