Project 1: Dynamic Programming

Behrad Rabiei
Dept. of Electrical and Computer Engineering
University of California San Diego
Email: brabiei@ucsd.edu

***Abstract*—This paper is the report for project 1 of ECE 276B (Planning and Learning in Robotics). The project requires you to find an optimal policy for navigating an environment using Dynamic Programming. We are asked to implement the algorithm to work with both know environments and unknown environments.**

I. Introduction

Autonomous robotic navigation represents a cornerstone of modern robotics, crucial for enhancing the autonomy and efficiency of robots in complex and dynamic environments. The development of robust navigation systems enables robots to perform a wide range of tasks, from industrial automation and logistics to autonomous vehicles and space exploration. By leveraging advanced sensors, algorithms, and learning techniques, these systems aim to improve safety, reduce human labor, and expand the capabilities of robots beyond traditional boundaries. The motivation behind advancing autonomous navigation is not only to increase operational efficiency but also to innovate in areas where human access is dangerous or impractical.

A. Markov Decision Process and Dynamic Programming in Autonomous Navigation

The Markov Decision Process (MDP) is a mathematical framework used for modeling decision making in situations where outcomes are partly random and partly under the control of a decision maker. MDPs are particularly useful in autonomous navigation, providing a structured way to determine the optimal path a robot should take in a stochastic environment. Dynamic Programming (DP) is a method for solving complex problems by breaking them down into simpler subproblems; it is crucial in solving MDPs efficiently. By utilizing DP, autonomous systems can evaluate the long-term consequences of their actions, ensuring that the chosen paths are not only locally optimal but also globally beneficial. This capability is indispensable for navigating through unpredictable terrains and adapting to new obstacles, enhancing both the robustness and reliability of autonomous robots.

II. Project Objective

The primary objective of this project is to design and implement a dynamic programming algorithm to efficiently guide an autonomous agent through a Door & Key environment. Our agent, depicted as a red triangle, must navigate to a goal location, shown as a green square, potentially obstructed by a closed door. To reach the goal, the agent might need to pick up a key to unlock the door, while managing its energy by minimizing the cost of actions such as moving forward (MF), turning left (TL), and turning right (TR), along with special actions like picking up a key (PK) and unlocking the door (UD). The project comprises two main scenarios: (A) a "Known Map" scenario where specific control policies are devised for each of seven distinct environments provided, and (B) a "Random Map" scenario that requires a robust control policy applicable across thirty-six random 8x8 environments. This task aims to enhance the agent's ability to adapt to both structured and unpredictable settings, thereby improving its autonomy and decision-making efficiency in dynamic conditions.

III. Problem Formulation

Now we will formally define our problem.

A. Action Space

The action space of the autonomous agent, denoted as $\mathcal{U}$, is defined as a set of possible actions that the agent can take at any given state. The action set includes:

$$\mathcal{U} = \{\text{MF}, \text{TL}, \text{TR}, \text{PK}, \text{UD}\}$$

where MF stands for Move Forward, TL for Turn Left, TR for Turn Right, PK for Pick Up Key, and UD for Unlock Door. Each action is associated with a positive energy cost, reflecting the effort required to execute the action.

B. State Space: Part A

The state space, denoted as $\mathcal{X}$, describes all possible situations that the agent can encounter within the environment. A state in $\mathcal{X}$ is defined by a dictionary containing the following elements:

$$\mathcal{X} = \{('x\_position', 'y\_position', 'direction', 'havekey', 'isgoal')\}$$

where x and y represent the agent's coordinates on the grid, d denotes the direction the agent is facing, k is a boolean indicating whether the agent has picked up the key, and is_goal is a boolean that becomes true when the agent reaches the goal location. The directions are encoded as:

- LEFT = 0
- UP = 1
- RIGHT = 2
- DOWN = 3

C. Constraints on State Space

The state space $\mathcal{X}$ is constrained such that states where the agent's x and y coordinates coincide with wall locations are excluded. This constraint ensures that the agent's path planning algorithm only considers valid movement within the boundaries and internal structures of the environment.

D. Motion Model: Part A

The motion model of the agent is a critical component in simulating how actions affect the state within the environment. It is defined mathematically by the transition function $T : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$, which maps a state and an action to a new state according to the rules specified below:

- **Move Forward (MF):** If the action is MF, the agent moves in the direction it is currently facing. The new position (x', y') is calculated as:

$$x' = x + \Delta x(d), \quad y' = y + \Delta y(d)$$

  where $\Delta x(d)$ and $\Delta y(d)$ are directional increments based on the agent's current direction d. The agent's goal state is updated if it reaches the goal position. Movement is constrained by the presence of doors and walls; the agent remains in the current state if a move would result in a collision with a closed door or a wall.
- **Turn Left (TL) and Turn Right (TR):** These actions modify the agent's direction:

$$d' = (d + \delta) \mod 4$$

  where δ is -1 for TL and $+1$ for TR, representing a counter-clockwise and clockwise turn, respectively.
- **Pick Up Key (PK):** The agent attempts to pick up a key if it is directly in front of it. If successful (*i.e.*, there is a key in the adjacent cell in the direction the agent is facing), the 'have_key' state is set to true. Otherwise, the state remains unchanged.
- **Unlock Door (UD):** The agent attempts to unlock a door if it is directly in front of it and the agent possesses a key. The state changes to place the agent on the other side of the door if successful. If the door cannot be unlocked or there is no door, the state remains unchanged.

The transition function ensures that the agent only transitions to states within the predefined state space $\mathcal{X}$, excluding any state corresponding to invalid positions (e.g., outside the grid or within a wall).

E. Extended State Space Definition: Part B

In the "Unknown Environment" task, the state space is extended to capture information about the environment. Since we know that the key, door states, and goal location are from a limited list of possibilities, we try to adapt our state space to keep track of their possible spawning locations. The state for this task is defined as a tuple $x \in \mathcal{X}$ where:

$$\begin{aligned} x = \{(&'x\_position' : x, \\ &'y\_position' : y, \\ &'direction' : d, \\ &'have\_key' : \text{key}, \\ &'door1' : d1, \\ &'door2' : d2, \\ &'goal\_loc' : g, \\ &'key\_loc' : k)\} \end{aligned}$$

Each component of the state tuple is described as follows:

- x, y: Coordinates of the agent on the grid.
- d: The current direction the agent is facing, encoded numerically.
- key: A boolean indicating whether the agent has picked up the key.
- $d1, d2$: States of the doors (open or closed), reflecting the conditions of specific barriers that might affect navigational strategies.
- g: The location of the goal, which can be one of three locations.
- k: The location of the key, which, like the goal, may be one of three locations.

F. Motion Model: Part B

1) Action Dynamics:

- **Move Forward (MF):** When the action is MF, the agent moves in its current direction:

$$x' = x + \Delta x(d), \quad y' = y + \Delta y(d)$$

  If the new position (x', y') coincides with a wall location, the state remains unchanged. Additional conditions are checked as follows:

$$\begin{aligned} &\text{If } (x', y') \text{ in key locations and have\_key} = 0, \\ &\quad \text{and } (x', y') = \text{key\_locations[key\_loc]}, \\ &\text{then return state.} \end{aligned}$$

$$\begin{aligned} &\text{If } (x', y') \text{ in door locations,} \\ &\quad \text{and } (x', y') = \text{door\_locations}[i] \text{ and door}[i] = 0, \text{ then return state} \end{aligned}$$

- **Turn Left (TL) and Turn Right (TR):** These actions adjust the agent's facing direction:

$$d' = (d + \delta) \mod 4$$

  where δ is -1 for TL and $+1$ for TR.
- **Pick Up Key (PK):** The 'have_key' state is updated under the following condition:

$$\begin{aligned} &\text{If } (x + \Delta x(d), y + \Delta y(d)) = \text{key\_locations}[k] \\ &\quad \text{and have\_key} = 0, \text{ then set have\_key} = 1. \end{aligned}$$

  Otherwise, the state remains unchanged.

- **Unlock Door (UD):** If the conditions are met, the door's state is updated and the agent moves through:

$$\begin{aligned}&\text{If } (x + \Delta x(d), y + \Delta y(d)) = \text{door\_locations}[i]\\ &\quad \text{and have\_key} = 1 \text{ and door}[i] = 0,\\ &\quad \text{then set door}[i] = 1 \text{ and update position.}\end{aligned}$$

  If these conditions are not met, the state remains unchanged.

2) State Validation: After any action, the resulting state is checked against all valid states in $\mathcal{X}$. If the new state does not exist in $\mathcal{X}$, the state reverts to the previous state, ensuring that the agent always remains in a valid state.

G. Time Horizon

The time horizon T of the decision process is defined as the cardinality of the state space $\mathcal{X}$ minus one, $T = |\mathcal{X}| - 1$. This formulation of the time horizon reflects the maximum number of transitions the agent might reasonably make to explore all possible states before reaching a terminal condition, ensuring comprehensive coverage of the state space in the decision-making process.

H. Cost Functions

1) Intermediate Cost Function: The intermediate cost function $c : \mathcal{X} \times \mathcal{U} \to \mathbb{R}$ assigns a cost to each action taken by the agent from any state. The cost associated with all actions is uniformly set to 1, except when an action directly leads to the goal position. In such a case, the cost is considered as part of the terminal cost, hence not incremented here. This cost structure incentivizes the agent to find the shortest path to the goal under normal circumstances.

2) Terminal Cost Function: The terminal cost function $c_T : \mathcal{X} \to \mathbb{R}$ is defined based on the goal achievement:

$$c_T(x) = \begin{cases} 0 & \text{if is\_goal}(x) = 1 \\ \infty & \text{otherwise} \end{cases}$$

This function assigns a cost of zero if the agent reaches the goal state, indicating successful completion of its objective. Conversely, if the agent fails to reach the goal state by the end of the time horizon, the cost is set to infinity, reflecting the failure of the mission and strongly penalizing non-optimal paths that do not lead to the goal.

IV. Technical Approach

In this project, we developed a dynamic programming algorithm to solve the Markov Decision Process (MDP) for autonomous robotic navigation in a simulated environment known as the Door & Key scenario. Our approach systematically defines and addresses the complexities associated with both known and unknown map scenarios, leveraging the principles of dynamic programming to compute an optimal policy for navigating through these environments.

A. Dynamic Programming Algorithm Implementation

The core of our technical approach involves implementing the dynamic programming algorithm outlined in lecture 4. The algorithm accepts the following inputs:

- A finite set of states $\mathcal{X}$ and actions $\mathcal{U}$,
- Initial and terminal state distributions μ_0 and μ_f,
- An intermediate cost function ℓ,
- A transition probability p_f,
- A terminal time T,
- A discount factor γ,
- A state-value function q.

The algorithm proceeds by initializing the value function $V_T(x)$ for each state x as $q(x)$. It then iteratively calculates the cost $Q_t(x, u)$ for each action u and updates the value function $V_t(x)$ by finding the minimum expected cost over all actions. The optimal policy $\pi_t(x)$ at each state is determined by selecting the action that minimizes $Q_t(x, u)$.

B. Solving for Known and Unknown Environments

1) **Known Map Scenario:**
   - **State Space and Action Dynamics:** The agent's state space and action set were clearly defined, with specific rules governing the transitions based on the agent's interactions with the environment, such as moving, turning, picking up a key, and unlocking a door.
   - **Algorithm Application:** For each of the seven predefined environments, the dynamic programming algorithm was applied using the exact map layout and the initial conditions provided. This allowed for a precise calculation of the optimal policy tailored to each specific environment.
2) **Random Map Scenario:**
   - **Extended State Space:** In scenarios where the map configuration was not known in advance, we extended the state space to include potential configurations of keys, doors, and goal locations. This adaptation was necessary to accommodate the variability in the map layouts and to ensure robustness in the policy generated.
   - **Adaptive Policy Computation:** The algorithm was designed to adjust dynamically as the agent discovered new information about the environment. This involved recalculating the optimal policy based on the updated state information, ensuring that the agent could adapt to unforeseen obstacles and changes in the environment layout.

V. Results

The following are the results of the algorithm. We can see that in instances where the key is not necessary, the agent goes straight to the goal. In instances where the key is necessary then it will get the key first. I have shown frame by frame for doorkey-6x6-direct and DoorKey-8x8-9. The rest of the paths are written out.

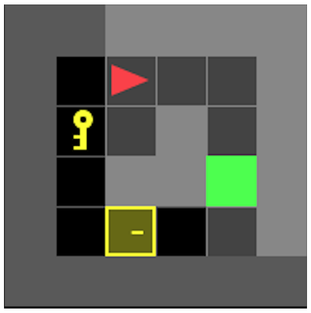

Fig. 1. Frame 1 for direct environment

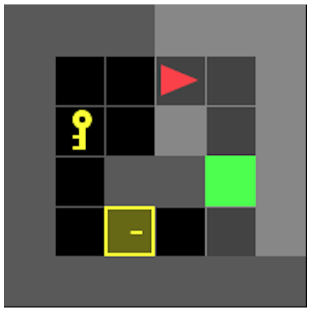

Fig. 2. Frame 2 for direct environment

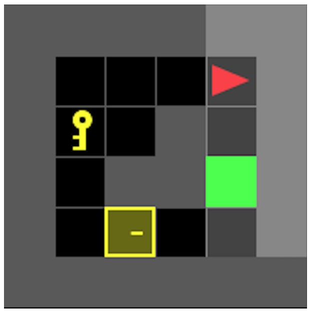

Fig. 3. Frame 3 for direct environment

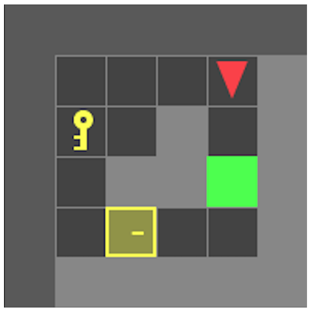

Fig. 4. Frame 4 for direct environment

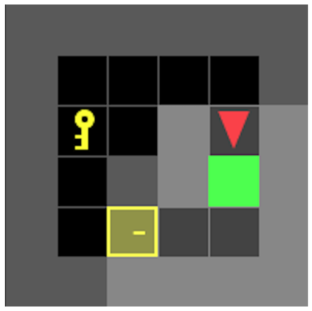

Fig. 5. Frame 5 for direct environment

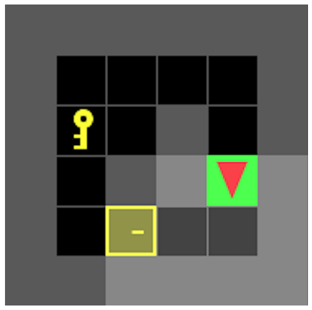

Fig. 6. Frame 6 for direct environment

Known Environment:
Random Environment:

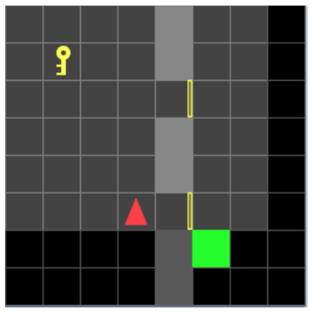

Fig. 7. Frame 1 for random environment

A. *Part A*

doorkey-5x5-normal [TR, TR, PK, TR, UD, MF, MF, TR, MF]
doorkey-6x6-normal [TL, MF, PK, TR, TR, MF, TR, MF, UD, MF, MF, TR, 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-6x6-direct [MF, MF, TR, MF, MF]
doorkey-8x8-direct [MF, TL, MF, MF, MF, TL, MF]
doorkey-6x6-shortcut [PK, TL, TL, UD, MF, MF]
doorkey-8x8-shortcut [TR, MF, TR, PK, TL, UD, MF, MF]

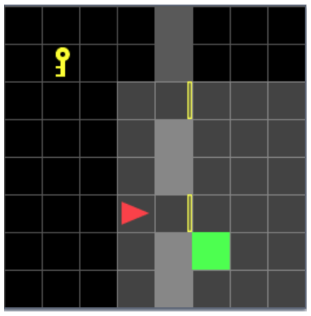

Fig. 8. Frame 2 for random environment

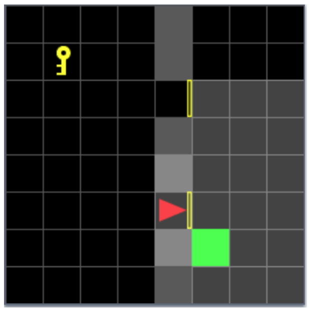

Fig. 9. Frame 3 for random environment

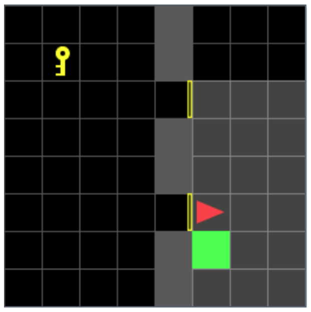

Fig. 10. Frame 4 for random environment

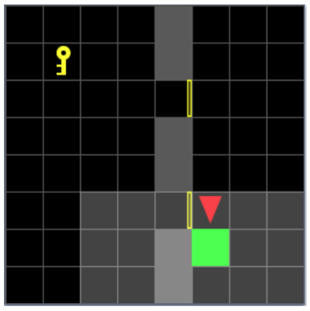

Fig. 11. Frame 5 for random environment

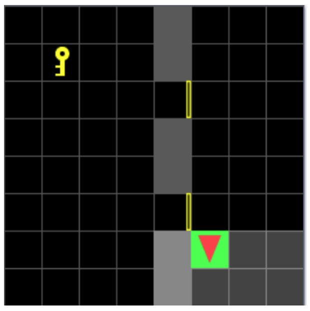

Fig. 12. Frame 6 for random environment

B. Part B

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, MF, TL, MF, PK, TL, MF, TL, MF, UD, MF, MF, TL, MF]
DoorKey-8x8-5 [TR, MF, MF, MF, TL, MF, MF]
DoorKey-8x8-6 [MF, MF, MF, TR, MF, MF, MF, TR, MF]
DoorKey-8x8-7 [TR, MF, MF, MF, TL, MF, MF]
DoorKey-8x8-8 [MF, MF, MF, MF, TL, MF, PK, TL, MF, TL, 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, TL, MF, UD, MF, MF, TR, MF, MF, MF, 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]

VI. Conclusion

The dynamic programming algorithm effectively computed optimal navigation policies for both known and unknown environments. The adaptability and robustness of the algorithm were demonstrated through its success in handling diverse and dynamic challenges in the simulation scenarios, validating its potential for real-world applications in autonomous robotics navigation.

References

[1] https://natanaso.github.io/ece276b/

CONTENTS