Dynamic Programming for Optimal Control

Jay Paek
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, California
jpaek@ucsd.edu

***Abstract*—This project addresses the challenge of autonomous navigation in a Door and Key environment using Dynamic Programming (DP). The objective is to guide an agent to a goal location while overcoming obstacles such as closed doors requiring keys for access. Two scenarios are explored: ”Known Map,” where specific environments are provided, and ”Random Map,” where environments vary randomly. In the ”Known Map,” individual control policies are computed for each environment, while a single policy adaptable to any of the 36 random 8x8 environments is devised for the ”Random Map.” The report presents a comprehensive overview of the problem formulation as a Markov Decision Process (MDP) with well-defined state and control spaces, motion models, initial states, planning horizon, and cost functions. The technical approach entails implementing a DP algorithm in Python to derive optimal control policies. Results showcase visualizations of agent trajectories under varying starting positions and orientations, assessing the algorithm's efficacy and limitations. This study contributes to understanding autonomous navigation challenges in complex environments, with potential applications in robotics and real-world scenarios.**

***Index Terms*—Robotics, dynamic programming, optimal control, searching, path-finding**

I. Introduction

Autonomous navigation poses a significant challenge in robotics, especially in dynamic environments with obstacles and complex spatial configurations. The Door & Key problem encapsulates one such scenario, where an agent must navigate through a maze-like environment to reach a goal while encountering doors that may block its path. The presence of locked doors necessitates the acquisition of keys, introducing a layer of decision-making and resource management. This problem is not only a theoretical exercise but also mirrors real-world applications in robotics, such as automated delivery systems, exploration missions in unknown territories, or even household assistance robots maneuvering through cluttered spaces.

Addressing the Door & Key problem requires a systematic approach grounded in principles of decision-making under uncertainty. Formulating the problem as a Markov Decision Process (MDP) provides a robust framework for designing optimal control policies. Key elements of the MDP include defining the state space, control space, motion model, initial state, planning horizon, and cost functions. By clearly delineating these components, we establish a mathematical foundation upon which to devise strategies that minimize the agent's energy expenditure while efficiently navigating the environment. This project aims to develop and implement a Dynamic Programming algorithm to derive optimal control policies, offering insights into effective navigation strategies and their performance under varying environmental conditions.

Through this project, we endeavor to contribute to the field of planning and learning in robotics by tackling a practical navigation problem within a controlled yet realistic setting. By employing Dynamic Programming techniques, we seek to optimize the agent's decision-making process, enabling it to navigate the Door & Key environment with minimal energy consumption. The ensuing discussion will delve into the technical intricacies of our approach, highlighting both successes and challenges encountered during algorithm development and implementation. Ultimately, our aim is to provide a comprehensive understanding of autonomous navigation strategies, paving the way for more sophisticated robotic systems capable of operating effectively in diverse and dynamic environments.

II. Notation and Preliminaries

The “agent” will be the autonomous navigator that we will try to produce an optimal policy.

$\mathcal{X}$ denotes the state space, an ordered set of all possible states, $\mathbf{x}$, of the agent. The order is determined by the construction of $\mathcal{X}$ in a first-in-first-out basis. $\mathcal{U}$ denotes the control space, the set of all possible control inputs, $\mathbf{u}$, for the agent.

$f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is the motion model for the agent, which computes the new state with respect to the control input.

$\ell : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ is the cost function for a control input $\mathbf{u}$ at any state $\mathbf{x}$.

T is the time horizon of the agent to reach the goal state. $t = 0, \dots, T$ will denote the time steps. $\mathfrak{q} : \mathcal{X} \to \mathbb{R}$ is the terminal cost assign to each state at time T. $\pi_t : \mathcal{X} \to \mathcal{U}$ denotes the control policy at a certain time t that maps a state $\mathbf{x}$ to a control input $\mathbf{u}$. $V_t : \mathcal{X} \to \mathbb{R}$ is the value function at time t where it assigns the cumulative cost to be at some state $\mathbf{x}$ at some time t.

A Markov decision process, abbreviated as MDP, is a probabilistic model with a state space $\mathcal{X}$, control space $\mathcal{U}$, initial state distribution $p_0(.)$, motion model distribution $p_f(.|\mathbf{x}, \mathbf{u})$, time horizon T, loss function ℓ, terminal cost function $\mathfrak{q}$, and foresight constant γ. In short, it will be summarized as MDP($\mathcal{X}, \mathcal{U}, p_0, p_f, T, \ell, \mathfrak{q}, \gamma$).

III. Problem Formulation

Our goal is to compute an optimal control policy for the agent navigating through a grid-like environment of size

$d \times d$. The environment is sectioned into different cells with coordinates assigned as (x, y). The horizontal direction from left to right will be $x = 0, \ldots, 7$ and the vertical direction from top to bottom will be $y = 0, \ldots, 7$. For example, the top left cell is $(0, 0)$ and the bottom right cell is (d, d). Each cell can either hold a key, be a wall, door, goal, or movable space.

The agent is spawned in this environment with two initial properties: position and direction. The initial position will be denoted by the cell coordinates of a movable space, (x, y). The initial direction will be either up (0), right (1), down (2), or left (3), and it is clear that the rightward and downward direction are the positive x and y directions, respectively, from the environment set-up. In general, any state $\mathbf{x} \in \mathcal{X}$ will have the following format:

$$\mathbf{x} = \begin{bmatrix} x\ position \\ y\ position \\ direction \\ door/key\ state \\ additional\ states \end{bmatrix}$$

where the first three entries is the position and direction for the agent, respectively. The format may switch between a vector and a tuple, but the order of the elements is preserved. For door/key state will either be

- (0) if the key is not obtained
- (1) if the key is obtained
- (2) if the target door is opened

Other additional states will be explained in the next section. We will index the ith component via the notation $[\mathbf{x}]_i$

$\mathcal{G} \subset \mathcal{X}$ denotes the states that are consider as the goal states.

We have $\mathcal{U} = \{0, 1, 2, 3, 4\}$ for $\mathbf{u} \in \mathcal{U}$,

$$\mathbf{u} = \begin{cases} 0 & \text{move forward} \\ 1 & \text{turn countclockwise} \\ 2 & \text{turn clockwise} \\ 3 & \text{pick up key} \\ 4 & \text{unlock door} \end{cases}$$

In this problem, the agent interacts with the environment in a certain way:

- The agent can not move to a spot with a key, wall, or a locked door.
- The agent can pick up the key only when it is adjacent to and facing towards the key.
- After picking up the key, the cell with a key becomes a movable cell.
- The agent can only open the door when it is adjacent and facing towards a locked door.
- After unlocking the door, the cell with the door becomes a movable cell.

Ultimately, we want to design an algorithm that computes the optimal control policy for the agent to reach the goal. We want to develop an algorithm that computes the optimal control policy for two different scenarios:

- Known Environment: The entire map is known and the location of the key, door, goal, and agent, along with the state of the door are known. This information can be used to compute the optimal control policy.
- Pseudo-Random Environment: The general map structure is an 8×8, and the perimeter is surrounded by walls. There is a vertical wall at column 4 with two doors at (4, 2) and (4, 5). Each door can either be locked or unlocked. The key is randomly located in one of three positions $\{$(1, 1),(2, 3),(1, 6)$\}$ and the goal is randomly located in one of three positions $\{$(5, 1),(6, 3),(5, 6)$\}$. The agent is initially spawned at (3, 5) facing up.

In mathematical terms, we want solve the following finite horizon optimal control problem:

$$\begin{aligned} \min_{\pi_{t:T-1}} V_t^{\pi}(\mathbf{x}) :=& \mathbb{E}_{\mathbf{x}_{t+1:T}} \left[\mathfrak{q}(\mathbf{x}_T) + \sum_{\tau=t}^{T-1} \ell(\mathbf{x}_\tau, \pi_\tau(\mathbf{x}_\tau)) \middle| \mathbf{x}_t = \mathbf{x} \right] \\ \text{s.t.} \quad & \mathbf{x}_{\tau+1} \sim p_f(.|\mathbf{x}_\tau, \pi_\tau(\mathbf{x}_\tau)), \tau = t, \ldots, T-1 \\ & \mathbf{x}_\tau \in \mathcal{X}, \pi_\tau(\mathbf{x}_\tau) \in \mathcal{U} \end{aligned}$$

IV. Technical Approach

There are two different scenarios which the agent must develop an optimal control policy for. The differences in technical approach for each scenario will be described in-depth in different subsections. However, the general approach to each scenario is very similar.

A. General Approach

We will be using the dynamic programming (dp) algorithm in order to compute the optimal control policy $\pi_{0:T}$. In essence, the dp algorithm will begin at the goal states at time T and initialize them to be of minimal terminal cost, while initializing all other states with ∞. Then at each previous time step, it will consider each possible control input at every state. Some of the states at the previous time step will end up at the goal state, so it will reward this control input at this state by giving it a finite cost. At the time step, it will choose to proceed with the control input that minimizes the culumative value function.

Afterwards, this is done until the algorithm worked its way all the way back to $t = 0$. Although this is not guaranteed to converge to a single control policy in general, since our system is determinstic, we can assume it is well-behaved.

Normally for a navigation and planning problem, we are given a probabilistic initial state and motion model, but with the given determinstic behavior of the agent, we can eliminate any probabilistic uncertainty within the dp algorithm. Additionally, since the environment is static, there is no need to return $\pi_{1:T}$. The control policy at each time step should remain the same for all states since the environment is not dynamic.

With some consideration regarding the door/key environment, it is possible to design the terminal cost function:

$$\mathfrak{q}(\mathbf{x}) = \begin{cases} 0 & \text{if } \mathbf{x} \in \mathcal{G} \\ \infty & \text{else} \end{cases}$$

coming from the fact that we want to deincentivize the agent from: not being at the goal state at $t = T$.

Algorithm 1 Deterministic Dynamic Programming

```
Require: MDP($\mathcal{X}, \mathcal{U}, f, T, \ell, \mathfrak{q}$)
  $V_T(\mathbf{x}) = \mathfrak{q}(\mathbf{x}), \forall \mathbf{x} \in \mathcal{X}$
  for $t = (T-1), \dots, 0$ do
      $Q_t(\mathbf{x}, \mathbf{u}) = \ell(\mathbf{x}, \mathbf{u}) + V_{t+1}(f(\mathbf{x}, \mathbf{u})), \forall \mathbf{x} \in \mathcal{X}, \mathbf{u} \in \mathcal{U}$
      $V_t(\mathbf{x}) = \min_{\mathbf{u} \in \mathcal{U}} Q_t(\mathbf{x}, \mathbf{u}), \forall \mathbf{x} \in \mathcal{X}$
      $\pi_t(\mathbf{x}) = \arg\min_{\mathbf{u} \in \mathcal{U}} Q_t(\mathbf{x}, \mathbf{u}), \forall \mathbf{x} \in \mathcal{X}$
  end for
  return $\pi_0, V_0$
```

Let $\mathbf{x} \in \mathcal{X} \subset \mathbb{Z}^n$ and $\mathcal{E} : \mathcal{X} \to \mathbb{Z}$ be an encoding function. Each entry of $\mathbf{x}$ will be a single digit integer, so we can encode each state to an n digit number. Each encoding will be unique since the states are unique. Construct a dictionary that will hash the encoded state, $\mathcal{E}(\mathbf{x})$ and give it a value as the index of $\mathbf{x}$ in $\mathcal{X}$. We do this so that we can index states without searching the state space in $\mathcal{O}(n)$ and instead can index the dictionary with $\mathcal{O}(1)$.

B. Known Environment

For the known environment, obtain the door (d_x, d_y), key (k_x, k_y), and goal (g_x, g_y) locations as well as all of the movable cells $\mathcal{M}$. In all environments, the door is initialized to be locked. Let $\mathbf{x} \in \mathcal{X}$ be as follows:

$$\mathbf{x} = \begin{bmatrix} x\ position \\ y\ position \\ direction \\ door/key\ state \end{bmatrix}$$

Initialize the state space:

$$\begin{aligned} \mathcal{X} = &\big(\mathcal{M} \times \{0,1,2,3\} \times \{0,1,2\}\big) \\ &\cup \big(\{(k_x, k_y)\} \times \{0,1,2,3\} \times \{1,2\}\big) \\ &\cup \big(\{(d_x, d_y)\} \times \{0,1,2,3\} \times \{2\}\big) \end{aligned}$$

which are the states that capture the agent in a movable cell facing any direction with any key possession state, are the key's location after obtaining the key since it becomes a movable cell, and represent the door's location after unlocked since it become a movable cell. Then, every $\mathbb{Z}^4$ vector will be encoded and put into a hash table.

For this scenario, we will use the following loss function:

$$\ell(\mathbf{x}, \mathbf{u}) = \begin{cases} 0 & \text{if facing key or door; } \mathbf{u} = 3, 4 \\ 1 & \text{if } f(\mathbf{x}, \mathbf{u}) \in \mathcal{X}; \mathbf{u} = 0, 1, 2 \\ \infty & \text{else} \end{cases},$$

So we incentivize the agent picking up the key and unlocking the door when needed. Any movement control input needs a positive value so the dp algorithm can distinguish which actions are most optimal. Set $T = |\mathcal{X}|$, and then we have everything initialized to apply the deterministic dp algorithm.

C. Psuedo-Random Environment

With some careful observation and thinking given the environment's general structure, it is possible to simplify the random environment problem:

- In general, when the agent desires to go from one cell to another, it is most optimal to minimize the number of turns throughout its traversal. This allows us to eliminate some states to not consider at all, decreasing the size of the state space. Since we want given a determinstic environment, we can consider getting rid of some states in order to steer the agent to a desirable path.
- If a key needs to be obtained, it is most optimal to unlock towards the closest door, regardless of goal location. Therefore, it is viable to only consider one door in any scenario.
- The door and key will spawn in three predetermined locations. These locations can be observed and encoded into the state space.

With these facts, we can narrow down the movable spaces to 19 spaces, excluding doors and goals.

$$\begin{aligned} \mathcal{M} = \{&[1,2],[1,3],[1,4],[1,5],[2,2], \\ &[2,5],[3,2],[3,3],[3,4],[3,5], \\ &[5,2],[5,3],[5,4],[5,5],[6,1], \\ &[6,2],[6,4],[6,5],[6,6]\} \end{aligned}$$

The format $\mathbf{x} \in \mathcal{X}$ will be as follows:

$$\mathbf{x} = \begin{bmatrix} x\ position \\ y\ position \\ direction \\ door/key\ state \\ door\ state \\ key\ state \\ goal\ state \end{bmatrix}$$

where the goal location is given by

$$(g_x, g_y) = \begin{cases} (5,1) & goal\ state = 0 \\ (6,3) & goal\ state = 1 \\ (5,6) & goal\ state = 2 \end{cases},$$

key location is given by

$$(k_x, k_y) = \begin{cases} (1,1) & key\ state = 0 \\ (2,3) & key\ state = 1 \\ (1,6) & key\ state = 2 \end{cases},$$

and the door's states are given by

$$\begin{cases} \text{both doors locked} & door\ state = 0 \\ (4,2) \text{ is unlocked} & door\ state = 1 \\ (4,5) \text{ is unlocked} & door\ state = 2 \\ \text{both doors unlocked} & door\ state = 3 \end{cases}$$

Construct the goal space and the state space

$$\begin{aligned} \mathcal{G} = \{(5,1),(6,3),(5,6)\} \times \{0,1,2,3\} \times \{0,1,2\} \\ \times \{0,1,2,3\} \times \{0,1,2\}^2 \end{aligned}$$

Algorithm 2 Random Environment State Space Construction

```
for $(d, i, j, k) \in \{0, 1, 2, 3\}^2 \times \{0, 1, 2\}^2$ do¸
    $\mathcal{X} \leftarrow (4, 2, d, 2 * (i\%2 == 0), i, j, k)$
    $\mathcal{X} \leftarrow (4, 5, d, 2 * (i < 2), i, j, k)$
    $\mathcal{X} \leftarrow (x, y, d, p, i, j, k), \forall(x, y) \in \mathcal{M}, p \in \{0, 1, 2, 3\}$
end for
return $\mathcal{X}$
```

$2 * (i\%2 == 0)$ and $2 * (i < 2)$ initializes the top and bottom door to be locked or unlocked depending on the door initialization. If open, then it will not require the agent to have $[\mathbf{x}]_4$, which is the door/key state, to be 2.

Construct the loss function:

$$\ell(\mathbf{x}, \mathbf{u}) = \begin{cases} 0 & \text{if facing closest key or door; } \mathbf{u} = 3, 4 \\ 1 & \text{if } f(\mathbf{x}, \mathbf{u}) \in \mathcal{X}; \mathbf{u} = 0, 1, 2 \\ \infty & \text{else} \end{cases},$$

and then set the time horizon: $T = |\mathcal{M} * 12|$ since it is the number of states disregarding the environment configuration. Proceed with the dp algorithm.

V. Results

A. Part A

The following figure is the optimal path for the $6 \times 6 - direct$ environment:

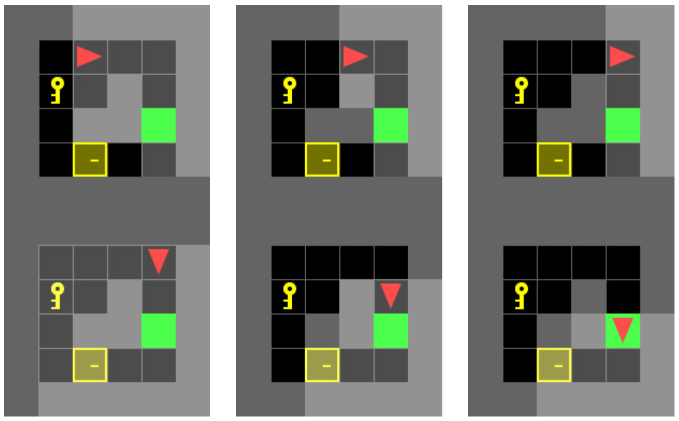

Fig. 1: Optimal path for the $6 \times 6 - direct$ environment (left-right, up-down)

More paths can be found at google drive link at the bottom of the document.

B. Part B

More paths can be found at google drive link at the bottom of the document.

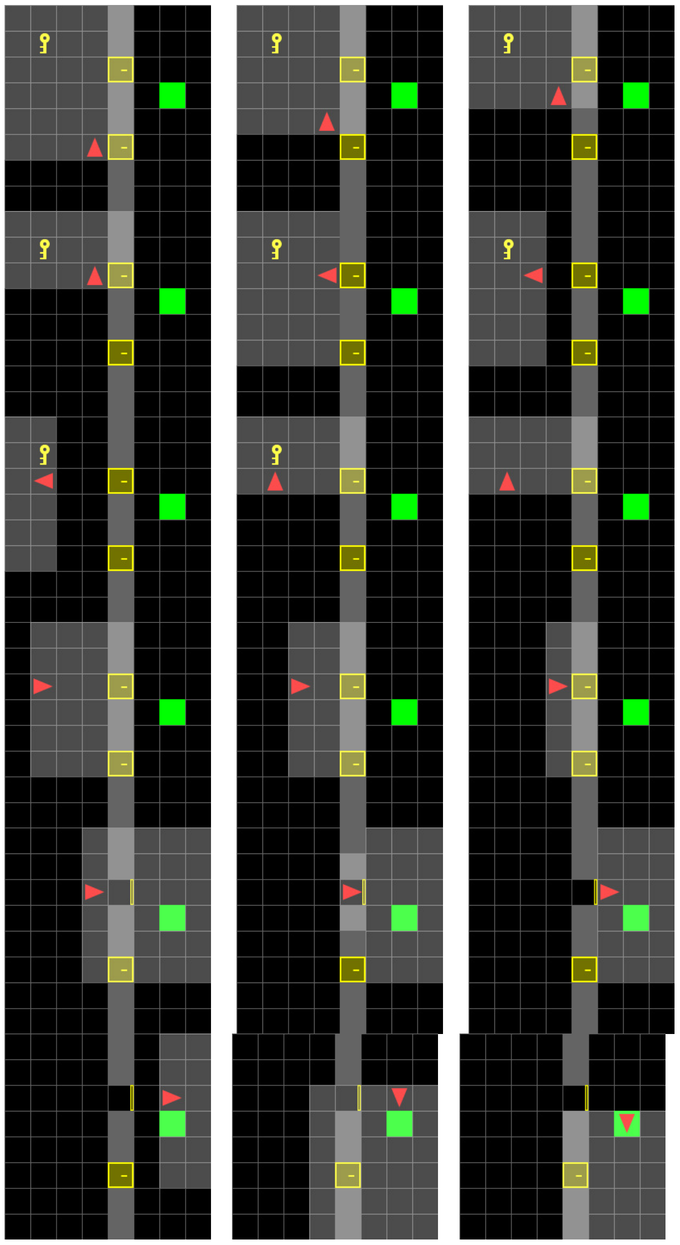

Fig. 2: Optimal path for the a random environment (left-right, up-down)

VI. Acknowledgments

Thank you to ECE276B Instructors who extended this deadline and prevented me from having a midterm on the due date of the project.

drive link