ECE 276B SP20 Project 1 Dynamic Programming Report

Pengcheng Cao[1]

Abstract— **This article is a class project report assigned by instructor Dr. Nikolay Atanastov of ECE 276B Planning and Controls in Robotics (Spring 2020). This report describes an application of the forward dynamic programming algorithm to solve a maze navigation problem with a defined control space U, corresponding cost space C, and state space χ.**

I. INTRODUCTION

In the relevant areas of robotics control and planning researches, the implementation of various types of motion planning or path planning algorithms have emerged in order to implement optimal control policies with minimal cost for robotic system to navigate throughout an environment. The navigation problem of a discretizable environment can be formulated as a deterministic shortest path problem (DSP), which is equivalent to deterministic finite-state (DFS) optimal control problem and is well-suited for dynamic programming algorithm as it can guarantee to find the optimal path with the minimal cost. Moreover, since the deterministic shortest path problem is symmetric, meaning a shortest path from start state s to terminal state tau is also a shortest path from tau to s. Thus, this paper selects the forward dynamic programming algorithm (forward DPA) to solve the DSP problem.

This paper presents an application of forward dynamic programming algorithm to find the optimal navigation sequence piece-wisely, first to find the 'key' object and pick up the 'key', then to find the 'door' and unlock it, and eventually to reach the goal position. The FDP algorithm is actually derived from dynamic programming algorithm. However, the main difference between forward and normal dynamic programming algorithm (DPA) is that the normal dynamic programming operates its iterations from the terminal planning horizon $t = T$ to the starting horizon $t = 0$, while forward DPA plans from the start state all the way till the terminal state.

II. PROBLEM FORMULATION

A. Deterministic Shortest Path

The so-called "**Door & Key**" environment, on which we will perform autonomous navigation, can be formulated as a 2-D static multi-stage deterministic shortest path problem. The overall state space, action space and corresponding cost space are given as follows:

*This work was not supported by any organization

[1]Pengcheng Cao is with Department of Mechanical and Aerospace Engineering of the University of California, San Diego, 9500 Gilman Dr, CA 92122, United States `p5cao at eng.ucsd.edu`

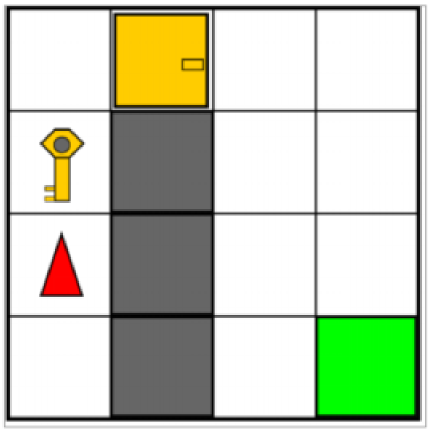


Fig. 1: Example plot of the Door & Key environment

State Space $\chi :=$ {all possible states $\mathbf{x} =$ [x, y, dir, is_carrying_key, is_unlocked]$|(x,y) \in$ {free positions}, $dir \in \{0,1,2,3\}$, is_carrying_key $\in$ {True, False}, is_unlocked $\in$ {True, False}}

Action Space $\mathscr{U} :=$ { move forward(MF), turn left(TL), turn right(TR), pick up key(PK), unlock door(UD)}

Cost Space $\mathscr{C} := \{1,1,1,0,0\}$

The terminal value function of each stage of the planned motion can be formulated as:

$$V_t^{\pi}(\mathbf{x}_t) = q(\mathbf{x}_T) + \sum_{t=0}^{T-1} l_t(\mathbf{x}_t, \pi_t(\mathbf{x}_t))$$

B. Motion Stages

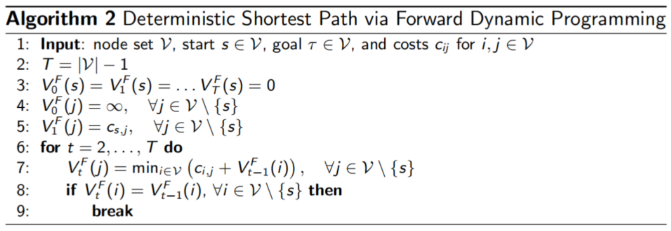
There are in total 3 stages till reaching the goal position. In the first stage, the agent will move towards and arrive at the one of adjacent grids of the key's position with its direction facing the key. The state space of stage 1 includes all the states in the non-'Wall'ed grids between positions of the start state and ready-to-pick-up-key state that permits the most cost-saving motion while having as few states as possible for the sake of the efficiency of the forward DP algorithm. Each stage cost is 1, action space = [MF,TL,TR], and the terminal cost is 0 from the action of picking up the key.

After the key is picked up, the agent will move towards and arrive at the adjacent grid of the door's position with its direction facing the door. The state space of stage 2 includes all the states in the non-'Wall'ed grids between positions of the picking-up-key state and ready-to-unlock-door state that permits the most cost-saving motion while having as few states as possible for the sake of the efficiency of the forward DP algorithm. Each stage cost is 1, action space = [MF,TL,TR], and the terminal cost is 0 from the action of unlocking the door.

At last, the agent will plan a path towards the goal position. The state space of stage 3 includes all the states in the non-'Wall'ed grids between positions of the unlocking-door state and ready-to-reach-goal state that permits the most cost-saving motion while having as few states as possible for the sake of the efficiency of the forward DP algorithm. Each stage cost is 1, action space = [MF,TL,TR], and the terminal cost is 1 from the action of moving forward into the goal position.

III. TECHNICAL APPROACHES

A. *Forward Dynamic Programming Algorithm*



Algorithm 2 Deterministic Shortest Path via Forward Dynamic Programming

```
1: Input: node set $\mathcal{V}$, start $s \in \mathcal{V}$, goal $\tau \in \mathcal{V}$, and costs $c_{ij}$ for $i, j \in \mathcal{V}$
2: $T = |\mathcal{V}| - 1$
3: $V_0^F(s) = V_1^F(s) = \dots V_T^F(s) = 0$
4: $V_0^F(j) = \infty, \quad \forall j \in \mathcal{V} \setminus \{s\}$
5: $V_1^F(j) = c_{s,j}, \quad \forall j \in \mathcal{V} \setminus \{s\}$
6: for $t = 2, \dots, T$ do
7:     $V_t^F(j) = \min_{i \in \mathcal{V}} \left(c_{i,j} + V_{t-1}^F(i)\right), \quad \forall j \in \mathcal{V} \setminus \{s\}$
8:     if $V_t^F(i) = V_{t-1}^F(i), \forall i \in \mathcal{V} \setminus \{s\}$ then
9:         break
```

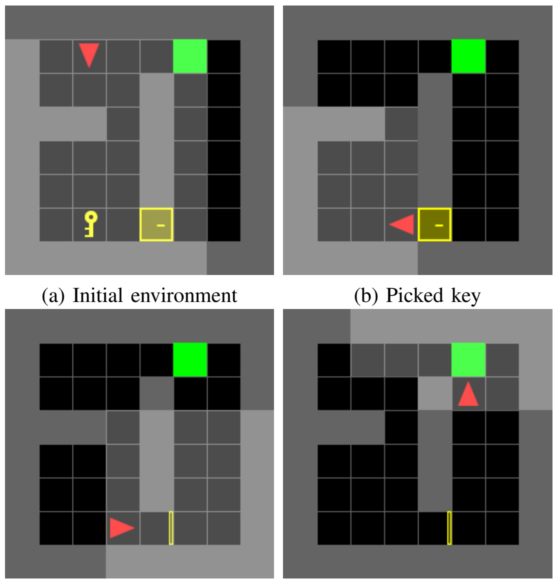
Fig. 2: Forward DP Algorithm

B. *Application to the Door&Key Problem*

As was mentioned earlier, the forward dynamic programming algorithm differs from DPA since it marches its plan step from 0 to the planning horizon T. In this problem, the node set $\mathcal{V}$ is the state space of the planning problem that can be rendered by a construct_state_space() function, s and τ are initial and terminal states, respectively, and each stage cost $c_{ij} = 1$. A calc_value_func() function can return the preliminary value functions and corresponding action sequence, i.e. control policy associated with the arrival at each possible intermediate state and the reachable goal states (these goal states must be in the constructed state space and the values at these states will be examined later after finishing the updates of the value functions to find the optimal goal state and return the corresponding optimal policy). After setting up the values in first time step 0 and 1, the values at all states will be iterated and examined til no value can be updated between 2 consecutive time steps. This generally means all current values in state space are minimal and thus the minimal value at goal state and optimal action sequence can be returned.

IV. Results and Conclusions

A. *Goal state examinations*



(a) Initial environment (b) Picked key

(c) Unlocked door (d) Reaching goal

Fig. 3: Environment doorkey-8x8-direct

In the Door&Key navigation problem, a state space can contain several possible goal states in the vicinity of the object position. For example, as shown in the 8x8 normal environment of Fig.3, the agent is able to arrive at the goal states [2, 5, 1, False, False] and [3, 6, 2, False, False] before picking up the key, and [4, 1, 0, True, True] and [5, 2, 3, True, True] before reaching the goal states. All these states are in the state spaces of their own planning stages. We will, however, to select only one on the goal states as the outcome of the optimal control sequence to guarantee the smallest possible costs, which is equivalent to minimal action steps in this DSP problem.

After the investigation of goal states values after the planning is finished, we find that Value([3, 6, 2, False, False]) = 9 and Value([2, 5, 1, False, False]) = 10 such that Value([3, 6, 2, False, False]) is less than Value([2, 5, 1, False, False]), so the optimal goal state is [3, 6, 2, False, False] with value 9 and optimal policy [TL, MF, TR, MF, MF, MF, MF, MF, TR, PK].

Similarly, we can identify the only optimal goal state is [5, 2, 3, True, True] by examining the values of both states. The optimal value is sum of stage costs 7 plus the terminal cost 1 of MF, which gives 8. And the optimal stage policy after the door is unlocked is [MF, MF, TL, MF, MF, MF, MF, MF].

B. *Value updates*

When iterating through the time steps of the planning horizon, the values associated with each state shall be updated till every state has reached the minimal value. In our algorithm, we set to break the loop when the values of current time step t is equal to that of former time step

t. Take Environment doorkey-8x8-direct as an example, the goal state value updates over each time step are given as follows.

Value updates of goal state [3, 6, 2, False, False] when planning to key, planning stops at time step t = 5:

$$[\infty, \infty, 9, 9, 9, 9]$$

Value updates of goal state [3, 6, 0, True, False] when planning to door, planning stops at time step t = 3:

$$[\infty, \infty, 2, 2]$$

Value updates of goal state [5, 2, 3, True, True] when planning to goal position(treasure), planning stops at time step t = 8:

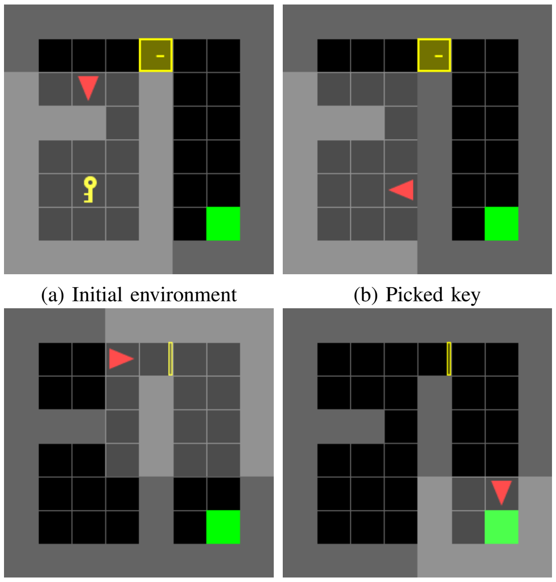
$$[\infty, \infty, \infty, \infty, \infty, \infty, 7, 7, 7]$$

C. *Overall optimal policy*

After all planning is done, the overall action sequence withe minimal costs/step number to navigate through the Door&Key environment is given as follows. The overall optimal policy takes 23 steps to reach the treasure/green block on the map, including the actions of moving forward, turning left or right, picking up the key and unlocking the door. The final cost of this sequence is 21.

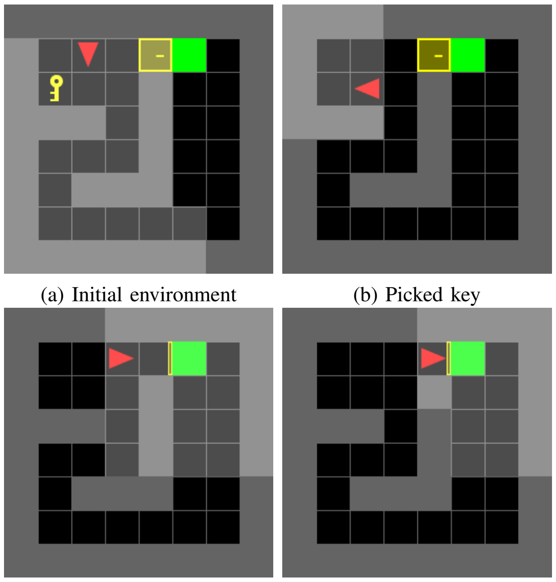
$$\begin{aligned}\pi^*_{env} = [&TL \Rightarrow MF \Rightarrow TR \Rightarrow MF \Rightarrow MF \Rightarrow MF \Rightarrow MF \Rightarrow MF \\ &\Rightarrow TR \Rightarrow PK \Rightarrow TL \Rightarrow TL \Rightarrow UD \Rightarrow MF \Rightarrow MF \Rightarrow TL \\ &\Rightarrow MF \Rightarrow MF \Rightarrow MF \Rightarrow MF \Rightarrow MF]\end{aligned}$$

V. Appendix


(a) Initial environment (b) Picked key

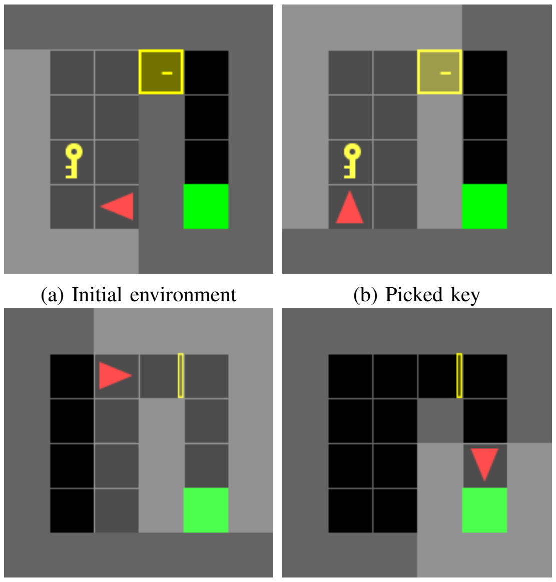
(c) Unlocked door (d) Reaching goal

Fig. 4: Environment doorkey-8x8-normal, step count = 23, cost = 21


(a) Initial environment (b) Picked key

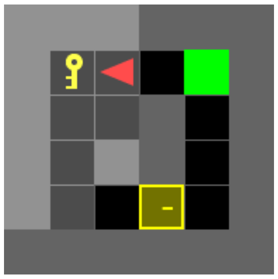
(c) Unlocked door (d) Reaching goal

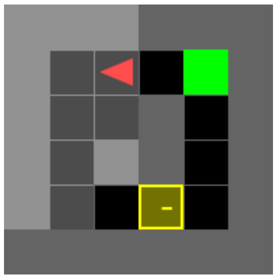
Fig. 5: Environment doorkey-8x8-shortcut, step count = 9, cost = 7

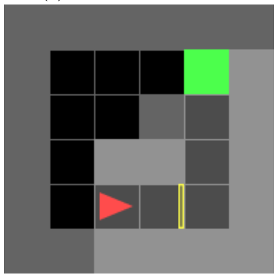

(a) Initial environment (b) Picked key

(c) Unlocked door (d) Reaching goal

Fig. 6: Environment doorkey-6x6-normal, step count = 14, cost = 12


(a) Initial environment


(b) Picked key


(c) Unlocked door

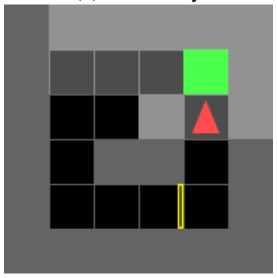

(d) Reaching goal

Fig. 7: Environment doorkey-6x6-direct, step count = 14, cost = 12

doorkey-6x6-shortcut_init.png

doorkey-6x6-shortcut_picked_k

(a) Initial environment

(b) Picked key

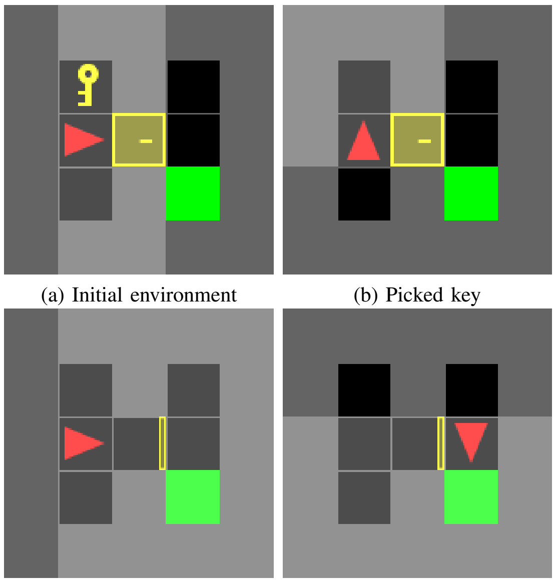
doorkey-6x6-shortcut_unlocked_door.png

doorkey-6x6-shortcut_reaching

(c) Unlocked door

(d) Reaching goal

Fig. 8: Environment doorkey-6x6-sc, step count = 5, cost = 3


(a) Initial environment (b) Picked key

(c) Unlocked door (d) Reaching goal

Fig. 9: Environment doorkey-6x6-direct, step count = 14, cost = 12