Autonomous Navigation to The Goal In Grid: Dynamic Programming

Zhuoqun Chen
Dept. Electrical and Computer Engineering
UC San Diego
La Jolla, USA
zhc057@ucsd.edu

***Abstract*—This report focuses on solving a discrete autonomous navigation problem in the MiniGrid Environment settings. By formulating the problem as a Deterministic Optimal Control(DOC) Problem, we solves it with Dynamic Programming(DP) Algorithm. To show the effectiveness of DP algorithm, we tested it in both known environments(a collection of 5x5, 6x6 and 8x8 grids with different settings) and random environments of 8x8 grid size where the key and the goal's position can be randomly sampled from a finite discrete set of tuples. We presented the experiment results and conclude the report in the end.**

***Index Terms*—Deterministic Optimal Control(DOC) , Deterministic Shortest Path(DSP), Dynamic Programming(DP), MiniGrid Environment, OpenAI Gym**

I. Introduction

Deterministic Optimal Control (DOC) techniques can be effectively applied in various robotics scenarios to achieve optimal performance and decision-making. For an instance, DOC related algorithms can be utilized in autonomous navigation systems to enable robots to plan optimal paths and make informed decisions. By modeling the environment and robot dynamics, the optimal control problem can be formulated to find the trajectory that minimizes a cost function, such as travel time or energy consumption, while avoiding obstacles and adhering to motion constraints. This allows robots to navigate efficiently in complex environments.

When the world environment can be viewed as discrete where the agent state space and control space are discrete and finite, DOC Problem can be solved by Dynamic Programming algorithms[1], for example see fig. 1. Under the same settings, some researchers also construct the world as a graph and formulate the problem as a Deterministic Shortest Path Problem. The equivalence of these two problems can be proved by constructing a graph representation of the DOC Problem.

In this report, we mainly focus on how to find a path with minimum cost for agents that lives in a grid world environment by leveraging Dynamic Programming algorithm, where the agent's movement is bounded in a finite size of grid with unreachable obstacles around. The agent can take several discrete actions(control input) to navigate in the grid to the final goal with finite horizon. The robot can unlock the door first while holding a key in his hand if the locked door is in the path that stops him to the final goal position. We organize the report in the following orders: In the first section, we give a brief introduction and we then formulate the problem in the second section. Then we present our technical approach in the third section and finally showed our experiment results, analysis and conclusion in the following sections.

II. Problem Formulation

Our door-key problem is based on the Deterministic Markov Decision Process(MDP) assumption with discrete and finite state space so we firstly introduce MDP.

A. Markov Decision Process

Markov Decision Process(MDP) with controlled tuple is defined as an ordered tuple $(\mathcal{X}, \mathcal{U}, x_0, f, T, \ell, \mathfrak{q}, \gamma)$, where $\mathcal{X}$ is a discrete state

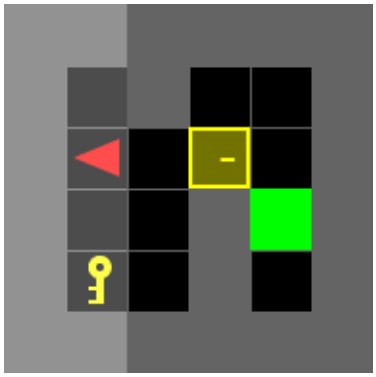

Fig. 1. Known DoorKey 6x6 Normal Environment Example: The agent(red triangle) desires to find a path to a known goal position(green square). The Black cells are traversable and free, so agent can go to that cell given action labeled 'Move Forward'. And the grey cells are the walls that the agent can't go into. If the door in front of the agent is locked, the agent need to unlock it first with the key. The goal for the agent is to find a path with the lowest cost to get to the goal.

space, $\mathcal{U}$ is a discrete control space, x_0 is an initial state defined on $\mathcal{X}$, f is a deterministic motion model or transition function defined on $\mathcal{X}$ that given $x_t \in \mathcal{X}$, $u_t \in \mathcal{U}$, x_{t+1} is given by $x_{t+1} = f(x_t, u_t)$.

B. Deterministic Optimal Control(DOC) Problem

With the above MDP assumption, we define the Deterministic Optimal Control(DOC) Problem as the following.

Given $x_0 \in \mathcal{X}$, our goal is to construct an optimal control sequence $u_{0:T-1}$ such that:

$$\begin{array}{ll} \min_{\mathbf{u}_{0:T-1}} & \mathfrak{q}\left(\mathbf{x}_T\right) + \sum_{t=0}^{T-1} \ell\left(\mathbf{x}_t, \mathbf{u}_t\right) \\ \text{s.t.} & \mathbf{x}_{t+1} = f\left(\mathbf{x}_t, \mathbf{u}_t\right), t = 0, \ldots, T-1 \\ & \mathbf{x}_t \in \mathcal{X}, \mathbf{u}_t \in \mathcal{U}\left(\mathbf{x}_t\right), \end{array}$$

There are many algorithms that can be designed to solve DOC problem and we will discuss about solving this problem with Dynamic Programming algorithm in the next section.[2]

III. Technical Approach

A. State Space Design

We define the state space $\mathcal{X}$ to be a Cartesian product of some observable sets in the grid world settings:

$$\begin{aligned} \mathbf{x} = & \\ & \text{agent position} \times \\ & \text{agent orientation} \times \\ & \text{agent holding key indicator} \times \\ & \text{door(s) status indicator} \times \\ & \text{key position(optional)} \times \\ & \text{goal position(optional)} \end{aligned} \tag{1}$$

1) State Space for Part A: Based on eq. (1), in Part A, we don't need to provide those two optional parts as the state of the agent in the grid because we are allowed to run the algorithm multiple times so such information can be ruled out from our state design. And we implement above in our implementation by grouping all the info into a dictionary variable. We represent the key status as that current agent is with key(**w**) or without key(**wo**) and the door status as that the door in the environment is open(**o**) or closed(**c**). We also encode the orientation of the agent to be one of the following status: **up**, **down**, **left**, **right**

2) State Space for Part B: As Part B requirement points out, the main difference of this part is to compute a single policy that can solve all 36 environment variations. So we need to add the optional part in eq. (1) of the state space design. Later we also need to fine-tune the motion model logic to encode the environment-specific random observations into the agent state.

According to the requirement, we already know all the layouts of the grids except that the positions of the keys and the goals are randomly selected in set $\{(1,1),(2,3),(1,6)\}$ and set $\{(5,1),(6,3),(5,6)\}$, respectively. So on top of the design in Part A, now we also incorporate this two pieces of information into the agent state. It is also reasonable to enlarge the cardinality of the dimension the door status observation because there are two doors now in the environment so the cardinality of the door status indicator will be 4:

$$\text{door status indicator} = \{\text{o-c}, \text{c-o}, \text{o-o}, \text{c-c}\} \tag{2}$$

where **c** encodes *closed* and **o** encodes *open*.

B. Stage Cost and Terminal Cost Design

We designed a simple yet effetient stage cost function and terminal cost function to help our agent to find a path with minimum path in the grid world settings:

stage cost is designed as:

$$l(\mathbf{x}, \mathbf{u}) := \begin{cases} 0 & \mathbf{x}_{\text{pos}} = \text{goal position} \\ 1 & \text{otherwise} \end{cases} \tag{3}$$

stage cost is designed as:

$$q(\mathbf{x_T}) := \begin{cases} 0 & \mathbf{x}_{\text{pos}} = \text{goal position} \\ \infty & \text{otherwise} \end{cases} \tag{4}$$

C. Time Horizon Design

We design the time horizon to be:

$$T = |\mathcal{X}| - 1 \tag{5}$$

D. Discounted Factor Design

For finite horizon steps, we design the discounted factor $\gamma = 1$.

E. Motion Model(Transition Function)

A key design for applying DP algorithm in the door-key problem settings is to define the motion model. For most of the case, the agent's state old state won't transition to new state, i.e., the state remains the same for most of the case, for example in the position part in the state, if the agent's position is inside the cell where the wall is, then such state is an absorption state that will never be transition to other state. That's why in the implementation, the motion model for most cases only consider special cases and whether such conditions have been met.

And the transition function is also the biggest difference between Part A and Part B. In Part A, we only need to consider some low dimensional cases, for example, when the agent is holding the key action and the action is "UD", if at that time there's a door in front of itself, then the agent only need to judge whether the door is open and if it's locked, then the door's status will transit to open and that's it. However, in Part B there are 2 doors' status, and each one can be open or closed, in this case, when encoutering the same situation, the agent needs to decide to update which door's status when holding the key in front of a locked door, which will largely expands the dimension of the search space.

In summary, we just enumerate all the possible situations where the designed state may transition, the leave the other states to be unchanged, this is how we defined our motion model f. And this is one advantage of leveraging Dynamic Programming in the grid world because we have all the observations and we know every possible combination about how old states will transition to new states(perfect privileged information about the world model).

F. Dynamic Programming algorithm used for solving DOC

We have all the components in our MDP, know we can leverage Dynamic Programming to solve our door-key grid environment.

Algorithm 1 Dynamic Programming on DOC

Input: MDP $(\mathcal{X}, \mathcal{U}, p_0, p_f, T, \ell, \mathfrak{q}, \gamma)$
Input: determined transition $x_{t+1} = f(x_t, u_t)$

$\gamma = 1$
$|\mathcal{X}| < \infty$ and $|\mathcal{U}| < \infty$
$V_T(\mathbf{x}) = \mathfrak{q}(\mathbf{x}), \quad \forall \mathbf{x} \in \mathcal{X}$
for $t = T - 1$ to 0 **do**
  $Q_t(\mathbf{x}, \mathbf{u}) = \ell(\mathbf{x}, \mathbf{u}) + \gamma V_{t+1}\left(f(\mathbf{x}, \mathbf{u})\right)$
  $V_t(\mathbf{x}) = \min_{\mathbf{u} \in \mathcal{U}(\mathbf{x})} Q_t(\mathbf{x}, \mathbf{u}), \quad \forall \mathbf{x} \in \mathcal{X}$
  $\pi_t(\mathbf{x}) = \arg\min_{\mathbf{u} \in \mathcal{U}(\mathbf{x})} Q_t(\mathbf{x}, \mathbf{u}), \quad \forall \mathbf{x} \in \mathcal{X}$
end for
return policy $\pi_{0:T-1}$ and value function V_0

In practice, we also set early termination condition for certain time step where the value function keeps unchanged for all the states.

By using above designed Dynamic Programming, we can solve the door-key grid problem.

G. The equivalence with Deterministic Shortest Path Problem(DSP)

Based on different formulation, the deterministics optimal control problem can also be solved by firstly reformulating it to be a deterministic shortest path problem on a graph can then leverage other graph-search based algorithm to solve it. Because

in this report we mainly focus on how to leverage Dynamic Programming algorithm by formulating the DOC problem, so we won't discuss too much about this equivalence, more proofs and details can be found in Professor's lectures.

IV. Experiment Results

In the above section, we give our dynamic programming algorithm as section III-F. In this section, we present the effectiveness of solving the Grid World problem by using Dynamic Programming.

Specifically, we tested the DP algorithm in a MiniGrid Environment that implements the OpenAI's gym API. And in our experiment, we didn't directly use the internal implementation, instead, we left the observation term untouched and defined our own motion model and the associated stage cost of each step.

Here are the results for using Dynamic Programming Algorithm both in known environments and in random environments.

A. Part A

1) doorkey-5x5-normal.env: The results for solving doorkey-5x5-normal can be seen from fig. 2

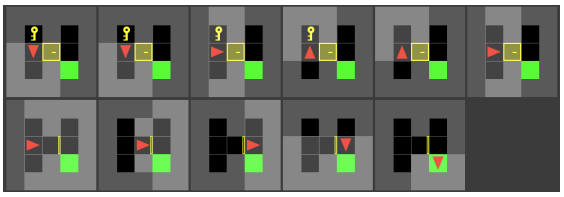

Fig. 2. doorkey-5x5-normal

The optimal control sequency is: ['TL', 'TL', 'PK', 'TR', 'UD', 'MF', 'MF', 'TR', 'MF']

2) doorkey-6x6-normal.env: The results for solving doorkey-6x6-normal can be seen from fig. 3

The optimal control sequency is: ['TL', 'MF', 'PK', 'TL', 'MF', 'TR', 'UD', 'MF', 'MF', 'TR', 'MF']

3) doorkey-6x6-direct.env: The results for solving doorkey-6x6-direct can be seen from fig. 4

The optimal control sequency is: ['MF', 'MF', 'TR', 'MF', 'MF']

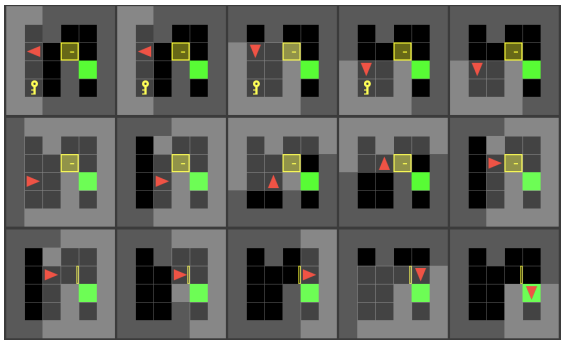

Fig. 3. doorkey-6x6-normal

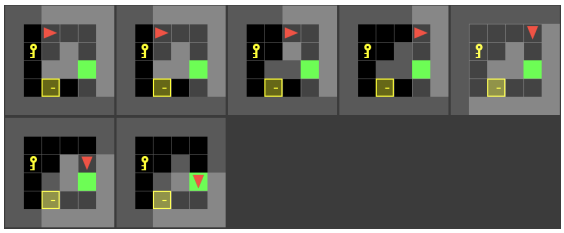

Fig. 4. doorkey-6x6-direct

4) doorkey-6x6-shortcut.env: The results for solving doorkey-6x6-shortcut can be seen from fig. 5

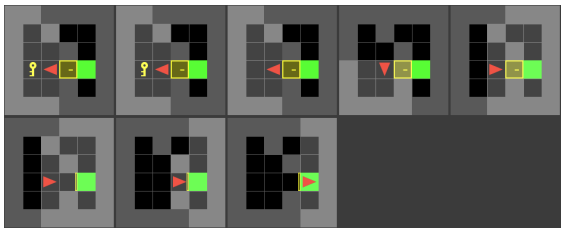

Fig. 5. doorkey-6x6-shortcut

The optimal control sequency is: ['PK', 'TL', 'TL', 'UD', 'MF', 'MF']

5) doorkey-8x8-normal.env: The results for solving doorkey-8x8-normal can be seen from fig. 6

The optimal control sequency is: ['TR', 'MF', 'TL', 'MF', 'TR', 'MF', 'MF', 'MF', 'PK', 'TL', 'TL', 'MF', 'MF', 'MF', 'TR', 'UD', 'MF', 'MF', 'MF','TR', 'MF', 'MF', 'MF'].

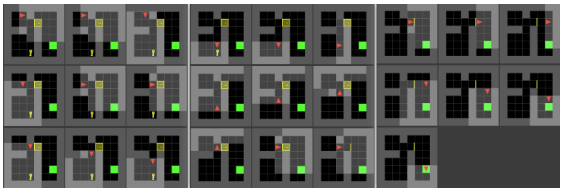

Fig. 6. doorkey-8x8-normal

6) doorkey-8x8-direct.env: The results for solving doorkey-8x8-direct can be seen from fig. 7

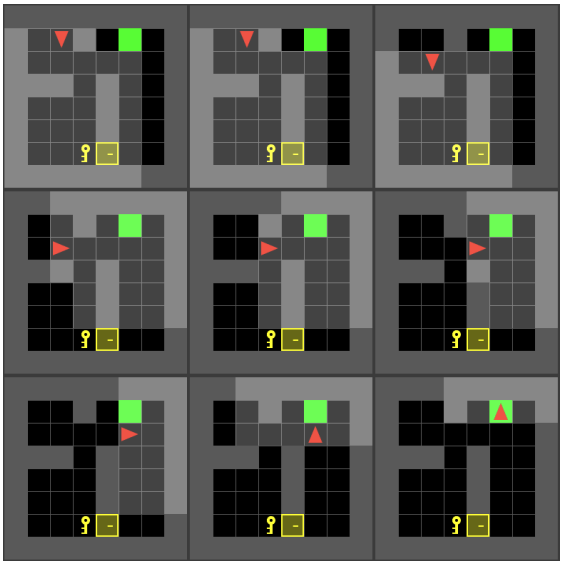

Fig. 7. doorkey-8x8-direct

The optimal control sequency is: ['MF', 'TL', 'MF', 'MF', 'MF', 'TL', 'MF']

7) doorkey-8x8-shortcut.env: The results for solving doorkey-8x8-shortcut can be seen from fig. 8.

The optimal control sequency is: ['TR', 'MF', 'TR', 'PK', 'TL', 'UD', 'MF', 'MF'].

B. Part B

In Part B's setting, the environment is randomly generated from all 36 possible environments, so we randomly initiated the slover for several times and collected the optimal control sequencies along their according trajectories. And the other observation is

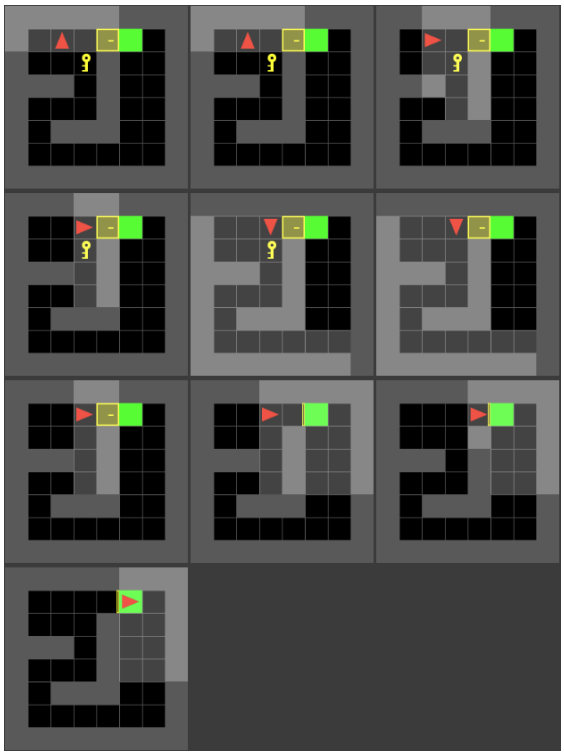

Fig. 8. doorkey-8x8-shortcut

that all the grid size is 8x8 and there are only a few wall cells in the fixed positions, so the overall search time is way much longer than in Part A.

1) Trial 1: In Trial 1, DoorKey-8x8-23 is selected as the environment.

The randomly generated key position is at $(2, 3)$, and the goal position is at $(5, 6)$.

It took 7.33s to get to the optimal policy.

The optimal control sequency is: ['TR', 'MF', 'MF', 'TR', 'MF'], , see fig. 9.

The according gif result is presented in the attached results folder.

2) Trial 2: In Trial 2, DoorKey-8x8-18 is selected as the environment.

The randomly generated key position is at $(2, 3)$, and the goal position is at $(6, 3)$.

It took 7.10s to get to the optimal policy.

The optimal control sequency is: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'MF', 'TR', 'MF'], see

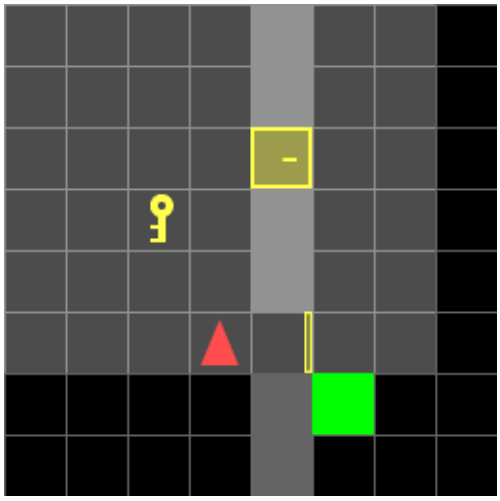

Fig. 9. Trial 1: DoorKey-8x8-23

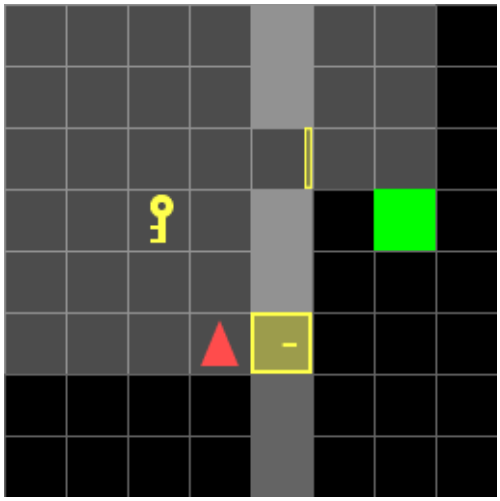

Fig. 10. Trial 2: DoorKey-8x8-18

fig. 10.

The according gif result is presented in the attached results folder.

3) Trial 3: In Trial 3, DoorKey-8x8-16 is selected as the environment.

The randomly generated key position is at $(2, 3)$, and the goal position is at $(5, 1)$.

It took 8.57s to get to the optimal policy.

The optimal control sequency is: ['MF', 'MF', 'TL', 'PK', 'TR', 'MF', 'TR', 'UD', 'MF', 'MF', 'TL', 'MF'], see fig. 11.

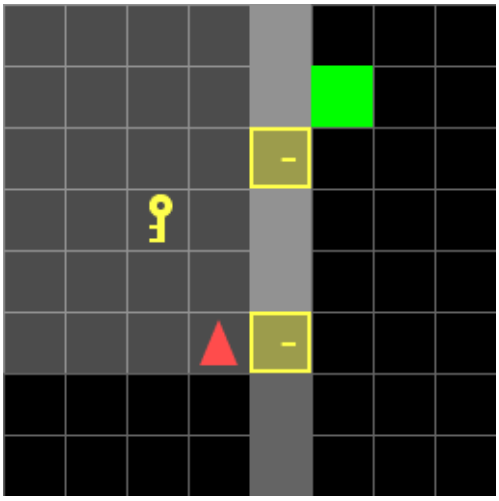

Fig. 11. Trial 3: DoorKey-8x8-16

The according gif result is presented in the attached results folder.

4) Trial 4: In Trial 4, DoorKey-8x8-13 is selected as the environment.

The randomly generated key position is at $(2, 3)$, and the goal position is at $(5, 1)$.

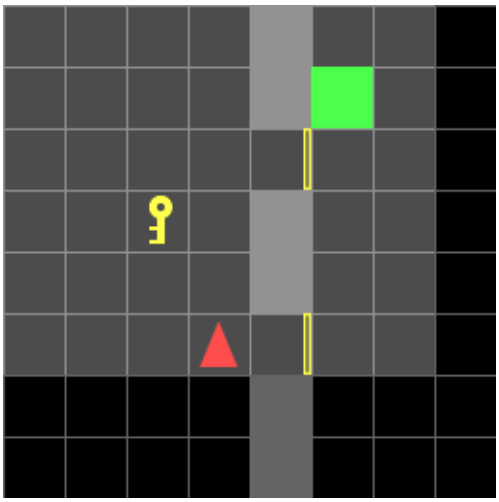

Fig. 12. Trial 4: DoorKey-8x8-13

It took 6.6s to get to the optimal policy.

The optimal control sequency is: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TL', 'MF'], see fig. 12.

The according gif result is presented in the attached results folder.

5) Trial 5: In Trial 5, DoorKey-8x8-26 is selected as the environment.

The randomly generated key position is at $(1, 6)$, and the goal position is at $(5, 1)$.

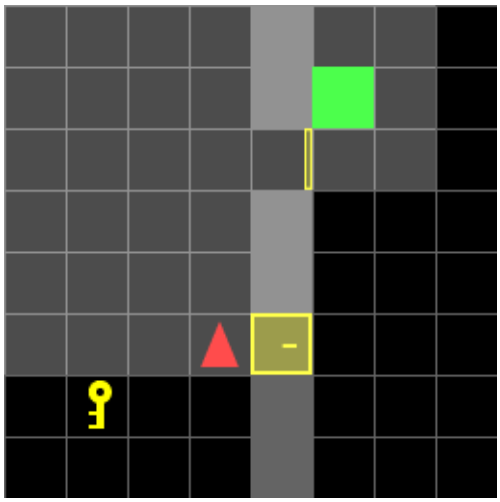

Fig. 13. Trial 5: DoorKey-8x8-26

It took 7.04s to get to the optimal policy. The optimal control sequency is: ['MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'TL', 'MF'], see fig. 13. The according gif result is presented in the attached results folder.

ACKNOWLEDGMENT

Thanks Prof. Nicolay for such meaningful lectures and the commitment of TA Zhirui Dai.

REFERENCES

[1] D. P. Bertsekas, *Dynamic Programming and Optimal Control*, 3rd. Belmont, MA, USA: Athena Scientific, 2005, vol. I.

[2] N. Atanasov, *Lecture pdfs and course website resources*, 2023.