UCSD Spring 2021 ECE276B: Door Key Problem via Dynamic Programming

Steven Wong
Machine Learning and Data Science
University of California, San Diego
San Diego, CA 92093
scw039@ucsd.edu

April 26, 2021

1 Introduction

In this project, we model the door key problem as a Markov Decision Process and attempt to solve it. The door key problem is one in which an agent located in a grid has to navigate to the goal cell in as little number of moves as possible. If there is a locked door in its way, the agent would have to first go find the key in order to proceed. This has potential application to robotics as we are planning a robot's trajectory given its surroundings. In this case, we minimize the number of steps taken, but really this cost can be defined as anywhere from fuel consumption to time taken. We will approach the door key problem using the Dynamic Programming Algorithm.

2 Problem Statement

We will formulate the door key problem as a Markov Decision Process with the following definitions. The state space $\mathcal{X}$ includes all possible states $\mathbf{x}$ which encode all the following attributes:

$$\mathbf{x} \in \begin{bmatrix} \text{Agent position} \in \{\text{valid states}\} \\ \text{Agent current direction} \in \{\text{up, down, left, right}\} \\ \text{Whether or not agent is carrying a key} \in \{\text{yes, no}\} \\ \text{Whether or not the door(s) is/are open} \in \{\text{yes, no}\} \\ \text{The door(s) location} \\ \text{The goal location} \\ \text{The key location} \end{bmatrix}$$

We define valid states of the agent's position more closely in the technical approach section of this paper. The control space $\mathcal{U}$ includes all the possible moves an agent can do at any time step. Any action $\mathbf{u}$ has:

$$\mathbf{u} \in \{\text{move forward, turn left, turn right, pick up key, unlock door}\}$$

The motion model is a function $f(x_t, u_t)$ that takes in the state and the action, and produces the next state x_{t+1}. For example, an agent who has the action command to turn left will end up in the same cell facing the cardinal direction that results in the left turn. Of these actions, some are valid at a given time step, and some are not. Those details will be described in the technical approach. The initial state x_0 will be where the agent starts off on the map, with all 7 attributes noted accordingly. The cost of going from one state to another, i.e. taking an action $\mathbf{u}$ will be $l(x, u) = 1$. Our terminal cost will be defined as follows:

$$\mathbf{q}(x) = \begin{cases} 0 & \text{x is at the goal} \\ \infty & \text{x is not at the goal} \end{cases}$$

The planning horizon T can be set to any number of time steps, as we will deploy a smart early termination to our DPA. This method will be further discussed in the next section. Finally, we will have a discount factor $\gamma = 1$. With these variables defined, we have now created a Markov Decision Process for the door key problem.

3 Technical Approach

As mentioned in the above section, we will touch upon how we define valid states and actions before we go into the Dynamic Programming Algorithm and its variables' data structures. Valid states are defined as follows:

State **x**'s location must not be a wall. (i.e. walls are invalid states)

Actions are only valid provided they do not fall under ones of these categories:

if the cell in front of state **x**'s location is a door and it is locked, we cannot move forward
if the cell in front of state **x**'s location is a door and we do not have a key, we cannot unlock the door
if the cell in front of state **x**'s location is not a key, we cannot pick up key

Next, we will use the Dynamic Programming Algorithm to solve the door key problem by following the pseudo code below.

Algorithm 1: Dynamic Programming Algorithm

Result: Returns the optimal control policy π
Inputs
All MDP variables defined in problem statement section ($\mathcal{X}$, $\mathcal{U}$, f, T, l , $\mathbf{q}$)
Initialization:
$V_T(x) = \mathbf{q}(x)$
for *t = (T-1) ... 0* **do**
  Calculate new state x_{t+1} using motion model f
  $Q_t(x, u) = l(x, u) + V_{t+1}(x_{t+1})$
  $V_t(x) = \min_{u \in \mathcal{U}(x)} Q_t(x, u)$ for every state x
  $\pi_t(x) = \arg\min_{u \in \mathcal{U}(x)} Q_t(x, u)$ for every state x
  if V_{t+1} is the same as V_t: terminate the program
end

To store the values Q, V, and π, we define them as follows:

Q is a ($T \times$ number of states $\times$ number of actions) matrix
V is a ($T \times$ number of states) matrix
π is a ($T \times$ number of states) matrix

Notice the early termination in the algorithm. If two consecutive rows of V are the same, it means an optimal path has been found, and thus we do not actually need to compute the rest of the time steps, for they will all be the same. Once π is computed, we start at the last row (first action we take at starting position) and work our way to the first. We use the motion model to come up with the next state using the action from π at that time step. This allows us to recover the entire action sequence.

4 Results

Here we explore the results of the Dynamic Programming Algorithm on two types of scenarios - the known and random map.

4.1 Results - Known Map

We compute the control policy for 7 pre-existing environments, and evaluate its performance. Below are examples of the seven solutions the Dynammic Programming Algorithm achieves. GIFs for all 7 solutions can be found attached to this paper.

4.1.1 Known 5x5 normal

Here we see a 5x5 normal case- the agent must pick up the key, unlock the door, before proceeding to the goal.

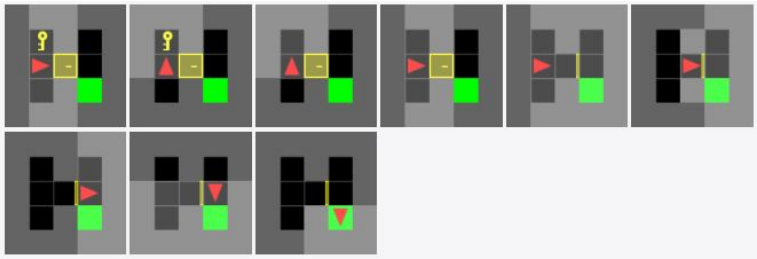

Figure 1: Known 5x5 Normal Results over time

Optimum Control Policy: [TL, PK, TR, UL, MF, MF, TR, MF]

4.1.2 Known 6x6 direct

Here we see the 6x6 direct case- the agent doesn't pick up the key and instead goes right for the door.

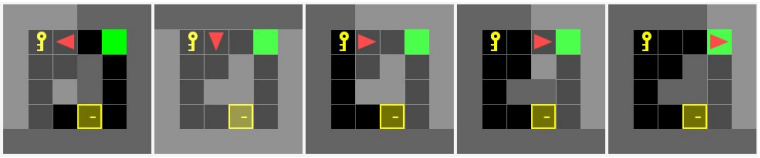

Figure 2: Known 6x6 Direct Results over time

Optimum Control Policy: [TL, TL, MF, MF]

4.1.3 Known 6x6 normal

Here we see a 6x6 normal case- the agent must pick up the key, unlock the door, before proceeding to the goal.

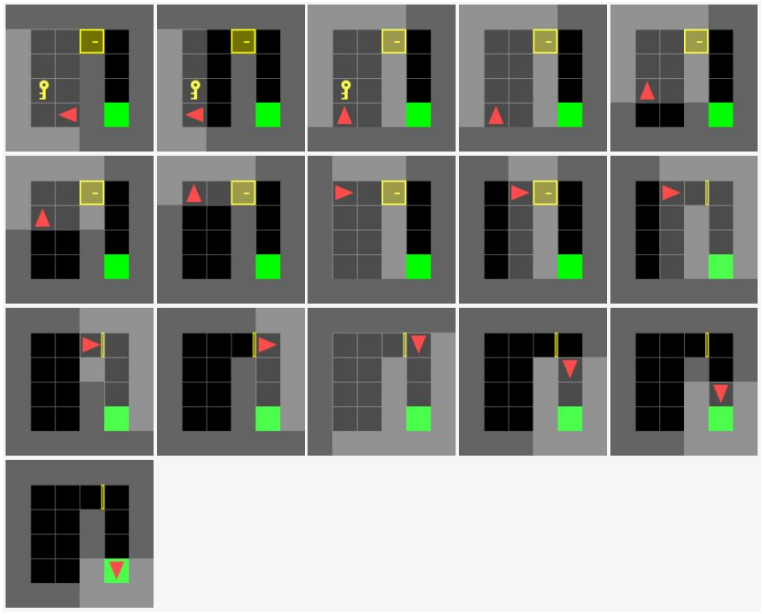

Figure 3: Known 6x6 Normal Results over time

Optimum Control Policy: [MF, TR, PK, MF, MF, MF, TR, MF, UD, MF, MF, TR, MF, MF, MF]

4.1.4 Known 6x6 shortcut

Here we see a 6x6 shortcut case- the agent must pick up the key and unlock the door since it is closer than going all the way around.

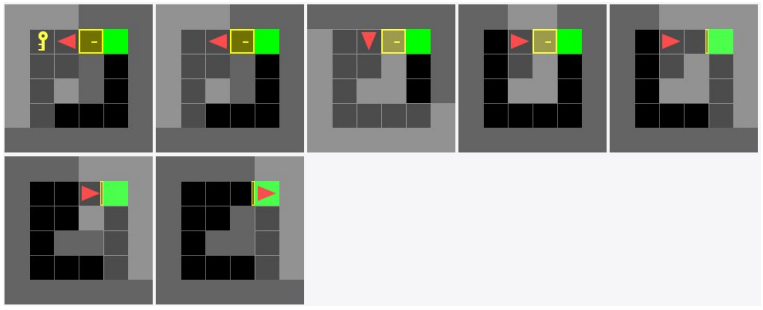

Figure 4: Known 6x6 Shortcut Results over time

Optimum Control Policy: [PK, TL, TL, UD, MF, MF]

4.1.5 Known 8x8 direct

Here we see the 8x8 direct case- the agent doesn't pick up the key and instead goes right for the door.

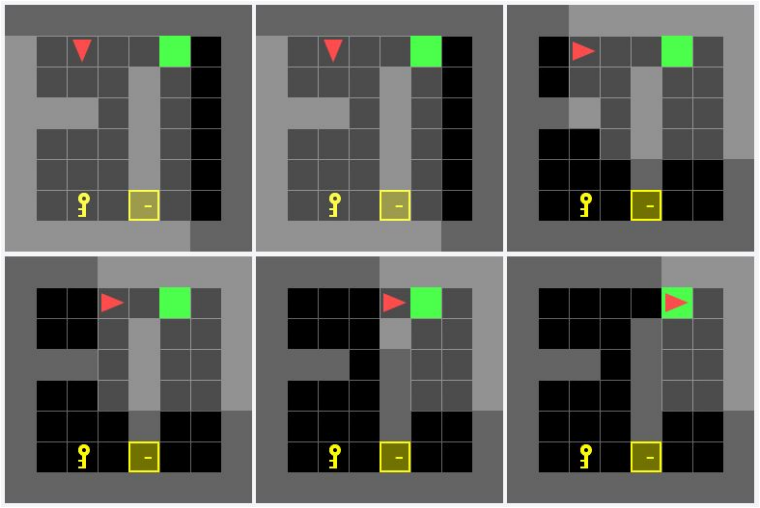

Figure 5: Known 8x8 Direct Results over time

Optimum Control Policy: [TL, MF, MF, MF]

4.1.6 Known 8x8 normal

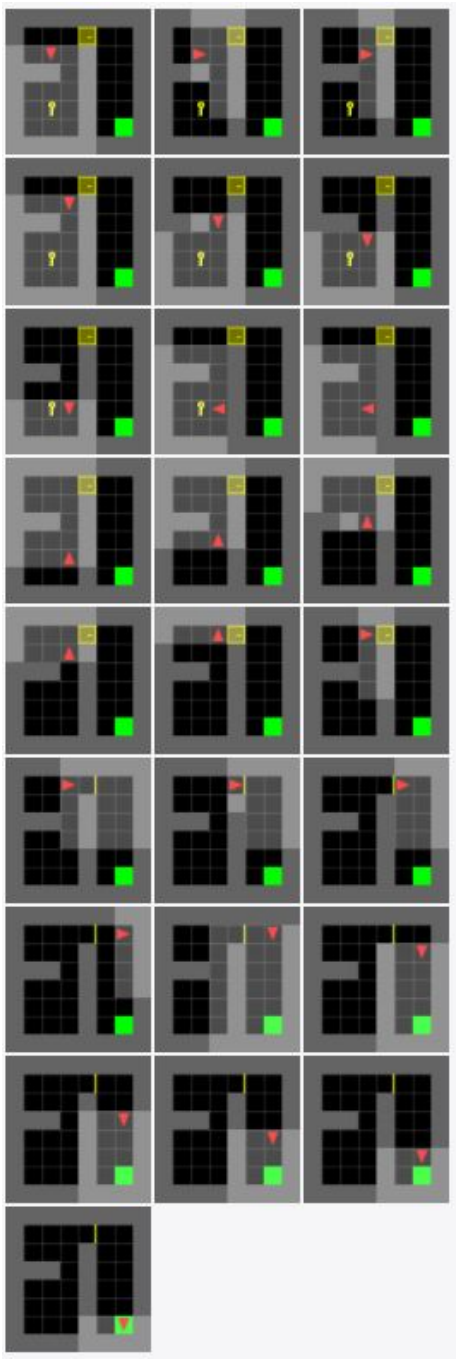

Figure 6: Known 8x8 Normal Results over time

Here we see a 8x8 normal case- the agent must pick up the key, unlock the door, before proceeding to the goal. Optimum Control Policy: [TL, MF, TR, MF, MF, MF, TR, PK, TR, MF, MF, MF, MF, TR, UD, MF, MF, MF, TR, MF, MF, MF, MF, MF]

4.1.7 Known 8x8 shortcut

Here we see a 8x8 shortcut case- the agent must pick up the key and unlock the door since it is closer than going all the way around.

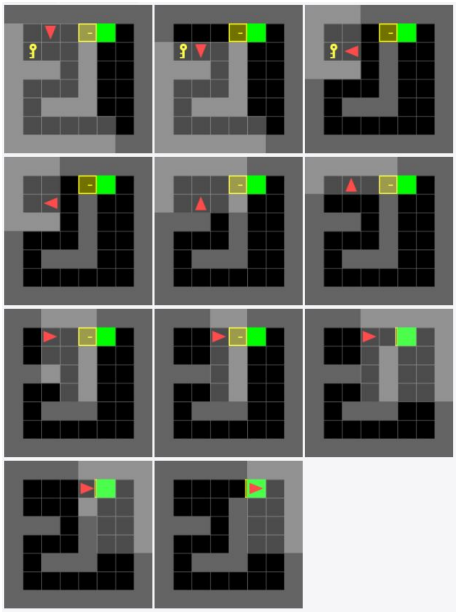

Figure 7: Known 8x8 Shortcut Results over time

Optimum Control Policy: [MF, TR, PK, TR, MF, TR, MF, UD, MF, MF]

4.2 Results - Random Map

In a random map environment, the features of the map are randomly placed according to a few rules. First, there is a vertical wall at column 4 with two doors. The doors can either be locked or open. The key is randomly located at one of three positions, and the goal is randomly located in one of three positions. Finally, the agent is initially spawned at the same position facing upwards. At the start of the program, an environment using the above stated rules will be randomly generated, and the Dynamic Programming Algorithm will attempt to solve it. Below are two examples of the random map results. Attached to the paper are more examples in GIF format.

4.2.1 Random Map 1

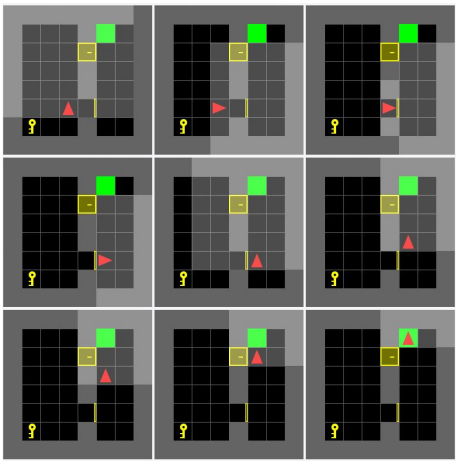

Figure 8: Random 8x8 Results over time

Optimum Control Policy: [TR,MF,MF,TL,MF,MF,MF,MF].
We see here that the agent chooses the shortest number of paths (i.e. skips getting the key and goes right through a pre-unlocked door).

4.2.2 Random Map 2

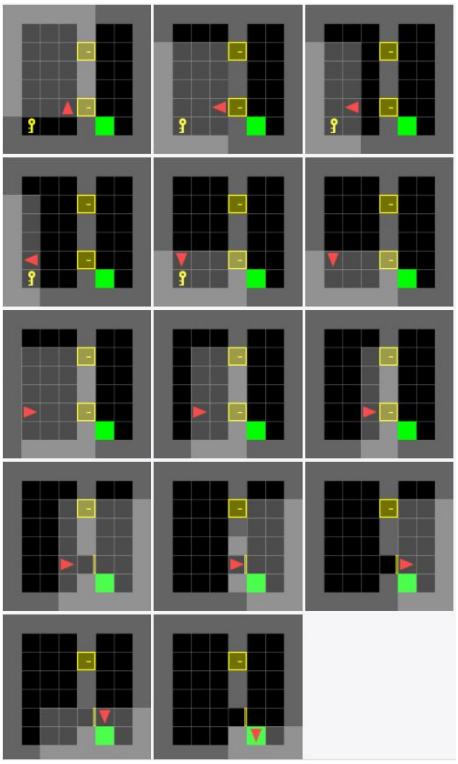

Figure 9: Random 8x8 Results over time

Optimum Control Policy: [TL,MF,MF,TL,PK,TL,MF,MF,UD,MF,MF,TR,MF].
Here the agent has no choice but to pick up the key. Once the key is found, the agent chooses the closest door and heads to the goal.

4.3 Discussion

In this project we explore the Dynamic Programming Algorithm for the door key problem. Overall, the algorithm is very successful in finding the path of least cost. We can improve on the production scalability of the code - the code is currently not generalizable to cases in which more than two doors are present. We can easily work our way around this by creating a state dictionary that grows based on the number of features there are. In addition, we can improve on the efficiency of the algorithm. Currently, we run at reasonable time (under 10 seconds). To make the algorithm more streamline, instead of storing the Q, V, and π matrices as pre-allocated matrices of size T, we can simply use a stacking function (like numpy's hstack/vstack/dstack) in order to tag on newly compute values of Q, V and π as they come along. This way, a T no longer needs to be specified, and we would not need a search for the index at which two consecutive row values of V are equivalent.