Dynamic Programming for Deterministic Shortest Path problem

Sambaran Ghosal
Department of Electrical and Computer Engineering
UC San Diego
La Jolla, USA
sghosal@ucsd.edu

Nikolay Atanasov
Department of Electrical and Computer Engineering
UC San Diego
La Jolla, USA
natanasov@ucsd.edu

***Abstract*—Deterministic shortest path problem aims at finding the shortest possible path from a start position to the goal position by following a sequence of control actions. We solve this problem for a door-key environment in gym where the goal is to make the robot move from start to goal. The robot may need to pick up a key to unlock doors that may be in the way of the goal. We implement a dynamic programming algorithm to solve the problem.**

***Index Terms*—Determinstic Shortest Path(DSP), Dynamic Programming, GYM, Label Correction**

I. Introduction

Often in real world robotics applications, a common problem is to get the robot from a start point to a goal point in the shortest path possible along with satisfying some constraints that may be provided by the environment(obstacles) or the robot kinematics and dynamics. This is one of the crucial problems in robotics and is often termed as Planning and Control. The problem of obtaining the shortest possible path from a start position to the goal position is termed as **Deterministic Shortest Path(DSP)** problem. The most common method to solve these kind of DSP problems is to apply a **Dynamic Programming(DP)** algorithm. There are many variants to the basic Dynamic Programming algorithm such as Label Correction, Backwards Dynamic programming, forward dynamic programming, Dijkstra's algorithm, Floyd-Marshall algorithm and many more.

In this project, we will focus on solving a door-key environment using the Backward Dynamic Programming algorithm. The environment consists of grids that represent empty spaces, walls, doors or a key and a goal where the robot has to reach. The robot can go through the door only if the door is unlocked. To unlock the door or pickup the key, the robot has to be in the cell adjacent to these objects and face towards the cell containing the key or the door. The doors can either be open or closed at the beginning. We want to find the shortest path from the robot start position to the goal position using the backward dynamic programming.

II. Problem Formulation

The **Deterministic Shortest Path** problem can be described as follows : Given a graph with set of vertices $\mathcal{V}$ which represent the states the robot can have, edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$ which represents possible transitions from states to states, and edge weights $\mathcal{C} = \{c_{ij} \in R \cup \infty | (i,j) \in \mathcal{V}$, where c_{ij} represents the cost of transition to go from a state/vertex i to j. The **objective** of the DSP problem is to obtain the shortest path from a given start state s to a goal state τ. **Path** can be defined as the sequence of vertices from s to τ as $P_{s,\tau} := \{i_{1:q} | i_k \in \mathcal{V}, i_1 = s, i_q = \tau\}$. The path length associated with a path is defined as the sum of cost of transition from i to $j \in P_{s,\tau}$ as $J^{i_{1:q}} = \sum_{k=1}^{q-1} c_{i_k, i_{k+1}}$. Hence the objective can be formulated as finding the path from s to τ with the minimum path length.

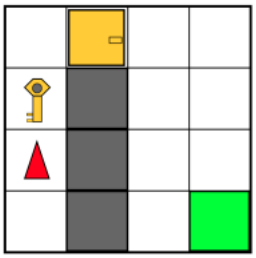

Fig. 1: Door Key Environment: An agent (red triangle) needs to navigate to a goal location (green square). The white squares are traversable, while the gray squares are obstacles that the agent cannot go through. The agent may need to pick up a key if a door along the way to the goal.

$$dist(s,\tau) = \min_{i_{1:q} \in P_{s,\tau}} J^{i_{1:q}} \tag{1}$$

$$i^*_{1:q} = \underset{i_{1:q} \in P_{s,\tau}}{argmin}\, J^{i_{1:q}} \tag{2}$$

Here $i^*_{1:q}$ is the shortest path from s to tau with the minimum path length. We can also obtain the sequence of controls $u \in \mathcal{U}$, where $\mathcal{U}$ is the set of feasible controls, that can produce the shortest path sequence starting from the start s to the goal τ.

One of the assumptions that has to be satisfied for feasible solutions to the DSP problem is that there should not be the presence of negative cost cycles in the graph i.e. it should not be possible that we start from a node, and after a finite sequence of paths get back to this node again and the cost of this whole transition is less than 0, otherwise we would be

stuck in a cycle of this transition because the cost keeps on decreasing.

III. Technical Approach

Given the DSP problem elements $\mathcal{V}$, edge set $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V}$, and edge weights $\mathcal{C} = \{c_{ij} \in R \cup \infty | (i,j) \in \mathcal{V}$, the DSP can be converted to a **Determinstic Finite State(DFS)** optimal control problem. The DFS problem is to find the sequence of optimal controls $u_{0:T-1}$ satisfying

$$\underset{u_{0:T-1}}{min}\ q(x_T) + \sum_{t=0}^{T-1} l(x_t, u_t) \tag{3}$$

where T is the terminal time, $q(x)$ is the terminal cost of a state, $l(x,u)$ is the stage cost of taking action u in state x.

A. DSP to DFS conversion

A DSP can be converted to a graph representation of the DFS problem. We specify a start node $s := (0, x_0)$ given $x_0 \in \mathcal{X}$, where $\mathcal{X}$ is the state space, the set of all possible vertices in the DSP problem. At any time t, every state $x \in \mathcal{X}$ is represented as node $i := (t,x)$ defined as follows

$$\mathcal{V} := \{s\} \cup (\cup_{t=1}^{T} \{(t,x) | x \in \mathcal{X}\}) \cup \{\tau\} \tag{4}$$

where s is the start node, τ is a dummy node with arc length $c_{i,\tau}$ equal to the terminal cost $q(x)$ of the DFS.

- The edge between two nodes $i = (t,x)$ and $j = (t', x')$ ss finite, $c_{ij} < \infty$, only if $t' = t+1$ and $x' = f(x,u)$ for some $u \in \mathcal{U}$.
- The edge weight between two nodes $i = (t,x)$ and $j = (t+1, x')$ is defined as the smallest cost between x and x':

$$C := \{c_{(t,x),(t+1,x')} = \underset{\substack{u \in \mathcal{U} \\ s.t. x' = f(x,u)}}{min} l(x,u)\} \cup \{c_{(T,x),\tau} = q(x)\} \tag{5}$$

Assuming the non-negative cycle cost, the DSP and DFS problem are equivalent using the above stated formulation. Besides that,

- The time horizon of the DFS optimal control problem is set to $T := |\mathcal{V}| - 1$, i.e. the number of vertices in the DSP graph subtracted by one.
- State space of the DFS problems is defined as $\mathcal{X} = \mathcal{V}$, and the control space is defined as $\mathcal{U} = \mathcal{V}$, which can be thought of that the control says which state to transition to from the current state.
- Motion Model of the DFS problem is defined as

$$x_{t+1} = f(x_t, u_t) := \begin{cases} x_t & x_t = \tau \\ u_t & otherwise \end{cases} \tag{6}$$

Stage and terminal costs are defined as

$$l(x,u) := \begin{cases} 0 & x = \tau \\ c_{x,u} & otherwise \end{cases} \tag{7}$$

$$q(x) := \begin{cases} 0 & x = \tau \\ \infty & otherwise \end{cases} \tag{8}$$

B. Dynamic Programming applied to DSP

Due to the above shown equivalence between the DSP and the DFS problems, the DSP can also be solved using the Dynamic Programming Algorithm. $V_t(i)$ defines the optimal cost from node i to τ in at most $T-t$ steps. Upon termination, we get $V_0(s) = J^{i^*_{1:q}} = dist(s,\tau)$. The algorithm can be terminated if for two successive times, the value function is unchanged for all states. The Backward Dynamic Algorithm is described below :

Algorithm 1 Backward Dynamic Programming on DSP

Require: : vertices $\mathcal{V}$, start s, goal τ, costs c_{ij}
  $T \leftarrow |\mathcal{V}| - 1$
  $V_T(\tau) = V_{T-1}(\tau) = ... = V_0(\tau) = 0$
  $V_T(i) = \infty, \forall i \in \mathcal{V} \sim \{\tau\}$
  $V_{T-1}(i) = c_{i,\tau}, \forall i \in \mathcal{V} \sim \{\tau\}$
  $\pi_{T-1}(i) = \tau, \forall i \in \mathcal{V} \sim \{\tau\}$
  for t = T-2 to 0 **do**
    $Q_t(i,j) = c_{ij} + V_{t+1}(j), \forall i \in \mathcal{V} \sim \{\tau\}, j \in \mathcal{V}$
    $V_t(i) = \underset{j \in \mathcal{V}}{min}\ Q_t(i,j), \forall i \in \mathcal{V} \sim \{\tau\}$
    $\pi_t(i) = \underset{j \in \mathcal{V}}{argmin}\ Q_t(i,j), \forall i \in \mathcal{V} \sim \{\tau\}$
    if $V_t(i) = V_{t+1}(i), \forall i \in \mathcal{V} \sim \{\tau\}$ **then**
      break
    end if
  end for

C. Formulating the door-key problem

Given the above general formulation of the DSP problem and construction of the equivalent DFS problem, we now try to formulate our door-key problem. Our project requires us to compute the optimal policy for two cases :

- Part A : This consists of 7 different environments with different sizes, different position of doors, keys, agent starting positions, goal and walls. It is required that we obtain the value function and the policy function for each of these separately.
- Part B : In this part, all the environments under consideration have the same size, same position of walls and doors. The key is said to be randomly chosen from three possible locations, as is the goal of the robot. The starting position of the robot is always (3,5) facing up. It is desired that we construct a single value and policy function that will be able to solve the DSP problem for any of the random maps satisfying the above criteria. The aim in this part will be to define our state space in such a way that all random environments are encapsulated.

For both Part A and Part B, we define the stage costs and terminal costs as follows :

$$l(x,u) := \begin{cases} 0 & x = \tau \\ 1 & otherwise \end{cases} \tag{9}$$

$$q(x) := \begin{cases} 0 & x = \tau \\ \infty & otherwise \end{cases} \tag{10}$$

The control space of both the environments are also the same. The robot can either move forward, turn left, turn right, pick up the key or unlock the door. We represent these controls as :

$$\mathcal{U} := \{MF, TL, TR, PK, UD\} \tag{11}$$

Let us now define the state space for both the parts.

*1) **State Space for Part A**:* For each of the seven environments in Part A, we can define a single state as a dictionary consisting of the agent position, the direction the agent is facing, whether the agent is carrying the key or not, and whether the door is locked or not. Each of the environments in this part has only one door. Hence state x can be defined as :

$$x := \{\text{agent pos, direction, key, door}\} \tag{12}$$

where agent position can be any coordinate in the 2D permissible grid of that environment, agent direction can be up, down, left and right, key can be 0 indicating agent does not carry the key and 1 indicating the agent is carrying the key, and door can be 0 indicating the door is closed and 1 indicating the door is open.

From this, the state space $\mathcal{X}$ can be defined as the set of all possible state values. This can be constructed by taking the cartesian product of all possible values for each of the state attributes. So if the environment under consideration has a size of $W \times H$, the total number of states, or the cardinality of state space, $|\mathcal{X}| = W \times H \times 4 \times 2 \times 2$. Hence the total number of vertices in the DSP problem is equal to the number of elements in the state space, and the equivalent DFS problem has a time horizon of $T = |\mathcal{X}| - 1$.

*2) **State Space for Part B**:* As described before, for Part B, each random environment has the same size, same position of the wall, same starting position of the agent and same starting direction, same position of the doors, either of the two doors can be locked or unlocked in the beginning. The goal, key position is randomly selected from three possible locations for each.

The requirement for this part is that we should be able to get a single policy that will be able to generate the sequence of optimal controls to go from the starting position of the agent to the goal position, given any map in part B. For this, we would need to define our state in such a way that the state space covers all states from all of the maps. Also now there are two doors, hence we represent the door variable as a tuple indicating which door is open and which door is closed. Extending the definition of state defined in part A, here the state is defined as a dictionary consisting of the agent position, agent direction, position of the key in the map, the position of goal in the map, whether the agent is carrying the key or not, and the tuple representing the open-close status of each door. Hence :

$$x := \{\text{agent pos, direction, key pos, goal pos, key, door}\} \tag{13}$$

where as before, agent pos is now all possible positions in the grid, direction is up, down, left and right, key pos can be any of the three positions (1,1), (2,3) or (1,6), goal pos can be any of the three positions (5,1), (6,3) and (5,6), key can be either 1 or 0, and door can be any of the following : (1,0), (0,1), (0,0) and (1,1).

As in part A, the state space $\mathcal{X}$ is the cartesian product of all possible values of the state attributes. Given the size of the environment in this part is 8×8, the cardinality of the state space, $|\mathcal{X}| = 8 \times 8 \times 4 \times 3 \times 3 \times 2 \times 4 = 18,432$.

After the state space is defined for both parts, we can then call the Dynamic Programming Algorithm described in **Algorithm 1** and obtain seven different policy functions and value functions for each environment in Part A or a single policy function and value function for all the random environments in Part B. Now we can find the optimal sequence of actions for any environment and starting position of the agent using a simple forward pass.

IV. Results

In this section, we discuss the results of the Dynamic Programming Algorithm on the DSP problem we have in hand. We show results for the 7 environments of Part A, and 5 random environments selected from Part B. For each environment, we show the sequence of states obtained by following the policy function evaluated for the environments.

A. Part A

1) doorkey-5X5-normal: For this environment, the optimal policy sequence from the starting to the goal position was obtained as follows : **'TL', 'PK', 'TR', 'UD', 'MF', 'MF', 'TR', 'MF', 'MF'**.

The transitions and value functions along the path is shown for some states in Figure (2).

2) doorkey-6X6-direct: For this environment, the optimal policy sequence from the starting to the goal position was obtained as follows : **'TL', 'TL', 'MF', 'MF', 'MF'**.

The transitions and value functions along the path is shown for some states in Figure (3).

3) doorkey-6X6-normal: For this environment, the optimal policy sequence from the starting to the goal position was obtained as follows : **'MF', 'TR', 'PK', 'MF', 'MF', 'MF', 'TR', 'MF', 'UD', 'MF', 'MF', 'TR', 'MF', 'MF', 'MF', 'MF'**. The transitions and value functions along the path is shown for some states in Figure (4).

4) doorkey-6X6-shortcut: For this environment, the optimal policy sequence from the starting to the goal position was obtained as follows : **'PK', 'TL', 'TL', 'UD', 'MF', 'MF', 'MF'**.

The transitions and value functions along the path is shown for some states in Figure (5).

5) doorkey-8X8-direct: For this environment, the optimal policy sequence from the starting to the goal position was obtained as follows : **'TL', 'MF', 'MF', 'MF', 'MF'**.

The transitions and value functions along the path is shown for some states in Figure (6).

6) doorkey-8X8-normal: For this environment, the optimal policy sequence from the starting to the goal position was obtained as follows : **'TL', 'MF', 'TR', 'MF', 'MF', 'MF', 'TR', 'PK', 'TR', 'MF', 'MF', 'MF', 'MF', 'TR', 'UD', 'MF', 'MF', 'MF', 'TR', 'MF', 'MF', 'MF', 'MF', 'MF', 'MF'**.

The transitions and value functions along the path is shown for some states in Figure (7).

7) doorkey-8X8-shortcut: For this environment, the optimal policy sequence from the starting to the goal position was obtained as follows : **'MF', 'TR', 'PK', 'TR', 'MF', 'TR', 'MF', 'UD', 'MF', 'MF', 'MF'**.

The transitions and value functions along the path is shown for some states in Figure (8).

B. Part B

For Part B, we had a single policy that would work with all possible random environments satisfying the conditions mentioned before. We apply our policy function on some random samples from the 36 environments present in Part B and present the optimal action sequence results below.

1) Sample 1: For this environment, the optimal policy sequence from the starting to the goal position was obtained as follows : **'TR', 'MF', 'MF', 'TL', 'MF', 'MF', 'MF', 'MF', 'MF'**.

The transitions and value functions along the path is shown for some states in Figure (9).

2) Sample 2: For this environment, the optimal policy sequence from the starting to the goal position was obtained as follows : **'TR', 'MF', 'MF', 'TR', 'MF', 'MF'**.

The transitions and value functions along the path is shown for some states in Figure (10).

3) Sample 3: For this environment, the optimal policy sequence from the starting to the goal position was obtained as follows : **'TR', 'MF', 'MF', 'MF', 'TL', 'MF', 'MF', 'MF'**.

The transitions and value functions along the path is shown for some states in Figure (11).

4) Sample 4: For this environment, the optimal policy sequence from the starting to the goal position was obtained as follows : **'TL', 'MF', 'MF', 'TL', 'PK', 'TL', 'MF', 'MF', 'UD', 'MF', 'MF', 'TL', 'MF', 'MF', 'MF', 'MF', 'MF'**.

The transitions and value functions along the path is shown for some states in Figure (12).

5) Sample 5: For this environment, the optimal policy sequence from the starting to the goal position was obtained as follows : **'MF', 'MF', 'MF', 'MF', 'TL', 'MF', 'PK', 'TL', 'MF', 'MF', 'MF', 'MF', 'TL', 'MF', 'UD', 'MF', 'MF', 'TR', 'MF', 'MF'**.

The transitions and value functions along the path is shown for some states in Figure (13).

In the results shown, we plot the value function at the next possible states that can be achieved by performing an action in the current state. We can observe that the agent always performs the action that results in the minimum value function at the next state. If two possible next states have the same value function, an action is picked up randomly out of the two by the dynamic programming algorithm. For the purpose of drawing the value function at the states as a bar graph, if the value function of the state was ∞, it was replaced with the value of 50 which is much greater than the rest of the values.

V. CONCLUSION AND FUTURE WORK

In this project, we implemented a Backward Dynamic programming algorithm to solve the problem of Deterministic Shortest Path problem by converting the DSP to its equivalent Deterministic Finite Search Optimal control problem. We considered two different cases, partA where each environment is random of different sizes, wall positions, agent starting positions, goals, keys and door, and partB where each environment has the same agent starting position, direction, same wall positions but the goal and keys are selected at random from three different positions. We implemented a Markov Decision Process that was able to capture the state of all possible random environments in this case.

Future work may include improved algorithms such as Label Correction algorithm. We may consider different variants of the Label Corrections such as Breadth First Search, Depth First Search or Best First Search, also known as Dijkstra's algorithm and perhaps search based and sampling based search algorithms such as the A* algorithm and RRT algorithm.

VI. ACKNOWLEDGEMENT

I would like to thank Professor Nikolay Atanasov for his help and suggestions throughout this project. His assistance helped me overcome some crucial problems encountered during the project.

REFERENCES

[1] Nikolay Atanasov, "https://natanaso.github.io/ece276b/ref/ECE276B_2_MC.pdf"
[2] Nikolay Atanasov, "https://natanaso.github.io/ece276b/ref/ECE276B_3_MDP.pdf"
[3] Nikolay Atanasov, "https://natanaso.github.io/ece276b/ref/ECE276B_4_DPA.pdf"
[4] Nikolay Atanasov, "https://natanaso.github.io/ece276b/ref/ECE276B_5_DSP.pdf"

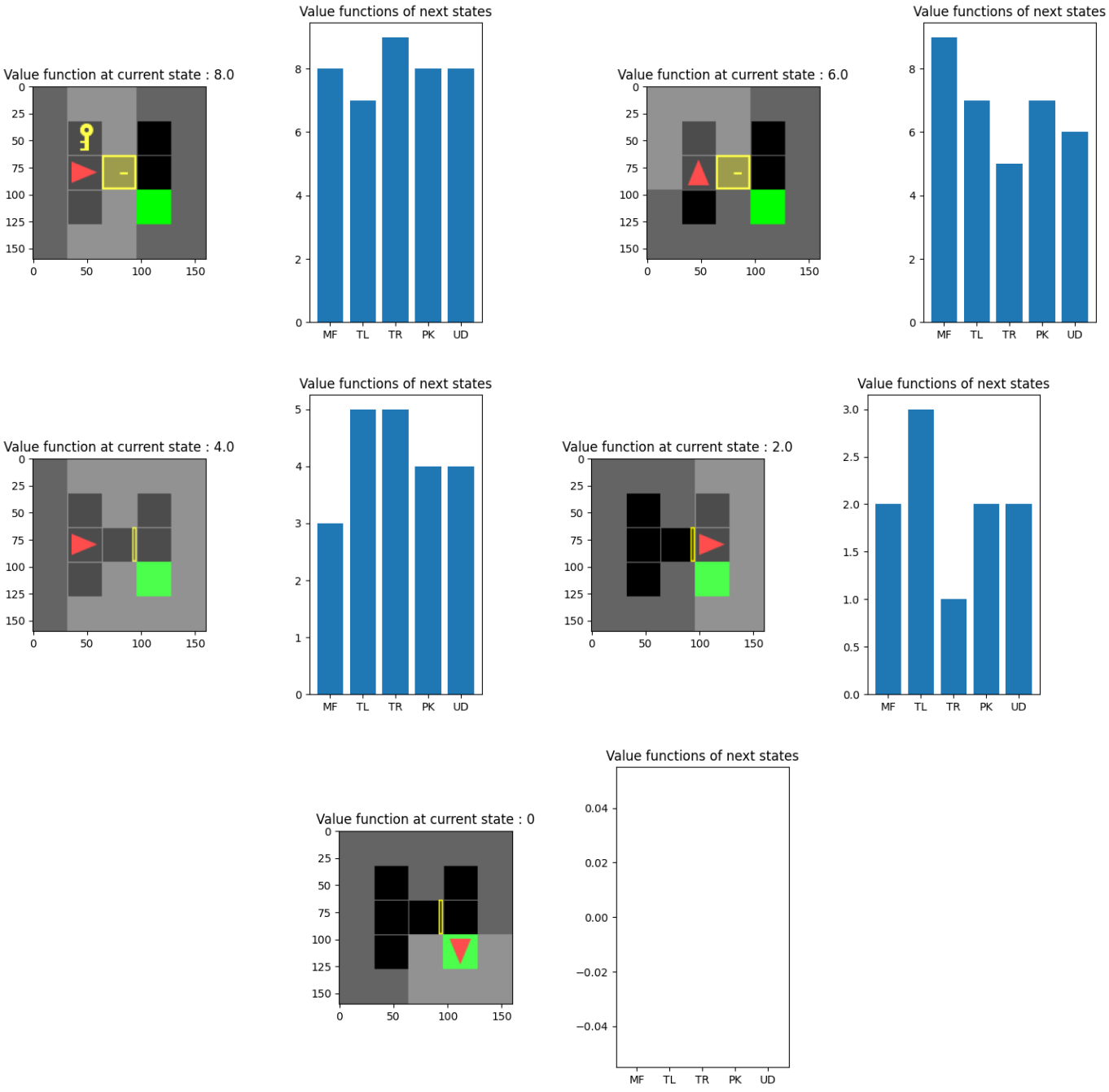

Fig. 2: Shortest Path for doorkey-5x5-normal environment

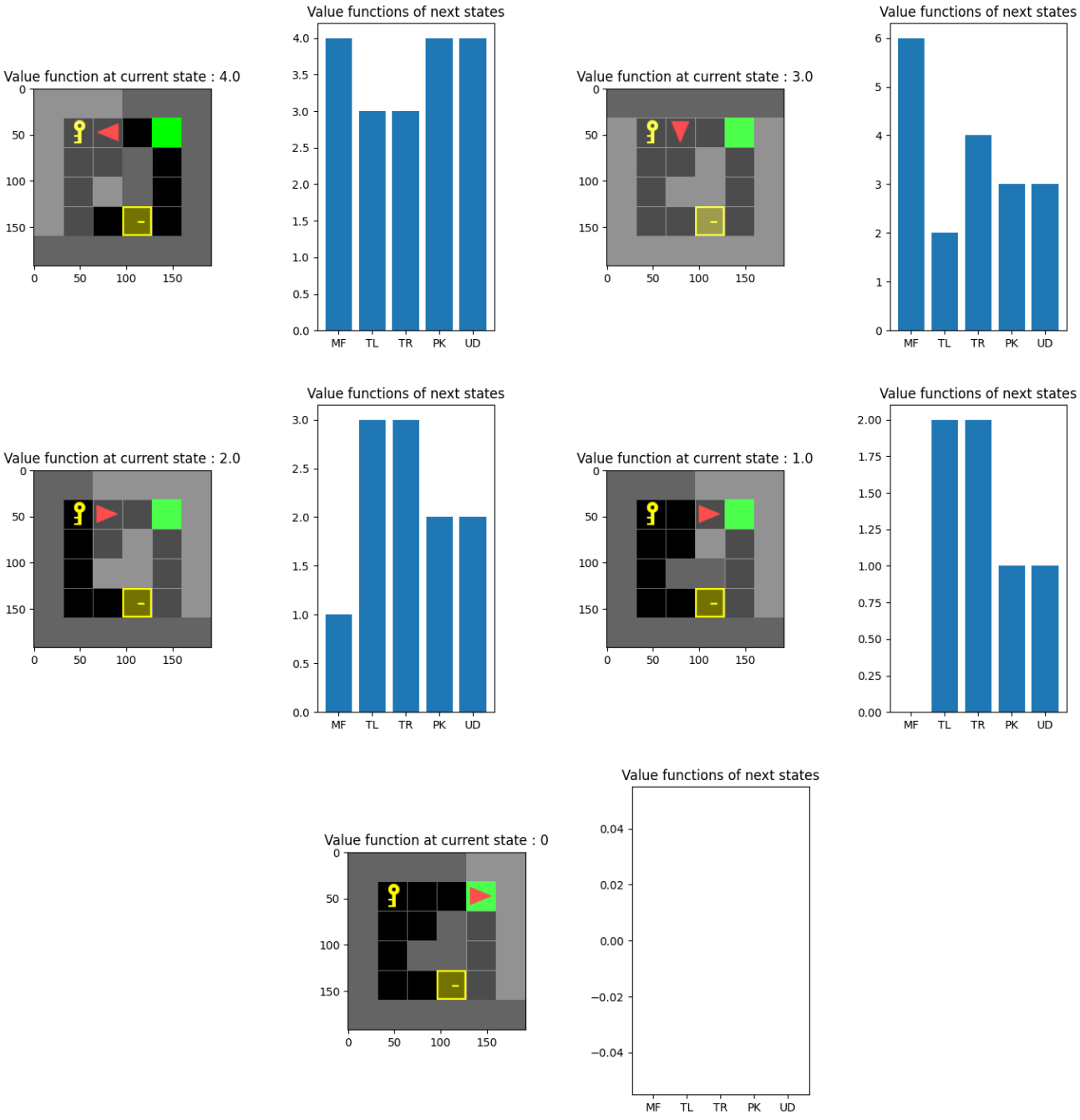

Fig. 3: Shortest Path for doorkey-6x6-direct environment

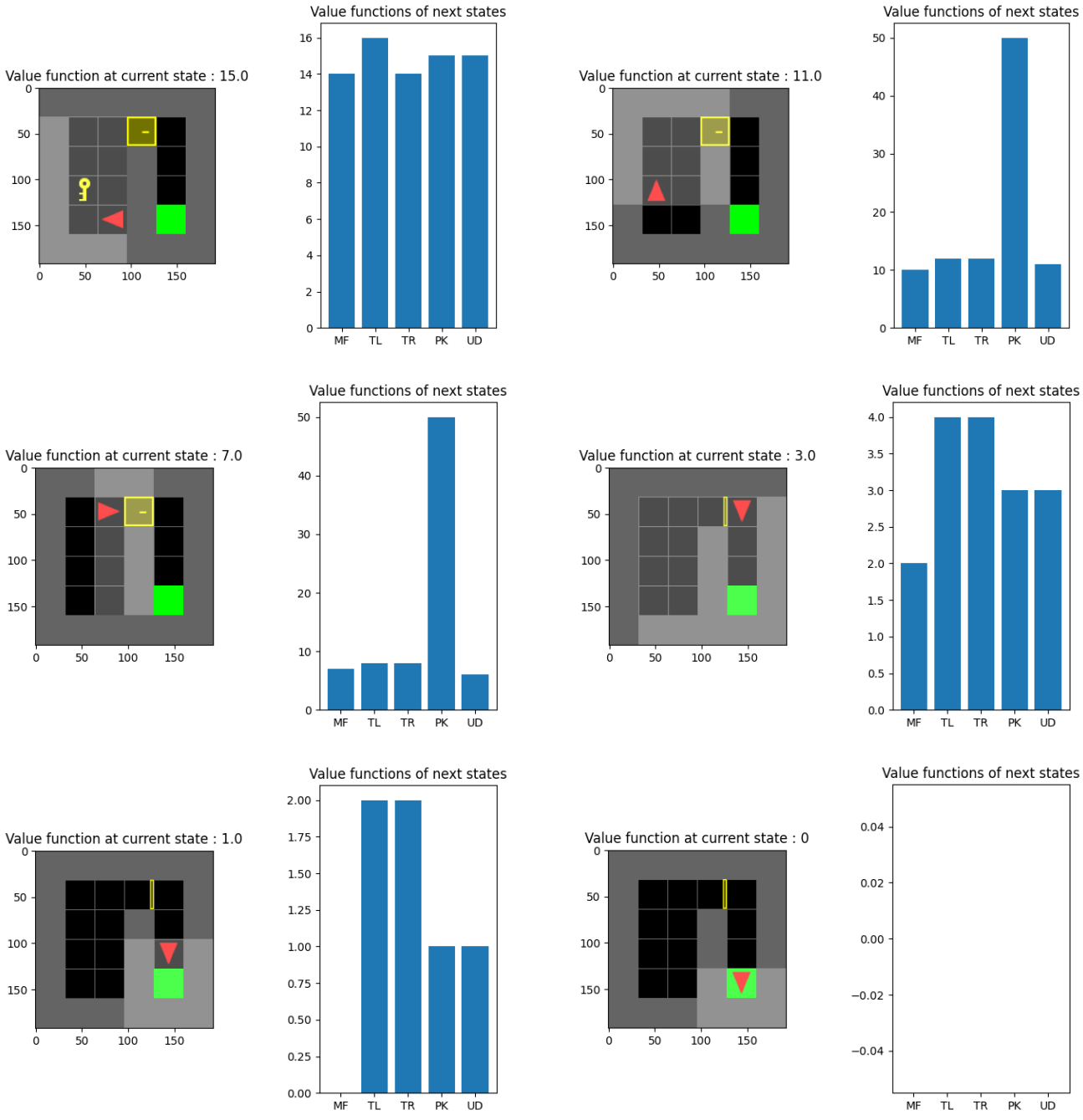

Fig. 4: Shortest Path for doorkey-6x6-normal environment

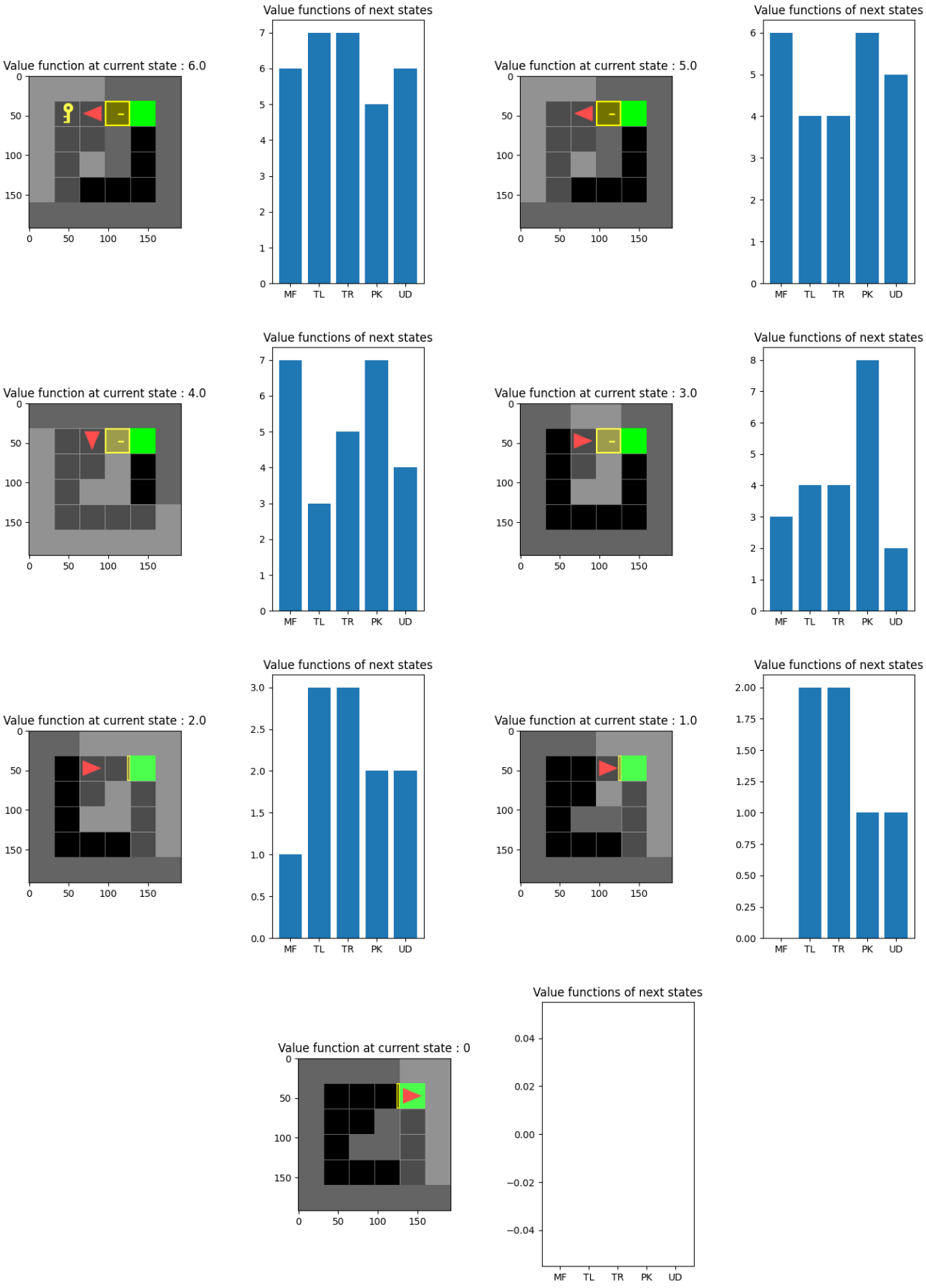

Fig. 5: Shortest Path for doorkey-6x6-shortcut environment

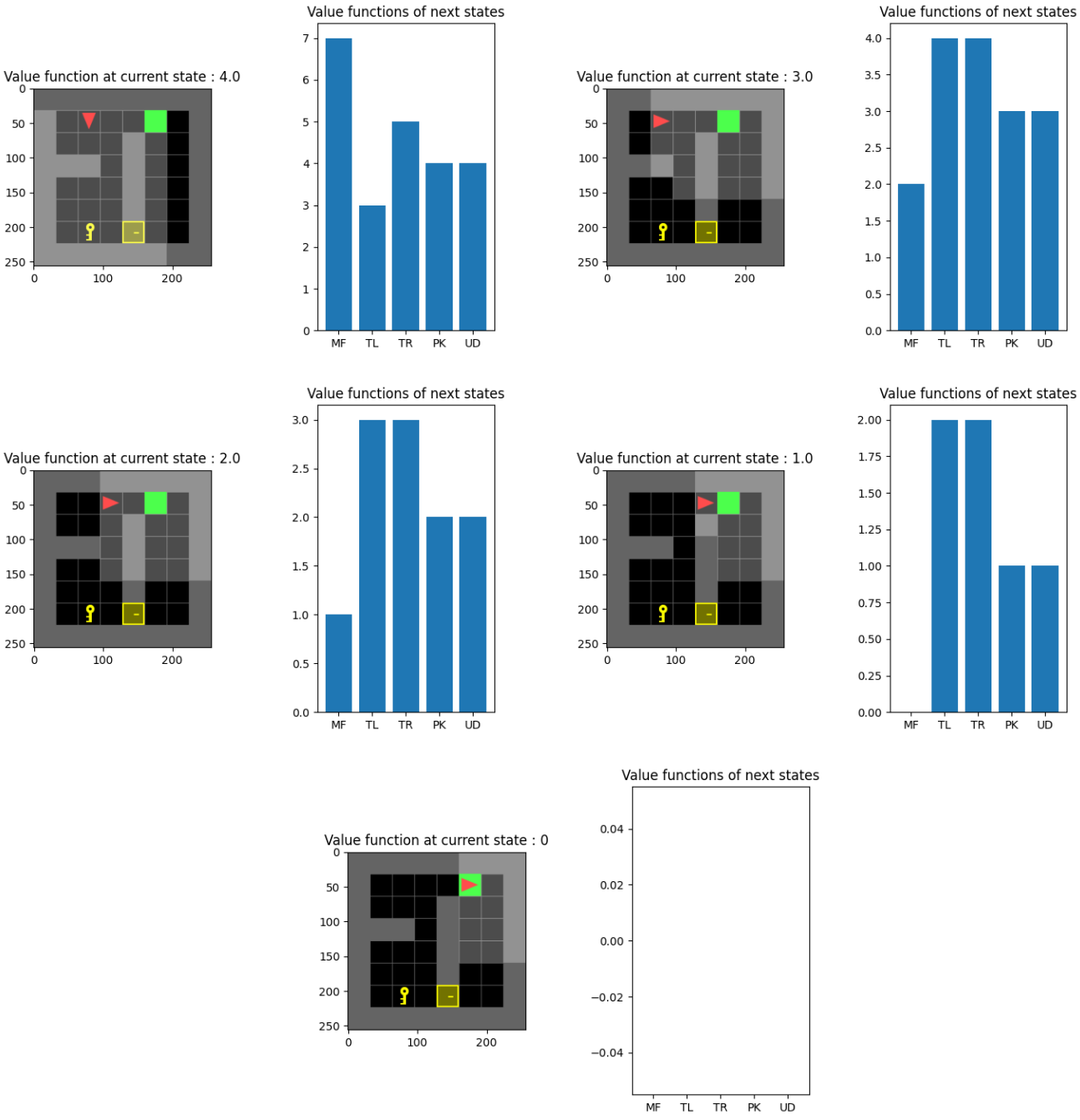

Fig. 6: Shortest Path for doorkey-8x8-direct environment

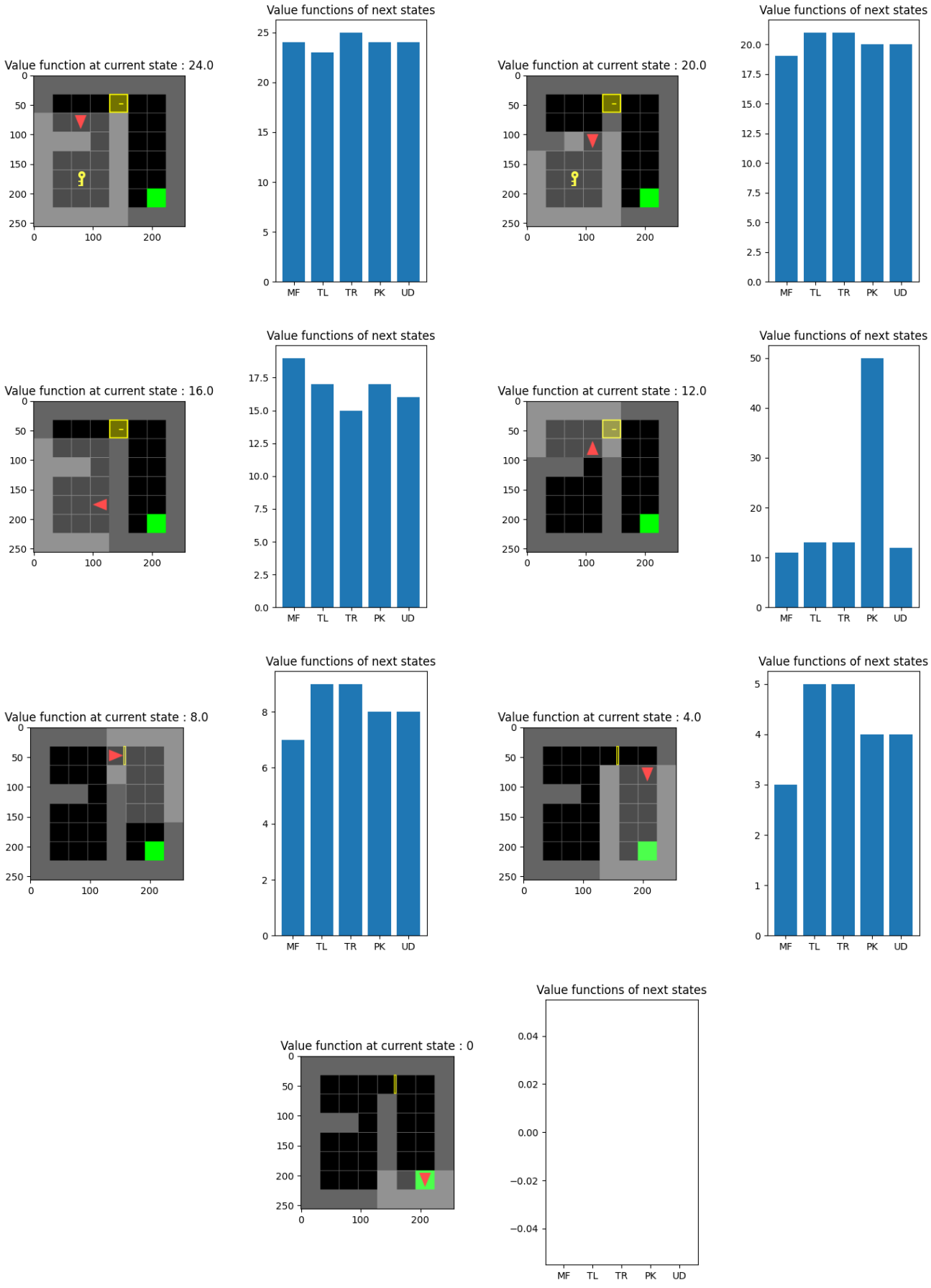

Fig. 7: Shortest Path for doorkey-8x8-normal environment

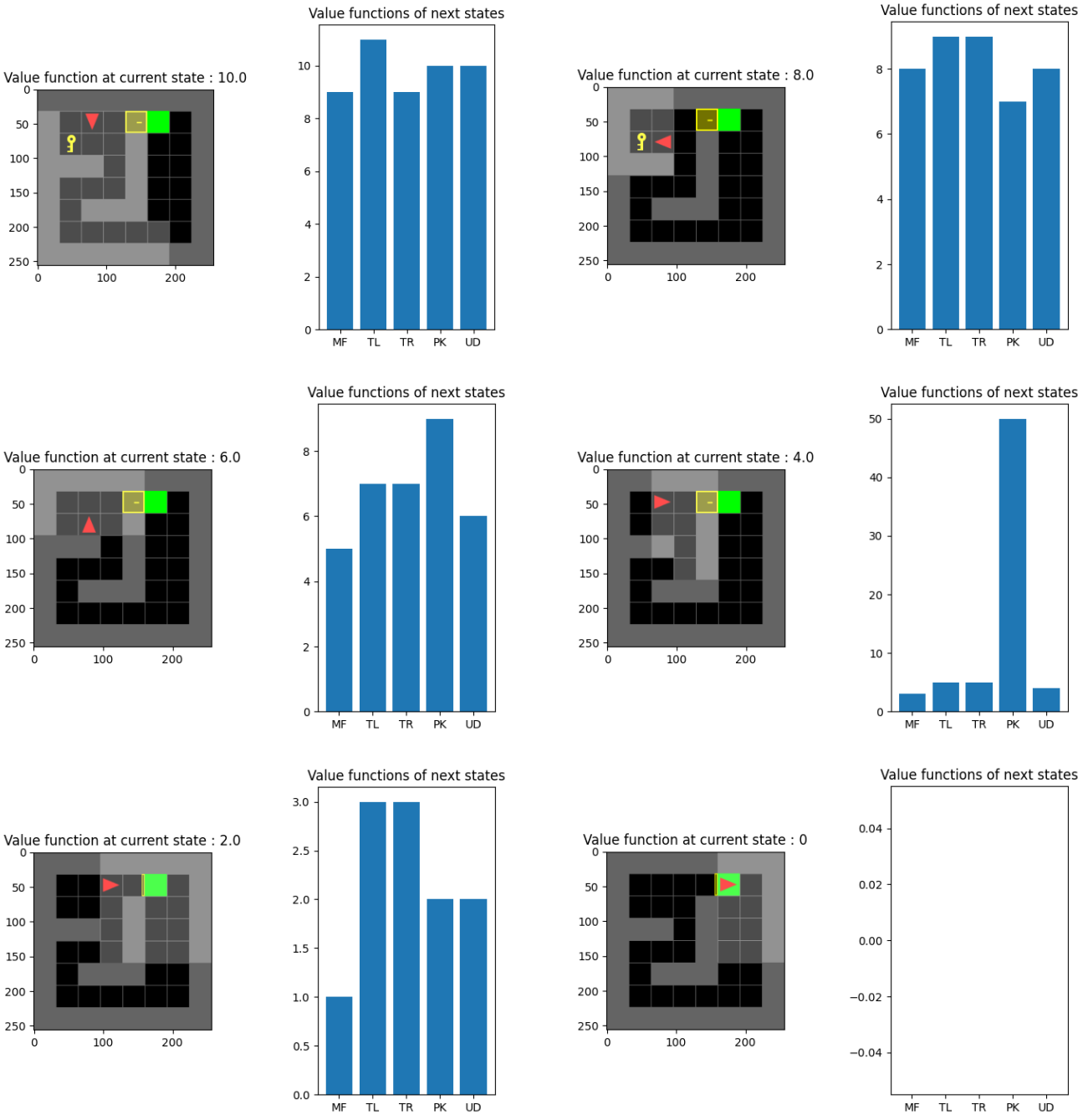

Fig. 8: Shortest Path for doorkey-8x8-shortcut environment

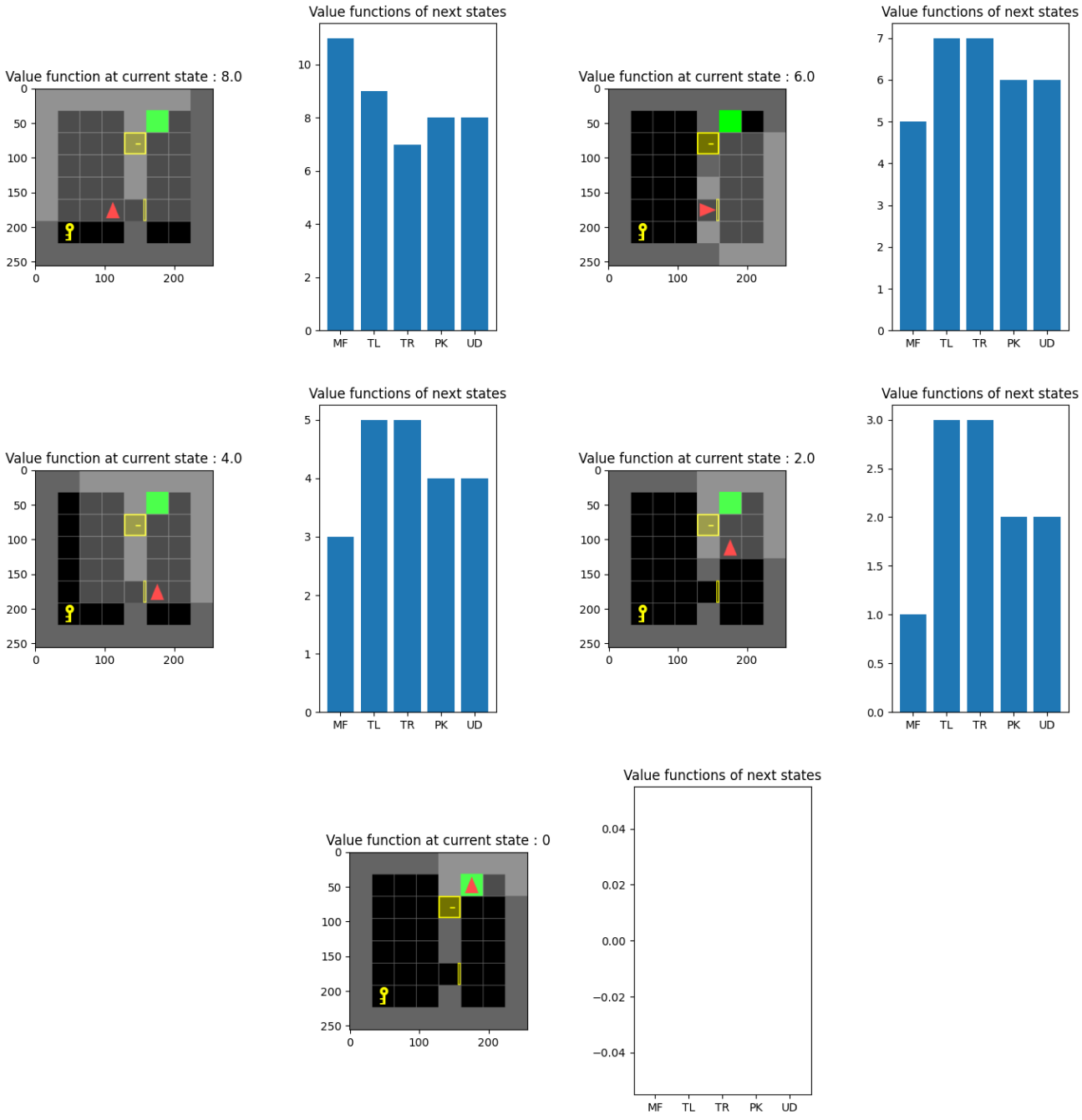

Fig. 9: Shortest Path for sample 1 of part B random environment

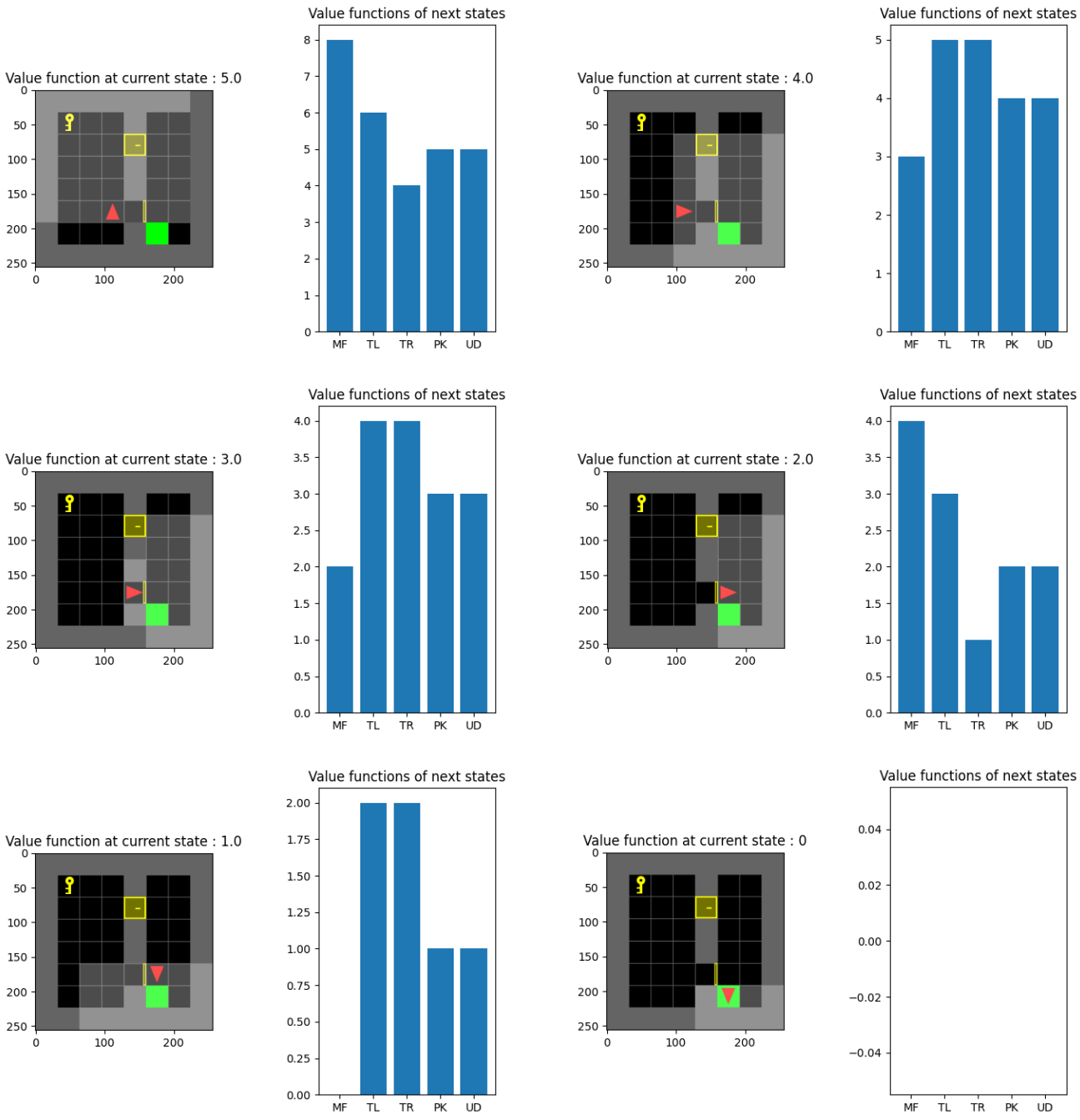

Fig. 10: Shortest Path for sample 2 of part B random environment

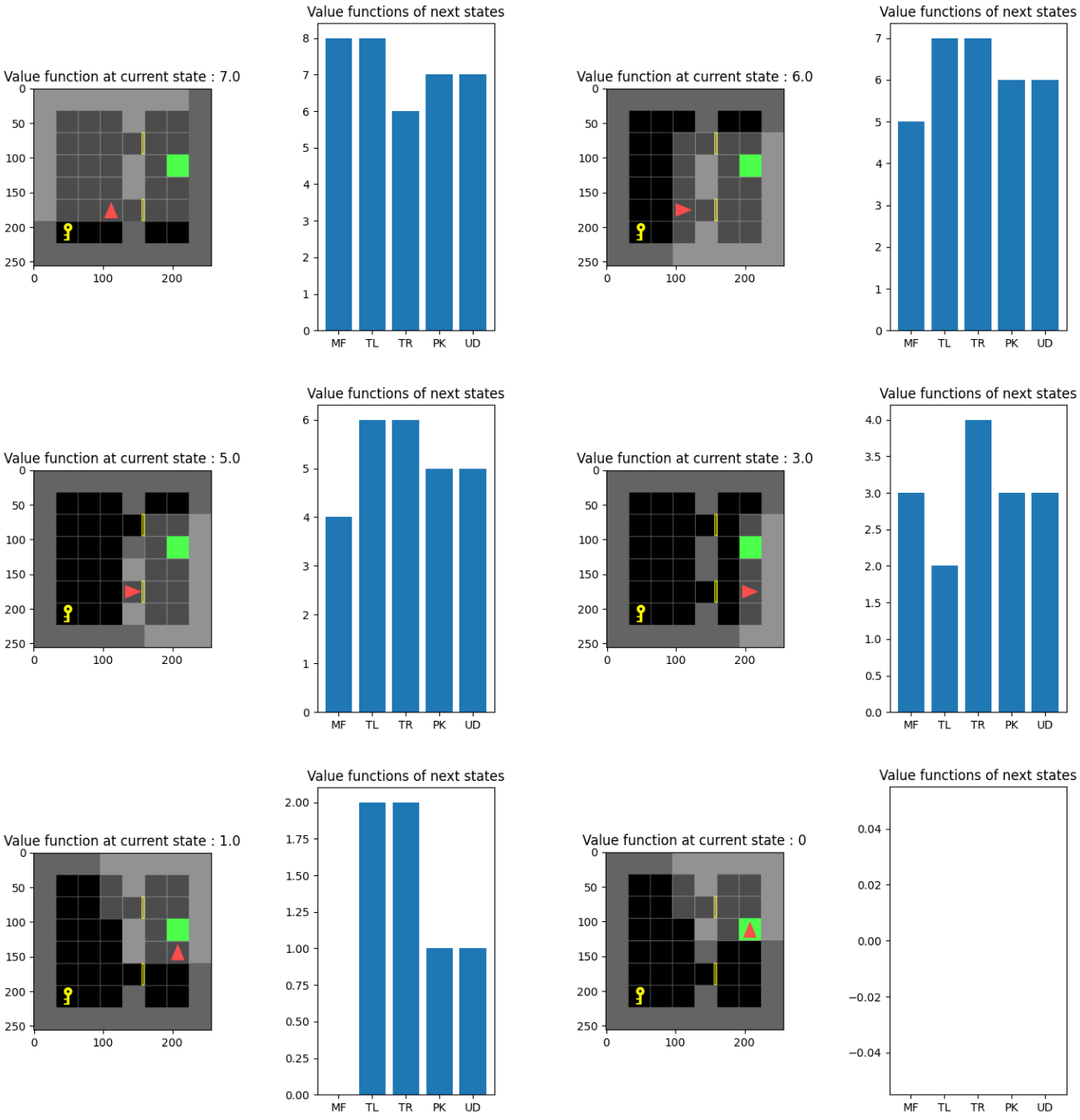

Fig. 11: Shortest Path for sample 3 of part B random environment

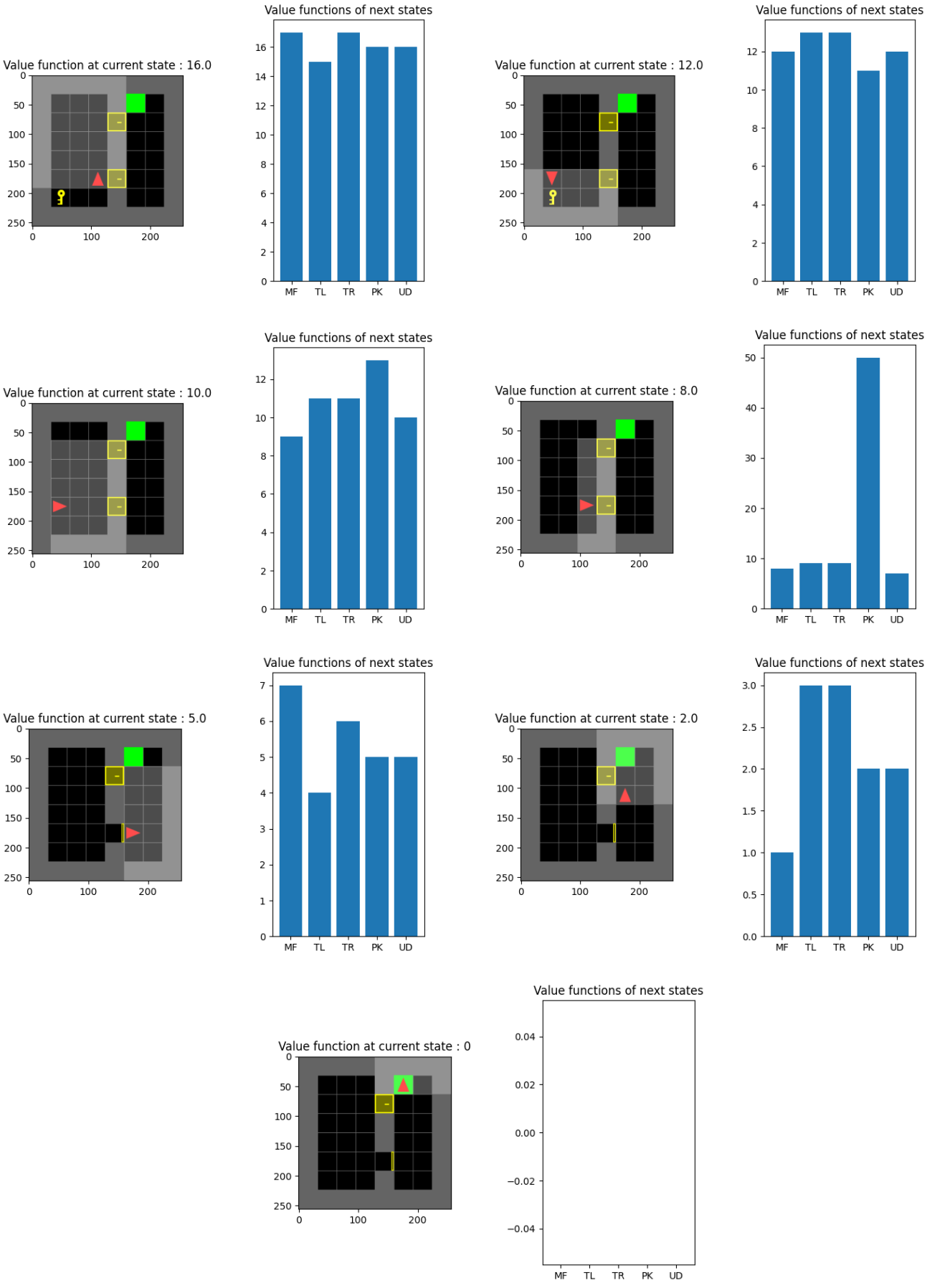

Fig. 12: Shortest Path for sample 4 of part B random environment

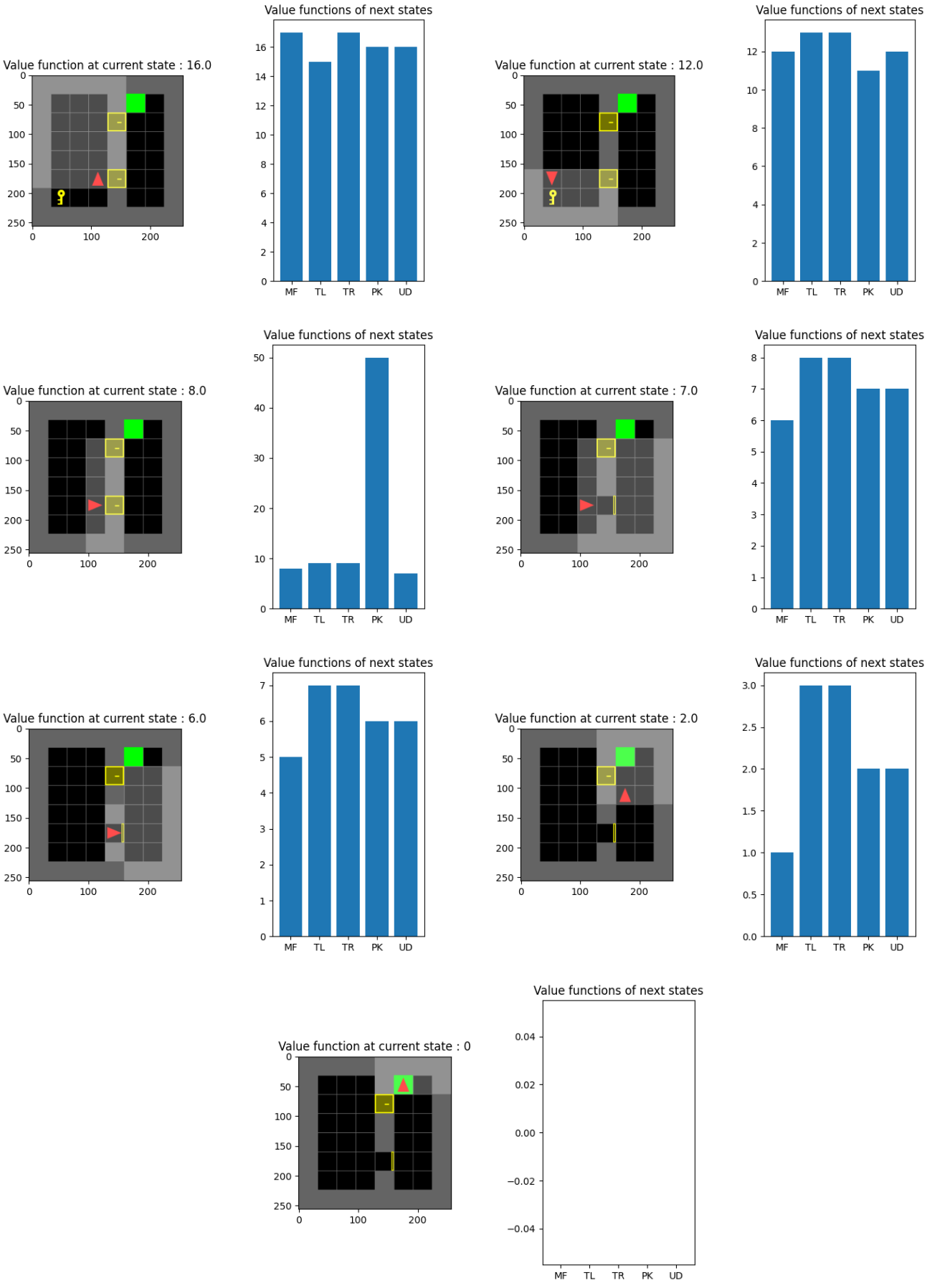

Fig. 13: Shortest Path for sample 5 of part B random environment