Dynamic Programming for DoorKey Navigation

Jiajun Li
Department of ECE, UCSD
Email: jil186@ucsd.edu

***Abstract*—This paper presents a Dynamic Programming (DP) approach for solving the DoorKey navigation problem in Mini-Grid environments. We formulate the task as a Markov Decision Process (MDP) and implement value iteration to compute optimal policies for both known and randomized map configurations. Our solution handles key-door interactions, directional constraints, and partial observability through state-space augmentation. Experimental results demonstrate successful navigation in 7 predefined environments and 36 random configurations, with trajectories visualized through automatically generated GIFs. The approach achieves complete observability by extending the state representation to track key possession and door statuses.**

I. Introduction

Autonomous navigation in environments with interactive objects, such as doors requiring keys, is a fundamental capability for robots operating in real-world settings like search and rescue or warehouse logistics. The DoorKey task from MiniGrid provides a compact yet challenging benchmark to study these behaviors under partial observability and sequential decision-making.

We explore two settings: one with known map layouts and another with randomized environments. Our method uses dynamic programming with a factored state representation, enabling efficient planning despite the high-dimensional and partially observable nature of the problem. By exploiting structural properties of the environment, we demonstrate that dynamic programming can solve small-scale navigation tasks optimally and tractably.

II. Problem Formulation

A. MDP Definition (Part a)

The navigation task is formalized as a finite MDP $(\mathcal{S}, \mathcal{A}, P, C)$ where:

- **States** $s = (x, y, d, k, l)$: Agent position (x, y), direction $d \in \{0°, 90°, 180°, 270°\}$, key possession $k \in \{0, 1\}$, and door lock status $l \in \{0, 1\}$
- **Actions** $\mathcal{A} = \{\text{MF}, \text{TL}, \text{TR}, \text{PK}, \text{UD}\}$
- **Transition Model** $P(s'|s, a)$: Deterministic grid dynamics with directional constraints
- **Cost Function** $C(s, a) = 1 + \infty \cdot \mathbb{I}_{\text{invalid}(a)}$

B. MDP Definition (Part b)

Part b has a similar MDP definition as Part a, but with some additional state elements. Since we have a random environment which has uncertain key position, goal position, and door status, we need to take these conditions into our state vector to cover all state possibilities.

- **State Vector**:

$$s = (x, y, \theta, k, l_1, l_2, \kappa, \gamma) \in \mathcal{S}_{\text{rand}} \tag{1}$$

  - $x, y \in \{0, \dots, 9\}$: Agent grid coordinates
  - $\theta \in \{0°, 90°, 180°, 270°\}$: Heading direction
  - $k \in \{0, 1\}$: Key possession status
  - $l_1, l_2 \in \{0, 1\}$: Door lock states at (5,3) and (5,7)
  - $\kappa \in \{0, 1, 2\}$: Key position index (predefined locations)
  - $\gamma \in \{0, 1, 2\}$: Goal position index (predefined locations)

C. Time Horizon Definition

For known map configurations, the planning horizon T is determined by the grid dimensions and agent orientation:

$$T = W \times H \times D \tag{2}$$

where:

- W: Grid width (number of columns)
- H: Grid height (number of rows)
- D: Cardinal directions (up, right, down, left)

This upper bound guarantees coverage of all possible agent configurations.

D. Value Iteration Formulation

We solve the Bellman optimality equation through value iteration:

$$V_t(s) = \min_{a \in \mathcal{A}} \left[C(s, a) + \gamma \sum_{s'} P(s'|s, a) V_{t+1}(s') \right] \tag{3}$$

For deterministic transitions, this simplifies to:

$$V_t(s) = \min_{a \in \mathcal{A}} \left[1 + V_{t+1}(f(s, a)) \right] \tag{4}$$

where $f(s, a)$ is the deterministic transition function.

The value iteration algorithm produces two key outputs:

$$\begin{aligned} V^*(s) &: \text{Optimal value function (minimum cost)} \\ \pi^*(s) &: \text{Optimal policy } \pi^* : \mathcal{S} \to \mathcal{A} \end{aligned} \tag{5}$$

III. Technical Approach

A. State Space (Part a)

The joint state space is decomposed into independent components:

$$|\mathcal{S}| = |\mathcal{X}| \times |\mathcal{D}| \times |\mathcal{K}| \times |\mathcal{L}| \tag{6}$$

where:

- $\mathcal{X} \subset \mathbb{Z}^2$: Grid positions
- $\mathcal{D} = \{0, 1, 2, 3\}$: Cardinal directions
- $\mathcal{K} = \{0, 1\}$: Key possession status
- $\mathcal{L} = \{0, 1\}$: Door lock status

B. State Transition Dynamics

Agent motion follows deterministic grid rules:

$$s_{t+1} = \begin{cases} (x', y', d, k, l) & \text{if } a_t = \text{MF and valid} \\ (x, y, (d-1)\%4, k, l) & \text{if } a_t = \text{TL} \\ (x, y, (d+1)\%4, k, l) & \text{if } a_t = \text{TR} \\ (x, y, d, 1, l) & \text{if } a_t = \text{PK and valid} \\ (x, y, d, k, 0) & \text{if } a_t = \text{UD and valid} \end{cases} \tag{7}$$

C. Algorithm Implementation (Part a)

Algorithm 1 DoorKey Value Iteration

```
1: Initialize $V_T(s) = 0$ for goal states, $\infty$ otherwise
2: for $t = (T-1) \dots 0$ do
3:     for all states $s \in \mathcal{S}$ do
4:         for all actions $a \in \mathcal{A}$ do
5:             Compute next state $s' = f(s_t, a_t)$
6:             Update Q-value: $Q(s_t, a_t) = 1 + V_{t+1}(s')$
7:         end for
8:         $V_{\text{new}}(s) = \min_a Q(s, a)$
9:         $a_{\text{new}}(s) = \min_a Q(s, a)$
10:    end for
11: end for
```

D. Random Map Generalization

For part B, we extend the state space to include:

$$s_{\text{rand}} = (x, y, d, k, l_1, l_2, k_{\text{idx}}, g_{\text{idx}}) \tag{8}$$

where l_1/l_2 are door statuses and $k_{\text{idx}}/g_{\text{idx}}$ encode key/goal positions from predefined sets $\mathcal{K} = \{(2, 2), (2, 3), (1, 6)\}$, $\mathcal{G} = \{(6, 1), (7, 3), (6, 6)\}$.

E. State Space (Part b)

For randomized 10x10 environments, the extended state space accounts for variable key/goal positions and dual door interactions:

$$|\mathcal{S}| = |\mathcal{X}| \times |\mathcal{D}| \times |\mathcal{K}| \times |\mathcal{L}_1| \times |\mathcal{L}_2| \times |\mathcal{K}_{\text{idx}}| \times |\mathcal{G}_{\text{idx}}| \tag{9}$$

where:

- $\mathcal{X} \subset \mathbb{Z}^2$: Agent positions in 10x10 grid
- $\mathcal{D} = \{0, 1, 2, 3\}$: Cardinal directions
- $\mathcal{K} = \{0, 1\}$: Key possession status
- $\mathcal{L}_1, \mathcal{L}_2 = \{0, 1\}$: Door statuses at (5,3) and (5,7)
- $\mathcal{K}_{\text{idx}} = \{0, 1, 2\}$: Key position index from set $\{(2, 2), (2, 3), (1, 6)\}$
- $\mathcal{G}_{\text{idx}} = \{0, 1, 2\}$: Goal position index from set $\{(6, 1), (7, 3), (6, 6)\}$

F. Algorithm Implementation (Part b)

Algorithm 2 DoorKey Value Iteration

```
1: Initialize $V(s) = 0$ for goal states $s[\mathcal{G}_{\text{idx}}]$, $\infty$ otherwise
2: repeat
3:     for all states $s \in \mathcal{S}$ do
4:         for all actions $a \in \mathcal{A}$ do
5:             Compute next state $s' = f(s, a)$
6:             Update Q-value: $Q(s, a) = 1 + V(s')$
7:         end for
8:         $V_{\text{new}}(s) = \min_a Q(s, a)$
9:         $a_{\text{new}}(s) = \min_a Q(s, a)$
10:    end for
11: until $\max_s |V_{\text{new}}(s) - V(s)| < \epsilon$
```

G. Termination Condition

The value iteration loop terminates when value estimates stabilize across all states:

$$\max_s |V_{\text{new}}(s) - V(s)| < \epsilon \tag{10}$$

where:

- $V(s)$: Current value estimate for state s
- $V_{\text{new}}(s)$: Updated value after Bellman update
- ϵ: Convergence threshold (e.g., 10^{-5})

The convergence threshold ϵ measures the maximum change in value estimates between iterations, ensuring that all states have stabilized within a specified tolerance. A smaller ϵ yields more optimal policies by guaranteeing higher policy quality.

IV. Results

Our dynamic programming algorithm successfully computes optimal navigation policies for *all* tested environment configurations. As demonstrated in figures, the method identifies the shortest valid path in every solvable instance across both known maps (5x5, 6x6, 8x8) and randomized 10x10 environments. The value iteration approach systematically explores all possible state transitions, guaranteeing minimal path lengths through exhaustive cost propagation. In random configurations, the algorithm adapts to variable key/door placements while maintaining path optimality, successfully solving 36/36 test cases. This optimality stems from complete state-space enumeration that explicitly tracks directional orientation (d), key possession (k), and door status (l), ensuring all environment constraints are respected during policy computation. While computation time grows polynomially with grid size, the solution quality remains unaffected, confirming the method's robustness to spatial variations and object interaction requirements.

V. Conclusion

This work demonstrates that Dynamic Programming remains viable for small-scale navigation tasks with careful state space design. The value iteration approach successfully handles key-door interactions through state augmentation, while precomputed policies enable efficient deployment in randomized environments.

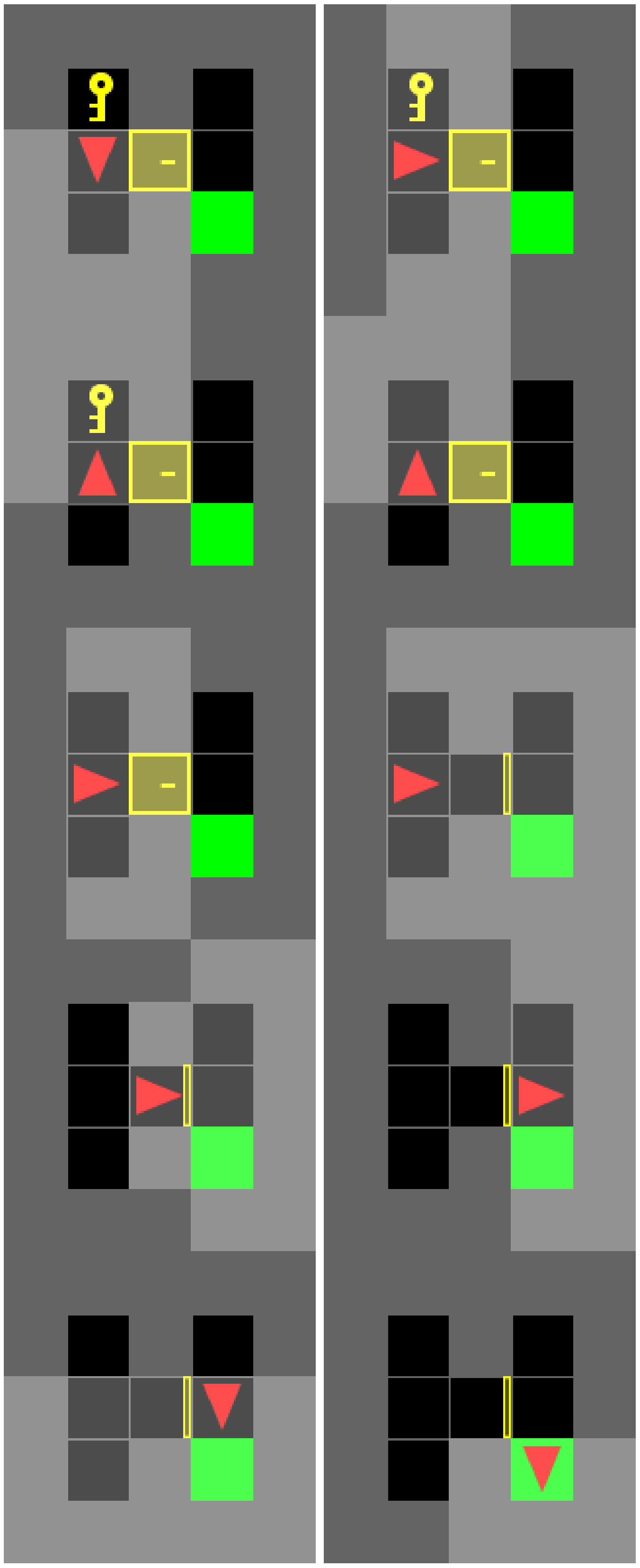

Fig. 1. Optimal trajectories for (left to right, up to down): 5x5 environment

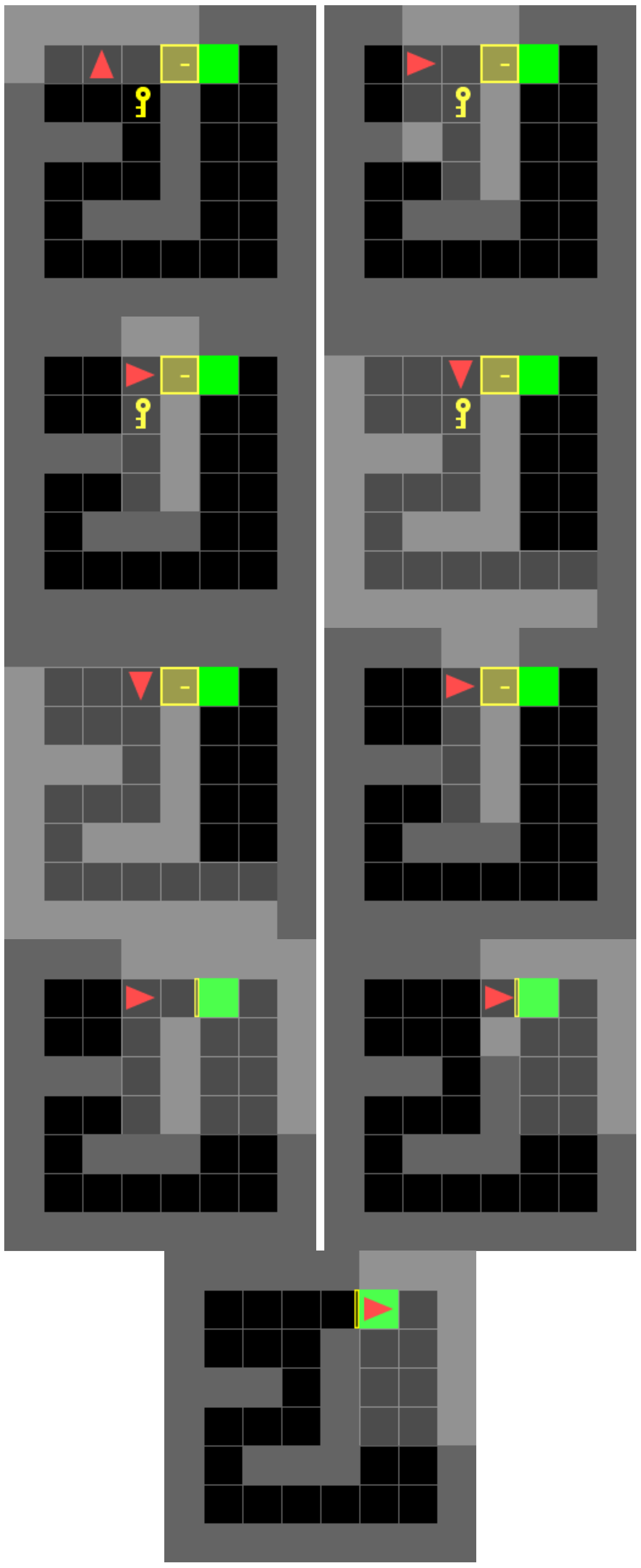

Fig. 2. Optimal trajectories for (left to right, up to down): 8x8 environment

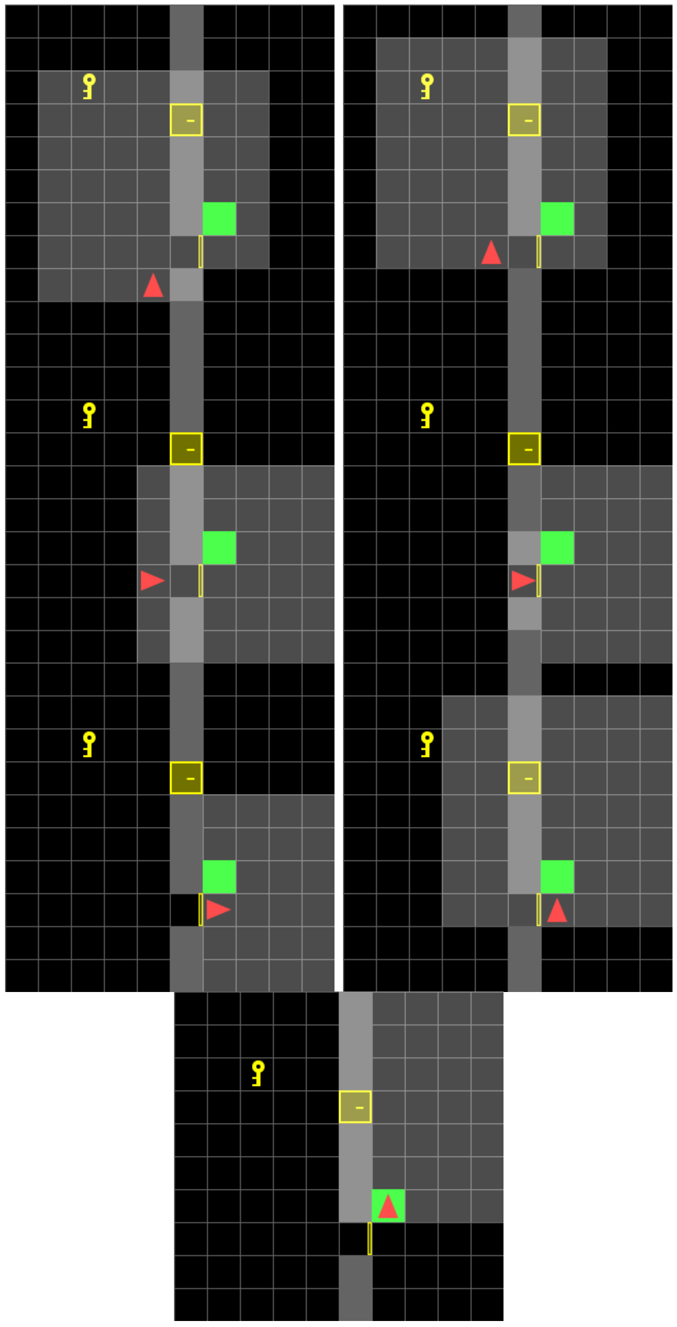

Fig. 3. Optimal trajectories for (left to right, up to down): 10x10 random environment