Dynamic Programming

Chung-Pang, Wang
University of California San Diego
Department of Electrical and Computer Engineering

***Abstract*—This project focuses on autonomous navigation in a Door & Key environment. The objective is to get our agent to the goal location. The environment may contain a door which blocks the way to the goal. If the door is closed, the agent needs to pick up a key to unlock the door. We will formulate Door & Key problem as a Deterministic Shortest Path problem and implement a Dynamic Programming algorithm to find the optimal path by minimizing the cost of reaching the goal.**

***Index Terms*—Dynamic Programming, Door & Key problem**

I. Introduction

The Door & Key problem involves an agent navigating through an grid environment with the goal of reaching a specific destination from specific starting position and orientation with minimal cost. Furthermore, the Door & Key problem requires the agent to interact with door and a key, adding layers of complexity and decision-making. To obtain the optimal path from the starting state to the goal, we need to compute a optimal policy that maps states to controls that cost agent the least to reach the goal. We can formulate the Door & Key problem as a Markov Decision Process (MDP) and obtain optimal policy by utilizing Dynamic Programming to solve the MDP. In the first part of the project, we will perform Dynamic Programming algorithm 7 times for 7 different known environments that the position of key, door and goal are known. We will have 7 different optimal policy for 7 different environments. In the second part of the project, we will perform a single Dynamic Programming algorithm for 36 random environments that have several possible key, door and goal position, etc. By expanding the state space, the algorithm can encompass all possible variations or configurations within the environments, ensuring that it captures every potential scenario or possibility.

II. Problem Formulation

A. Markov Decision Process for Door & Key problem

To formulate the Markov Decision Process for the Door & Key problem we first need to define discrete state space $\mathcal{X}$, discrete control space $\mathcal{U}$, prior $p_0(\cdot)$, $p_f(\cdot, x_t, u_t)$ conditional pdf defined on $\mathcal{X}$ for given $\mathbf{x} \in \mathcal{X}$ and $\mathbf{u} \in \mathcal{U}$, finite time horizon T, $\ell(\mathbf{x}, \mathbf{u})$ stage cost of applying control $\mathbf{u} \in \mathcal{U}$ in state $\mathbf{x} \in \mathcal{X}$, $\mathfrak{q}(\mathbf{x})$ terminal cost of being in state $\mathbf{x}$ at time T and discount factor $\gamma \in [0, 1]$. Since the states and the state transitions are fully deterministic, we will not need prior and conditional pdf in the algorithm. Besides, we will set discount factor to be 1 to simplify the algorithm. In the first part of the project, Door & Key problem consist of position of the agent, orientation of the agent, door status (open or closed) and key status (picked or not). There are 5 control actions $\mathbf{u}$ Move Forward (MF), Turn Left (TL), Turn Right (TR), Pickup the key (PK) and Unlock the door (UD).

$$\begin{aligned} \mathbf{x} &= (\mathbf{x_t},\ \mathbf{y_t},\ \mathbf{Orientation},\ \mathbf{Door},\ \mathbf{Key}) \\ \mathbf{u} &= \{\mathbf{MF},\ \mathbf{TL},\ \mathbf{TR},\ \mathbf{PK},\ \mathbf{UD}\} \end{aligned} \tag{1}$$

where $Door \in \{0, 1\}$, $Key \in \{0, 1\}$ and $Orientation \in \{Left, Right, Up, Down\}$. Stage cost, the cost of each action is one and set to 0 if the state reaches the goal. Terminal cost, the cost to terminate at state $\mathbf{x}$ is designated as infinite across all states with the exception of the goal state, at which point it is nullified.

$$\begin{aligned} \ell(x, u) &= \begin{cases} 0 & \text{if } x = \tau \\ 1 & \text{otherwise} \end{cases} \\ \mathfrak{q}(x) &= \begin{cases} 0 & \text{if } x = \tau \\ \infty & \text{otherwise} \end{cases} \end{aligned} \tag{2}$$

B. Solving MDP with Dynamic Programming

Now we have the model of the MDP for Door & Key problem. To obtain the optimal path of MDP, we will define a control policy function $\pi_t(\mathbf{x})$, a function from state $\mathbf{x}$ at time t to control $\mathbf{u}$ and a Value function $V_t^\pi(\mathbf{x})$, an expected long-term cost starting in state $\mathbf{x}$ at time t and following policy π. With these two function, we can now optimize the functions over controls $\mathbf{u}$ to obtain optimal value function $V_t^*(\mathbf{x})$ and optimal control policy $\pi_t^*(\mathbf{x})$. The minimization formulation to find minimizer of the value function and it's corresponding optimal policy is similar to optimal control problem. Optimal control problem can be written as:

$$\begin{aligned} V_0^*(\mathbf{x}_0) &= \min_\pi V_0^\pi(\mathbf{x}_0) \\ \pi^* &\in \arg\min_\pi V_0^\pi(\mathbf{x}_0) \end{aligned} \tag{3}$$

We can solve the MDP problem by reformulating it as an optimal control problem and solve it with Dynamic Programming algorithm (DP). Dynamic Programming is an algorithm for computing the optimal value function $V_0^*(\mathbf{x}_0)$ and an optimal policy π^* backwards in time. In most cases, DP is much more efficient than a brute-force approach evaluating all possible policies. For this Door & Key problem specifically, we do not need probability term since the problem is deterministic, we will ignore the terms related to probabilities computation. Dynamic Programming algorithm states as Algorithm 1.

Algorithm 1 Dynamic Programming for MDP

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

x' is the next state given current state $\mathbf{x}$ and control $\mathbf{u}$. x' can be computed by motion model $f(\mathbf{x}, \mathbf{u})$.

III. Methods

A. Deterministic Shortest Path via Dynamic Programming

Solving the Door & Key problem involves dealing with start and goal states. However, in the original formulation of Dynamic Programming, there was no goal and start node. Therefore, we can reformulate DP with the format of considering start and goal states to solve the problem more easily. The Door & Key problem is a type of deterministic shortest path problem since we are finding the shortest path from the starting states to the goal states and the states transitions are deterministic. In fact, The finite-state deterministic shortest path problem is equivalent to a finite-horizon finite-state deterministic optimal control (DOC) problem which is what we formulated earlier. Thus, we proposed a deterministic shortest path version of dynamic programming (as shown in Alg.2)to solve the Door & Key problem.

Algorithm 2 DSP via Dynamic Programming

Input: vertices V, start $s \in V$, goal $\tau \in V$, and costs c_{ij} for $i, j \in V$
$T = |V| - 1$
$V_T(\tau) = V_{T-1}(\tau) = \cdots = V_0(\tau) = 0$
$V_T(i) = \infty, \forall i \in V \setminus \{\tau\}$
$V_{T-1}(i) = c_{i\tau}, \forall i \in V \setminus \{\tau\}$
$\pi_{T-1}(i) = \tau, \forall i \in V \setminus \{\tau\}$
for $t = T-2, ..., 0$ **do**
  $Q_t(i,j) = c_{ij} + V_{t+1}(j), \quad \forall i \in V \setminus \{\tau\}, j \in V$
  $V_t(i) = \min_{j \in V} Q_t(i,j), \quad \forall i \in V \setminus \{\tau\}$
  $\pi_t(i) = \text{argmin}_{j \in V} Q_t(i,j), \quad \forall i \in V \setminus \{\tau\}$
  if $V_t(i) = V_{t+1}(i), \forall i \in V \setminus \{\tau\}$ **then**
    break
  end if
end for

The time horizon for the DP is the number of all possible states minus 1. $V_t(i)$ is the optimal cost-to-go from node i to node τ in at most $T - t$ steps and the algorithm can be terminated early if $V_t(i) = V_{t+1}(i), \ \forall i \in V \setminus \{\tau\}$.

B. Dynamic Programming for Known Environments

As mentioned in (1), the states $\mathbf{x} \in \mathbb{R}^{N \times N \times 4 \times 2 \times 2}$ for known Environments including the position and orientation of the agent, key status and door status as shown in Fig.1 . There are 5 possible actions that can be executed in each step and each actions cost equally. For known environments, start position, orientation and goal position are known. Since we will not know the orientation of the agent when it reach the goal, the orientation of the goal is not specified beforehand. Initially, we set the value at the goal state at each time step in the value function to be 0 and everywhere else to be infinity. In each time step, we first compute the sum of the cost of the current action and evaluate the value function in previous time step at the new state given by the motion model. The motion model for Known Environments:

$$f(\mathbf{x}, \mathbf{u}) = \begin{cases} Move\ Forward & \text{if} \quad u = \text{MF} \\ Turn\ Left & \text{if} \quad u = \text{TF} \\ Turn\ Right & \text{if} \quad u = \text{TR} \\ Key = 1 & \text{if} \quad u = \text{PK} \\ Door = 1 & \text{if} \quad u = \text{UD} \end{cases} \tag{4}$$

Besides, we need to set some constraints to eliminate invalid actions or states. If there is an obstacle in front of the agent, then the agent can not move forward. If there are no door or key in front of the agent, then the agent will cost 1 for doing nothing and if the agent don't have a key, even if there is a door in front the agent still cannot unlock the door.

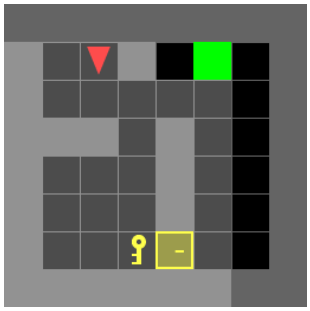

Fig. 1. Known Environment

C. Dynamic Programming for Random Environments

Unlike known environments, in random environments, there are 3 possible key positions $\{(1,1),(2,3),(1,6)\}$, 3 possible goal positions $\{(5,1),(6,3),(5,6)\}$ and 2 doors at $(4,2)$ and $(4,5)$ with 4 possible door status. Random environments are all 8×8 grids, the states space become extremely large compare to the previous known environments. Therefore, we need to alter the states defined in (1). The states for random environments:

$$\mathbf{x} = (\mathbf{x_t},\ \mathbf{y_t},\ \textbf{Orientation},\ \textbf{Goal},\ \textbf{Door Status},\ \textbf{Key Position},\ \textbf{Key Status}) \tag{5}$$

where

$$Door\ Status = \left\{ \begin{array}{c} \text{Door1 Locked Door2 Locked,} \\ \text{Door1 Unlocked Door2 Unlocked,} \\ \text{Door1 Locked Door2 Unlocked,} \\ \text{Door1 Unlocked Door2 Locked} \end{array} \right\}$$

$$Key\ Status = \left\{ \begin{array}{c} \text{Key Picked Up,} \\ \text{Key Not Picked Up} \end{array} \right\}$$

Since the states have changed, motion model will also change. The structure of the motion model is still the same as (4), we only need to change the conditions for PK and UD to adapt to different possible key, door positions and status.

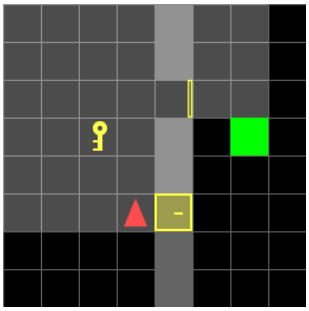

Fig. 2. Known Environment

IV. Implementation Details

A. Known Environments

For known environments, there are $height \times width \times 4 \times 2 \times 2$ states, take 5×5 grids environment as example, there are 400 possible states. The states are defined as an $height \times width \times 4 \times 2 \times 2$ array:

$$\textbf{States} : (\text{x},\ \text{y},\ \text{Headings Id}\ (0, .., 4), \text{door status}\ (0\ or\ 1), \text{key status}\ (0\ or\ 1))$$

I designed the dimensions of both the value function array and the optimal policy array to match the dimensions of the states array. This alignment allows for straightforward and efficient access to values corresponding to each state. Then, I strictly follow Algorithm 2 to iterate through all of the possible position, orientations, door status, key status and actions to first compute the Q value and only append the action and value if the Q value is less than the value function evaluate at the new state. Finally, we have the optimal policy that tells the agent which action will cost the least to get to the goal at given states. Indexing optimal policy array from the starting states will lead us to the goal with minimum cost.

B. Random Environments

For Random environments, there are $8 \times 8 \times 4 \times 3 \times 4 \times 3 \times 2$ states as shown in (5), thus, there are 18432 possible states. The states are defined as an $8 \times 8 \times 4 \times 3 \times 4 \times 3 \times 2$ array:

$$\begin{aligned} \textbf{States} = (&\text{x},\ \text{y},\ \text{Headings ID}\ (0, .., 4), \\ &\text{Goal Positions ID}\ (0, 1, 2)\ \text{Door Status}\ (0\ or\ 1), \\ &\text{Key Positions Id}\ (0, 1, 2),\ \text{Key Status}\ (0\ or\ 1)) \end{aligned}$$

TABLE I
Optimal Policy for Known Environments

Optimal Actions	1	2	3	4	5	6	7	8	9	Total Costs
5x5-normal	TL	TL	PK	TR	UD	MF	MF	TR	MF	9
6x6-direct	MF	MF	TR	MF	MF	–	–	–	–	5
6x6-normal	TL	MF	PK	TL	MF	TL	MF	TR	–	8
6x6-shortcut	PK	TL	TL	UD	MF	MF	–	–	–	6
8x8-shortcut	TR	MF	TR	PK	TL	UD	MD	MF	–	8
8x8-direct	MF	TL	MF	MF	MF	TL	MF	–	–	7
8x8-normal	TR	MF	TL	MF	TR	MF	MF	MF	PK	–
8x8-normal-continued	TL	TL	MF	MF	MF	TR	UD	MF	MF	–
8x8-normal-continued	MF	TR	MF	MF	MF	–	–	–	–	23

The dimensions of both the value function array and the optimal policy array to match the dimensions of the states array. I also strictly follow Algorithm 2 to iterate through all possible states and actions. Even though the states space is larger, due to the early termination condition implemented in line 12 of Algorithm 2, the path finding program concludes significantly prior to reaching the terminal horizon at $|V| - 1$.

V. Results

A. Known Environments

Since the environments are known, we are executing path finding algorithm 7 times, obtaining 7 optimal policies for 7 different environments as shown in table 1. Even though the time horizon are few hundred iterations, my path finding algorithm terminated within 20 iterations due to value function termination condition. As shown in Fig. 3. to Fig. 9., the algorithm successfully generate optimal policies that lead the agent to reach the goal with minimum cost.

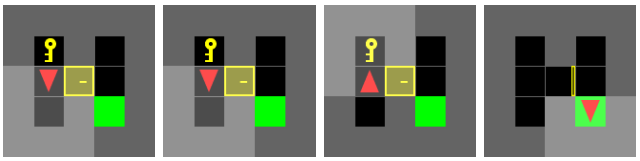

Fig. 3. doorkey-5x5-normal

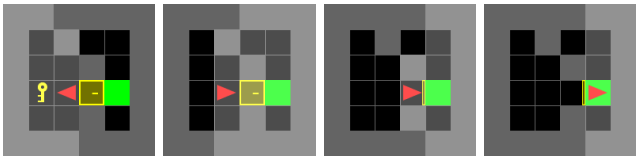

Fig. 4. doorkey-6x6-shortcut

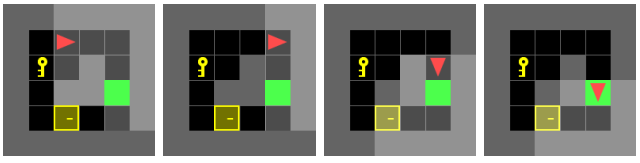

Fig. 5. doorkey-6x6-direct

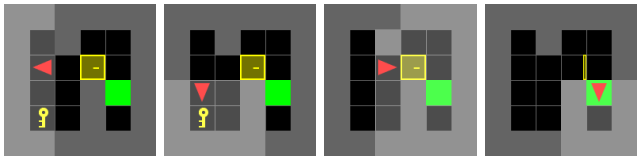

Fig. 6. doorkey-6x6-normal

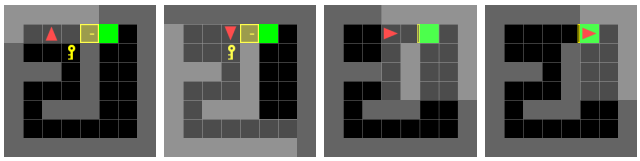

Fig. 7. doorkey-8x8-shortcut

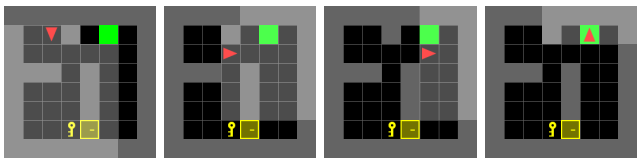

Fig. 8. doorkey-8x8-direct

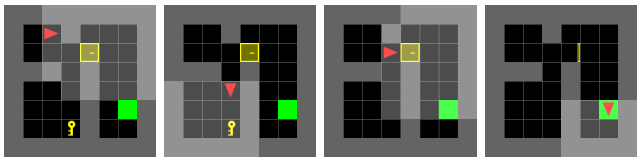

Fig. 9. doorkey-8x8-normal

B. Random Environments

For the random environments case, we can only use a single policy for all 36 random environments. By specifying each possibility into our state space and iterate through all of the possible states in the dynamic programming algorithm, the optimal policy will automatically capture all possible goal, key positions and door status as shown in Fig. 10 to Fig. 13. The time horizon for random environments is 18431 which is significantly larger than known environments case due to uncertainty. However, my algorithm find the optimal path with 23 iterations and terminated within 10 seconds, showing the power of dynamic programming solving such complex decision-making problem.

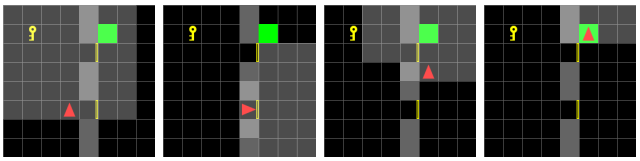

Fig. 10. Random DoorKey-8x8-1

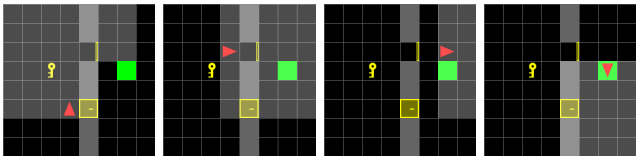

Fig. 11. Random DoorKey-8x8-1

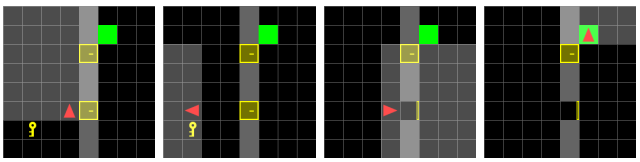

Fig. 12. Random DoorKey-8x8-1

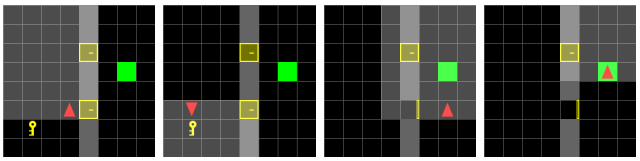

Fig. 13. Random DoorKey-8x8-1

REFERENCES

[1] UCSD ECE276B slides on Markov Chains https://natanaso.github.io/ece276b/ref/ECE276B_2_MC.pdf

[2] UCSD ECE276B slides on Markov Decision Processeshttps://natanaso.github.io/ece276b/ref/ECE276B_3_MDP.pdf

[3] UCSD ECE276B slides on Deterministic Shortest Pathhttps://natanaso.github.io/ece276b/ref/ECE276B_5_DSP.pdf