Dynamic Programming for Autonomous Navigation in the Door & Key Environment

Anderson Compalas
Department of Electrical and Computer Engineering
University of California San Diego
La Jolla, California
acompalas@ucsd.edu

***Abstract*—This project implements a Dynamic Programming (DP) algorithm for optimal autonomous navigation in the Door & Key environment. An agent must navigate from an initial position to a goal location, potentially retrieving a key and unlocking a door along the way. The problem is formulated as a deterministic Markov Decision Process (MDP) with non-uniform action costs, and solved via value iteration. Two scenarios are addressed: (A) a known-map setting where a separate optimal policy is computed for each of seven environments, and (B) a random-map setting where a single unified policy is computed over an augmented state space covering all 36 possible environment configurations. All policies are verified to reach the goal at minimum cost.**

***Index Terms*—Markov Decision Process, Dynamic Programming, Value Iteration, Autonomous Navigation, Optimal Control**

I. Introduction

Autonomous navigation under constraints is a fundamental challenge in robotics. A robot must plan a sequence of actions to reach a goal while respecting environmental obstacles and satisfying task-specific requirements such as acquiring tools or manipulating objects along the way. The Door & Key environment captures this challenge in a structured grid world: an agent must navigate to a goal location, and if a locked door blocks the path, it must first retrieve a key and unlock the door before proceeding.

This problem is directly relevant to real-world robotics applications such as indoor service robots navigating through locked rooms, warehouse automation systems that must interact with physical infrastructure, and search-and-rescue robots operating in environments with access-controlled areas. In all of these settings, the robot must plan over a sequence of high-level actions with heterogeneous costs and constraints.

Our approach formulates the problem as a deterministic MDP and solves it using the Dynamic Programming algorithm. Value iteration is run over the full state space to compute an optimal cost-to-go function and a corresponding closed-loop policy. For the known-map scenario, one policy is computed per environment. For the random-map scenario, the state space is augmented with discrete indices for the key location and goal location, enabling a single DP pass to produce a policy valid across all 36 environment configurations.

II. Problem Statement

A. MDP Formulation

We model the Door & Key navigation problem as a deterministic, fully observable, first-exit MDP defined by the tuple $(\mathcal{X}, \mathcal{U}, f, x_0, T, \ell, q)$.

B. State Space $\mathcal{X}$

Known Map. Each state encodes the complete configuration of the agent and the environment:

$$x = (c, r, \theta, h, \boldsymbol{d}) \tag{1}$$

where:

- $(c, r) \in \{0, \ldots, W-1\} \times \{0, \ldots, H-1\}$ is the agent's grid position (column, row),
- $\theta \in \{0, 1, 2, 3\}$ is the agent's facing direction (RIGHT, DOWN, LEFT, UP),
- $h \in \{\text{False}, \text{True}\}$ indicates whether the agent is carrying the key,
- $\boldsymbol{d} \in \{\text{False}, \text{True}\}^{N_d}$ is a tuple of door-open flags, one per door.

This formulation satisfies the Markov property because all variables affecting future transitions, agent pose, key possession, and door states, are fully captured in the state x.

Random Map. For the random-map scenario, the state is augmented with two discrete indices encoding environment-specific parameters that vary across the 36 configurations:

$$x = (c, r, \theta, h, \boldsymbol{d}, k, g) \tag{2}$$

where $k \in \{0, 1, 2\}$ indexes the key location and $g \in \{0, 1, 2\}$ indexes the goal location. This augmentation allows a single MDP to represent all 36 environments simultaneously. The total extended state space has $10 \times 10 \times 4 \times 2 \times 4 \times 3 \times 3 = 28{,}800$ states.

C. Control Space $\mathcal{U}$

The agent has five actions:

$$\mathcal{U} = \{\text{MF}, \text{TL}, \text{TR}, \text{PK}, \text{UD}\} \tag{3}$$

corresponding to Move Forward, Turn Left, Turn Right, Pick Up Key, and Unlock Door.

D. Motion Model f

The transition function $f : \mathcal{X} \times \mathcal{U} \to \mathcal{X}$ is deterministic. Since the environment is fully observable and there is no process noise, the transition probability reduces to a point mass: $p_f(x' \mid x, u) = \mathbf{1}[x' = f(x, u)]$.

The motion model is defined as follows. Let $(dc, dr) = \Delta(\theta)$ be the unit displacement vector in the facing direction θ, and let $(fc, fr) = (c + dc, r + dr)$ denote the front cell.

Turn Left / Turn Right:

$$f(x, \text{TL}) = (c, r, (\theta - 1) \bmod 4, h, \boldsymbol{d}) \tag{4}$$

$$f(x, \text{TR}) = (c, r, (\theta + 1) \bmod 4, h, \boldsymbol{d}) \tag{5}$$

Move Forward: If (fc, fr) is a wall, or a closed door, the agent stays in place. Otherwise:

$$f(x, \text{MF}) = \begin{cases} (fc, fr, \theta, h, \boldsymbol{d}) & \text{if passable} \\ x & \text{otherwise} \end{cases} \tag{6}$$

Pick Up Key: Succeeds only if the front cell contains the key and the agent is not already carrying one:

$$f(x, \text{PK}) = \begin{cases} (c, r, \theta, \text{True}, \boldsymbol{d}) & \text{if } (fc, fr) = \text{key\_pos},\ h = \text{False} \\ x & \text{otherwise} \end{cases} \tag{7}$$

Unlock Door: Succeeds only if the front cell is a locked door and the agent has the key:

$$f(x, \text{UD}) = \begin{cases} (c, r, \theta, h, \boldsymbol{d}') & \text{if } (fc, fr) \in \text{doors},\ h = \text{True} \\ x & \text{otherwise} \end{cases} \tag{8}$$

where $\boldsymbol{d}'$ is $\boldsymbol{d}$ with the corresponding door flag set to True.

Invalid actions (e.g., MF into a wall, PK when the key is not in front) are modeled as no-op transitions: the state does not change, but the action cost is still incurred.

E. Initial State x_0

For each known-map environment, the initial state is read directly from the loaded environment:

$$x_0 = (\text{init\_col},\ \text{init\_row},\ \text{init\_dir},\ \text{False},\ \boldsymbol{d}_0) \tag{9}$$

where $\boldsymbol{d}_0$ encodes the initial open/locked status of each door.

For the random-map scenario, the initial state is fixed across all 36 environments:

$$x_0 = (4,\ 8,\ \text{UP},\ \text{False},\ \boldsymbol{d}_0,\ k,\ g) \tag{10}$$

where k and g depend on the specific environment configuration.

F. Planning Horizon T

The problem is formulated as a **first-exit MDP**: the episode terminates as soon as the agent enters the goal cell. There is no fixed time horizon. This is equivalent to the infinite-horizon formulation where terminal states absorb with zero cost, and value iteration converges to the optimal cost-to-go.

G. Stage Cost ℓ

The stage cost depends only on the action taken, not on the state:

$$\ell(x, u) = \begin{cases} 1 & u \in \{\text{TL}, \text{TR}\} \\ 3 & u = \text{MF} \\ 2 & u = \text{PK} \\ 5 & u = \text{UD} \end{cases} \tag{11}$$

This cost structure reflects the physical difficulty of each action: turning is cheapest, moving forward costs more, and unlocking a door is the most expensive action.

H. Terminal Cost q

The terminal cost is zero. The goal of the agent is to minimize the total stage cost accumulated before reaching the goal:

$$q(x) = 0 \quad \text{for all } x \in \mathcal{X}_{\text{goal}} \tag{12}$$

I. Optimal Control Objective

The agent seeks a policy $\pi^* : \mathcal{X} \to \mathcal{U}$ that minimizes the total cost to reach the goal:

$$\pi^* = \arg\min_{\pi} \sum_{t=0}^{T-1} \ell(x_t, \pi(x_t)) \tag{13}$$

subject to $x_{t+1} = f(x_t, \pi(x_t))$ and $x_T \in \mathcal{X}_{\text{goal}}$.

III. Technical Approach

A. Dynamic Programming via Value Iteration

We solve the MDP using the Dynamic Programming algorithm from Lecture 4. For a first-exit MDP with deterministic transitions and discount factor $\gamma = 1$, the Bellman optimality equation reduces to:

$$V^*(x) = \min_{u \in \mathcal{U}} \left[\ell(x, u) + V^*(f(x, u))\right] \tag{14}$$

with boundary condition $V^*(x) = 0$ for all $x \in \mathcal{X}_{\text{goal}}$.

Since the expectation in the standard DP recursion disappears for deterministic transitions (p_f is a point mass), Eq. (14) is solved exactly at each state. The optimal policy is recovered as:

$$\pi^*(x) = \arg\min_{u \in \mathcal{U}} \left[\ell(x, u) + V^*(f(x, u))\right] \tag{15}$$

B. Value Iteration Algorithm

We implement value iteration, which iteratively applies the Bellman update until convergence. The algorithm is described in Algorithm III-B.

Initialize $V(x) \leftarrow 0$ for $x \in \mathcal{X}_{\text{goal}}$; $V(x) \leftarrow \infty$ otherwise
repeat
  $\delta \leftarrow 0$
  for each $x \in \mathcal{X} \setminus \mathcal{X}_{\text{goal}}$ **do**
    $V_{\text{new}}(x) \leftarrow \min_{u \in \mathcal{U}} \left[\ell(x, u) + V(f(x, u))\right]$
    $\pi(x) \leftarrow \arg\min_{u \in \mathcal{U}} \left[\ell(x, u) + V(f(x, u))\right]$
    $\delta \leftarrow \max(\delta,\ |V_{\text{new}}(x) - V(x)|)$
  end for
  $V \leftarrow V_{\text{new}}$
until $\delta < \epsilon$

11: **return** V, π

Convergence is declared when the maximum change in the value function across all states falls below $\epsilon = 10^{-6}$. Because the state space is finite with strictly positive stage costs and absorbing goal states, value iteration is guaranteed to converge to the unique optimal solution.

C. State Space Construction

For the known-map scenario, the full state space is enumerated as the Cartesian product:

$$\begin{aligned}\mathcal{X} = \{0,\dots,W-1\} \times \{0,\dots,H-1\} \\ \times \{0,\dots,3\} \times \{0,1\} \times \{0,1\}^{N_d}\end{aligned} \tag{16}$$

For a map with $W = H = 8$ and $N_d = 1$ door, this gives $8 \times 8 \times 4 \times 2 \times 2 = 1{,}024$ states.

For the random-map scenario, the augmented state space adds the key index $k \in \{0,1,2\}$ and goal index $g \in \{0,1,2\}$, giving $10 \times 10 \times 4 \times 2 \times 4 \times 3 \times 3 = 28{,}800$ states.

D. Part A: Known-Map Policy

For each of the seven known environments, we call `dp_solve(env)`, which runs value iteration on the specific map layout. The environment is parsed by scanning the grid for walls, key, door, and goal positions. The resulting policy is then simulated from the initial state to extract the optimal action sequence.

E. Part B: Random-Map Unified Policy

The 36 random environments share a fixed wall layout (10×10 grid with a vertical wall at column 5 and two doors at $(5,3)$ and $(5,7)$), but differ in key position ($k \in \{(2,2),(2,3),(1,6)\}$), goal position ($g \in \{(6,1),(7,3),(6,6)\}$), and door states (each open or locked).

Rather than computing 36 separate policies, we augment the state with (k,g) and solve a single MDP over the extended state space. The key insight is that since the agent observes the full state (including which key and goal are active), the augmented policy $\pi^*(c,r,\theta,h,\boldsymbol{d},k,g)$ is optimal for each individual environment when k and g are set to match that environment's configuration.

A separate transition function `transition_random` handles the fixed random-map geometry, including the two doors at fixed positions and the three possible key locations.

IV. Results

A. Part A: Known Maps

Table I summarizes the optimal cost and action sequence length for each of the seven known environments. Figure 1 shows the corresponding optimal trajectories. All reported trajectories were visually verified to confirm that the computed policies reach the goal and correctly retrieve the key and unlock the door when required.

Several patterns are evident. The *direct* environments have lower costs because the agent can reach the goal without retrieving the key. The *shortcut* environments have moderate costs because using the locked door gives a shorter path despite the pickup and unlock costs. The *normal* environments have the highest costs because the agent must take a longer route to retrieve the key before reaching the goal.

TABLE I: Part A: Known Map Results

Environment	Optimal Cost	Steps
doorkey-5x5-normal	28	13
doorkey-6x6-normal	33	14
doorkey-6x6-direct	13	5
doorkey-6x6-shortcut	15	6
doorkey-8x8-normal	54	21
doorkey-8x8-direct	17	7
doorkey-8x8-shortcut	19	8

The value iteration algorithm converged in 12 to 25 iterations across all seven maps.

B. Part B: Random Maps

The unified policy solved all 36 random environments successfully. Table II summarizes the random-map results. Figures 2–4 show the optimal trajectories for all 36 random-map environments.

The cost variation across environments reflects the interaction between key location, goal location, and door states. The minimum cost of 14 occurs when a direct path to the goal is available without key retrieval. The maximum cost of 53 occurs when the key and goal placement require a longer route with key pickup and door unlocking.

The value iteration algorithm converged in 25 iterations over the 28,800-state extended state space, completing efficiently while producing a single policy valid for all random-map configurations.

C. Algorithm Performance Discussion

What worked. Value iteration converged reliably for all environments and both scenarios. The deterministic transition model and strictly positive costs allow the Bellman updates to propagate optimal costs from the goal through the finite state space. The state augmentation approach for Part B was effective because it encoded all 36 environment configurations in a single MDP, producing one coherent policy.

Invalid action handling. Modeling invalid actions as no-op transitions that still incur cost simplified the transition function. The DP naturally avoids invalid actions because they increase cost without making progress.

Convergence behavior. The ∞ to finite transition tracking in the convergence check was important. Without this check, the algorithm could incorrectly terminate before values propagated from the goal to reachable states.

Limitations. The main limitation is full state-space enumeration. The method works well for these small grid worlds, but larger maps, more doors, or more unknown environment parameters would increase the state space substantially. For larger problems, graph search methods such as A* or more compact planning representations would be more scalable.

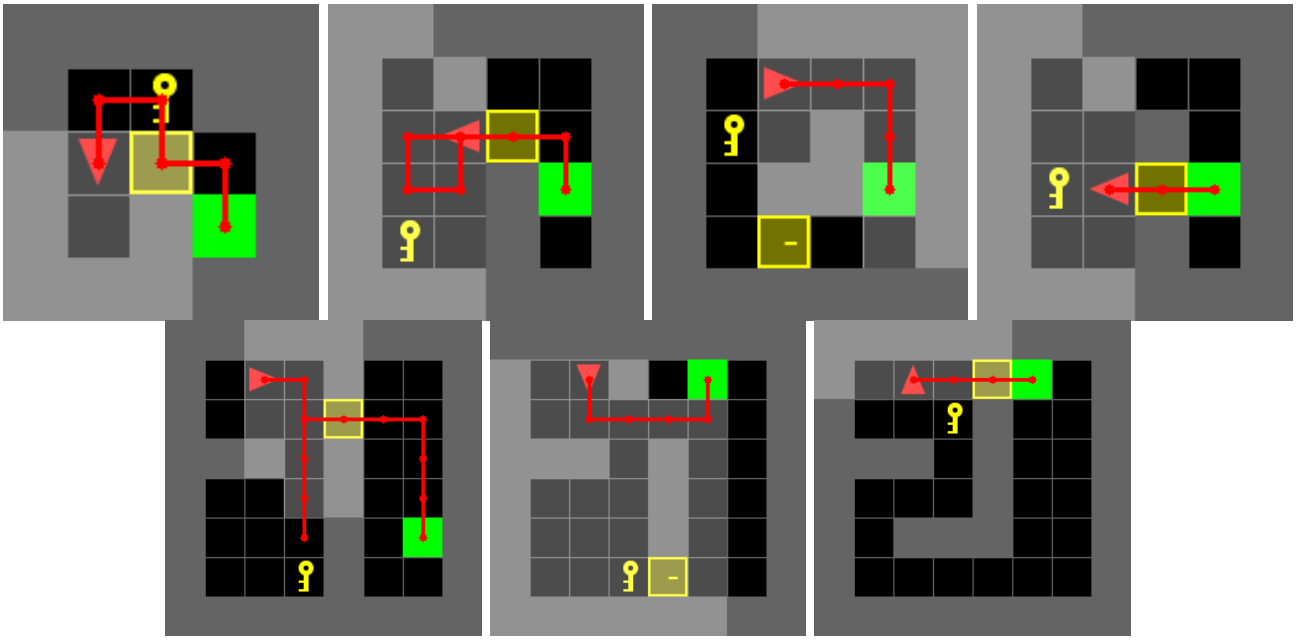

Fig. 1: Optimal trajectories for the seven known-map environments. The red line shows the sequence of cells visited by the agent from the initial state to the goal, overlaid on the initial environment state. The agent picks up the key by facing the key cell and executing PK without entering it, so the red path passes adjacent to the key rather than through it.

TABLE II: Part B: Random Map Results – All 36 Environments

Environment	Cost	Steps	Solved	Environment	Cost	Steps	Solved
DoorKey-10x10-1	29	11	OK	DoorKey-10x10-19	26	10	OK
DoorKey-10x10-2	29	11	OK	DoorKey-10x10-20	40	15	OK
DoorKey-10x10-3	29	11	OK	DoorKey-10x10-21	14	6	OK
DoorKey-10x10-4	50	19	OK	DoorKey-10x10-22	32	12	OK
DoorKey-10x10-5	25	9	OK	DoorKey-10x10-23	14	6	OK
DoorKey-10x10-6	25	9	OK	DoorKey-10x10-24	47	18	OK
DoorKey-10x10-7	26	10	OK	DoorKey-10x10-25	29	11	OK
DoorKey-10x10-8	46	17	OK	DoorKey-10x10-26	29	11	OK
DoorKey-10x10-9	14	6	OK	DoorKey-10x10-27	29	11	OK
DoorKey-10x10-10	32	12	OK	DoorKey-10x10-28	50	19	OK
DoorKey-10x10-11	14	6	OK	DoorKey-10x10-29	25	9	OK
DoorKey-10x10-12	53	20	OK	DoorKey-10x10-30	25	9	OK
DoorKey-10x10-13	29	11	OK	DoorKey-10x10-31	26	10	OK
DoorKey-10x10-14	29	11	OK	DoorKey-10x10-32	46	17	OK
DoorKey-10x10-15	29	11	OK	DoorKey-10x10-33	14	6	OK
DoorKey-10x10-16	44	17	OK	DoorKey-10x10-34	32	12	OK
DoorKey-10x10-17	25	9	OK	DoorKey-10x10-35	14	6	OK
DoorKey-10x10-18	25	9	OK	DoorKey-10x10-36	41	16	OK
Summary: 36/36 solved, min cost = 14, max cost = 53, mean cost = 30.1, value iteration converged in 25 iterations.							

V. Conclusion

This project implemented a Dynamic Programming solution to the Door & Key navigation problem. The problem was formulated as a deterministic first-exit MDP with a five-dimensional state space encoding position, direction, key possession, and door status. Value iteration over the full state space produced optimal closed-loop policies for all seven known-map environments and a single unified policy for all 36 random-map configurations via state augmentation. All policies were verified to reach the goal at minimum cost, with GIF visualizations confirming correct agent behavior.

Acknowledgments

We thank Professor Nikolay Atanasov and the ECE 276B teaching assistants for providing the project framework and starter code.

References

[1] N. Atanasov, "ECE 276B: Planning & Learning in Robotics – Lecture 3: Markov Decision Processes," University of California San Diego, 2026.

[2] N. Atanasov, "ECE 276B: Planning & Learning in Robotics – Lecture 4: The Dynamic Programming Algorithm," University of California San Diego, 2026.

Appendix

A. *Project Structure*

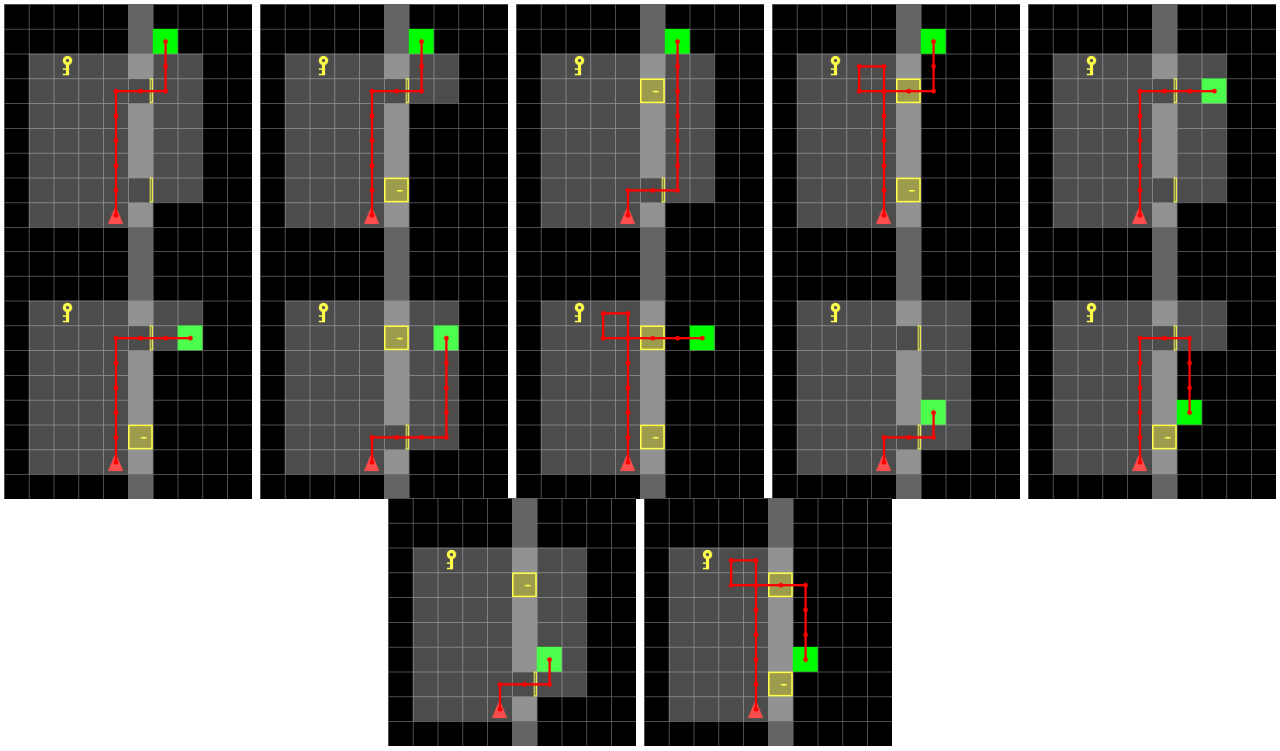

Fig. 2: Optimal trajectories for random-map environments 1–12.

```
ECE276B_PR1/
  starter_code/
    doorkey.py        # Main file: DP solver + Part A
     + Part B
    utils.py
    create_env.py
    example.py
    envs/
      example-8x8.env
      known_envs/     # 7 known map .env files (
    provided by instructor)
      random_envs/    # 36 random map .env files (
    provided by instructor)
    gif/              # Output GIFs written here
    requirements.txt
```

B. Dependencies

```
pip install -r requirements.txt
```

Which installs: `numpy`, `gymnasium==0.28.1`, `matplotlib`, `minigrid`, `imageio`, `opencv-python`.

C. Important: Regenerate Environment Files

Both the known-map and random-map `.env` files were provided by the instructor in `envs/known_envs/` and `envs/random_envs/` respectively. These files are pickle files serialized with a specific version of NumPy. Loading them on a different NumPy installation will produce the following error:

```
ValueError: <class 'numpy.random._pcg64.PCG64'>
    is not a known BitGenerator module.
```

To resolve this, re-serialize all environment files using your local NumPy version by running the following command once before first use:

```
cd starter_code
python create_env.py
```

This does not change any environment configurations, only the pickle format. Our `doorkey.py` then loads and solves these provided environments directly.

D. Running the Solver

Run both Part A and Part B from the `starter_code/` directory:

```
cd starter_code
python doorkey.py
```

This will:

1) Solve all 7 known-map environments (Part A) and save GIFs to `gif/`
2) Build the unified policy for all 36 random-map environments (Part B) and save GIFs to `gif/`

Expected output:

```
  Converged in 15 iterations
doorkey-5x5-normal: cost=28, steps=13
  TL -> TL -> MF -> TR -> PK -> MF -> TR -> UD ->
    ...
GIF is written to ./gif/doorkey-5x5-normal.gif
...
--- Part B: Random Maps ---
  Total states: 28800
  Converged in 25 iterations
  [OK] DoorKey-10x10-1.env: cost=29, steps=11
```

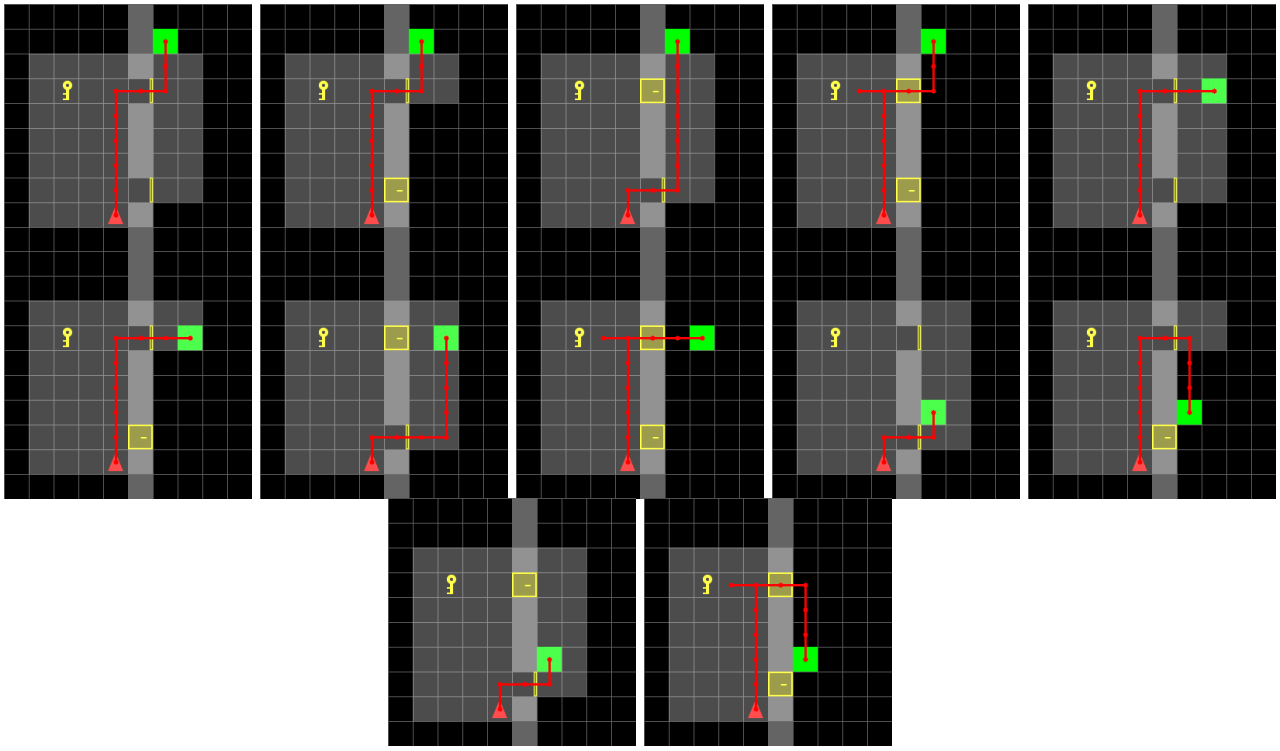

Fig. 3: Optimal trajectories for random-map environments 13–24.

```
...
  Solved 36/36 environments
```

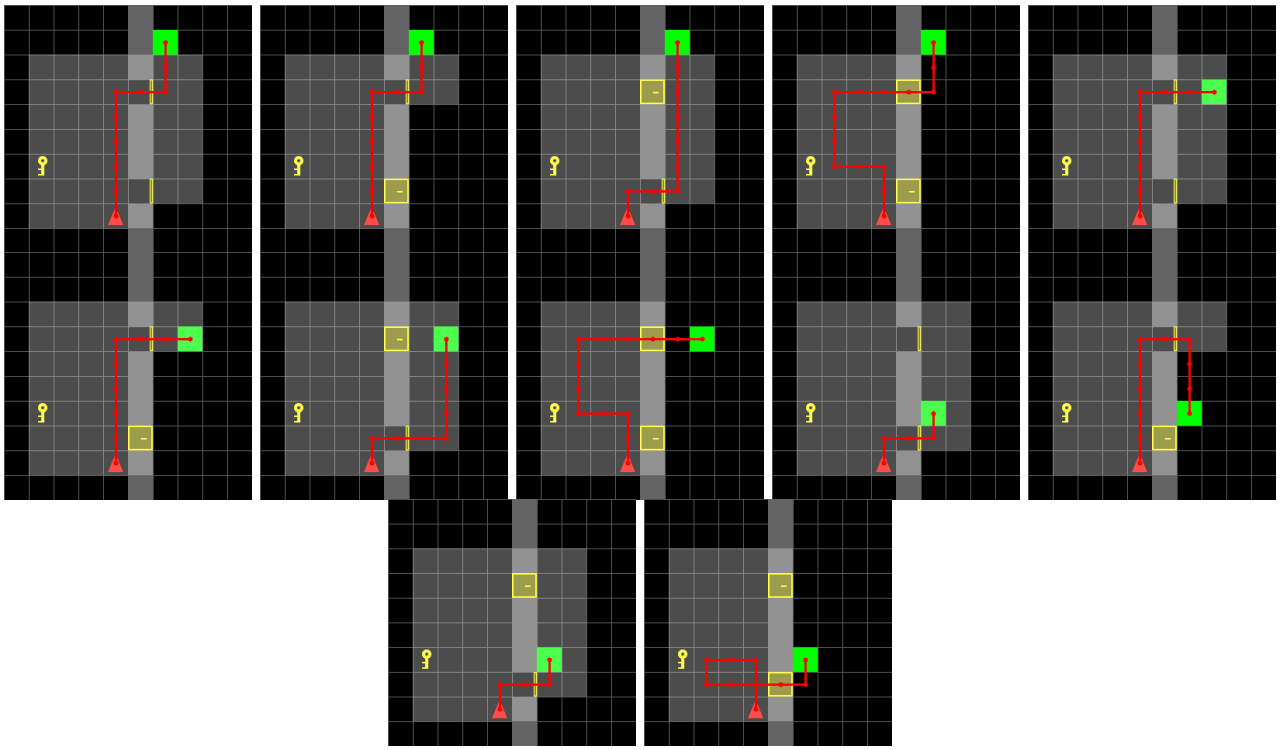

Fig. 4: Optimal trajectories for random-map environments 25–36.