ECE 276B PR1: Markov Processes and Dynamic Programming

or Why I Ate Nearly Nothing for a Week

Roumen Guha
Department of Electrical and Computer Engineering
University of California, San Diego
La Jolla, United States
roumen.guha@gmail.com

***Abstract*—This paper presents a particular solution to tackle the problem of determining the shortest path in an environment from START to GOAL. Very briefly, we implemented a label-correcting algorithm, using a priority queue for the set of OPEN nodes. Therefore this is essentially an implementation of Dijkstra's algorithm. The results section contains pictures and discussion concerning the effectiveness of this algorithm.**

***Index Terms*—motion-planning, maze, door, key, markov decision process, dynamic programming algorithm, deterministic finite state, deterministic shortest path, label-correcting algorithm**

I. Introduction

The goal of this project was to obtain the shortest path from an agent's START position to a known GOAL position. In this scenario, we will have a door in the environment that we need to unlock to be able to pass through, and we will need to collect the KEY for this DOOR. However, we there may sometimes be a path to GOAL that does not require us to pass through DOOR. An example environment is shown in Fig. 1.

We consider the environment fully-observable, and hence our agent knows where the GOAL, DOOR, and KEY cells are and can use this information to plan a path through the environment. This is a fairly common setting in robotics, and is considered an open-loop methodology: the control inputs (i.e. the action sequence) is determined before the agent begins moving, and does not change as the agent moves through the space.

A **Markov Decision Process (MDP)** is a Markov Reward Process with predefined transitions. We define an MDP and convert this formulation to a **Deterministic Shortest Path (DSP)** problem and solve this with a **Label-Correcting Algorithm (LCA)**.

In the Problem Formulation section, we will succinctly model the motion-planning problem in mathematical terms. In the Technical Approach section, we will detail the implementation of the DSP, discussing some interesting decisions we made in our implementation. Finally, in the Results section, we will review the path planned by the algorithm and discuss the its performance, and will note how the decisions we made in the implementation affected the outcome we observe.

This project was worked on with no one, which is probably why I don't consider it my best work. Please have mercy.

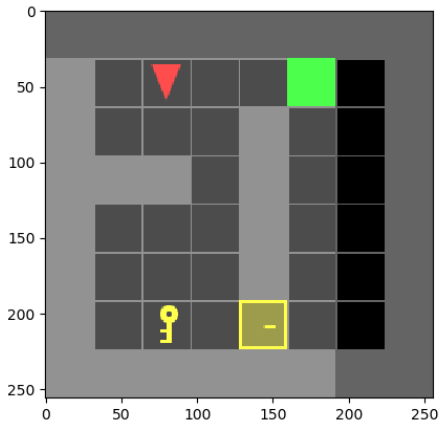


Fig. 1: Example of environment setting (doorkey-8x8-direct).

II. Problem Formulation

In this section, we precisely define the quantities we are interested in. We also cover some necessary knowledge necessary to understand how this algorithm was implemented.

A. Deterministic Finite State

An MDP is a fully-observed problem with controlled transitions between states. Therefore, for this scenario, it can be defined by the tuple $(\mathcal{X}, \mathcal{U}, p_0, p_f, T, l, \mathfrak{q}, \gamma)$ where:

- $\mathcal{X}$ is a discrete set of states
- $\mathcal{U}$ is a discrete set of controls or actions
- p_0 is a probability mass function (PMF) describing the initial state of the robot/agent at time $t = 0$: $\mathbf{x}_0$
- $p_f(\cdot|\mathbf{x}_t, \mathbf{u}_t)$ is distribution corresponding to the motion model $f(\mathbf{x}_t, \mathbf{u}_t)$, represented as a conditional PMF defined on $\mathcal{X}$ for given $\mathbf{x}_t \in \mathcal{X}$ and $\mathbf{u}_t \in \mathcal{U}$. This can be

summarized by a matrix $P^u_{i,j} := p_f(j|\mathbf{x}_t = i, \mathbf{u}_t = u)$ in the finite-dimensional case
- $T \in \mathbb{Z}^+$ is a finite time-horizon
- $l(\mathbf{x}, \mathbf{u})$ is a function specifying the cost/reward of applying control $\mathbf{u} \in \mathcal{U}$ in state $\mathbf{x} \in \mathcal{X}$
- $\mathfrak{q}(\mathbf{x})$ is a terminal cost of being in state $\mathbf{x}$ at time T
- $\gamma \in [0, 1]$ is a discount factor. For our finite time-horizon case, we can take $\gamma = 1$

We can determine, at time $t = 0$, an admissible control policy $\boldsymbol{\pi}$. This is defined as a sequence $\pi_{0:T-1}$ of functions π_t that map a state $\mathbf{x}_t \in \mathcal{X}$ to a feasible control input $\mathbf{u}_t \in \mathcal{U}(\mathbf{x}_t)$.

We can then define the deterministic finite-horizon **value function** $V_0^{\boldsymbol{\pi}}(\mathbf{x}_0)$ (i.e. the cumulative cost of a policy $\boldsymbol{\pi} := \pi_{0:T-1} = \{\pi_0, \ldots, \pi_{T-1}\}$ applied to an MDP with initial state $\mathbf{x_0} \in \mathcal{X}$ at time $t = 0$):

$$V_0^{\boldsymbol{\pi}}(\mathbf{x}_0) := \mathfrak{q}(\mathbf{x}_T) + \sum_{t=0}^{T-1} l(\mathbf{x}_t, \pi_t(\mathbf{x}_t)) \tag{1}$$

which we seek to minimize with an optimal control policy $\boldsymbol{\pi}$:

$$\min_{\boldsymbol{\pi}} V_0^{\boldsymbol{\pi}}(\mathbf{x}_0) = \min_{\boldsymbol{\pi}} \mathfrak{q}(\mathbf{x}_T) + \sum_{t=0}^{T-1} l(\mathbf{x}_t, \pi_t(\mathbf{x}_t)) \tag{2}$$

And hence the optimal open-loop control sequence is precomputed at time $t = 0$:

$$\begin{aligned} \mathbf{u}_{0:T-1} &= \arg\min_{\pi_{0:T-1}} V_0^{\boldsymbol{\pi}}(\mathbf{x}_0) \\ &= \arg\min_{\pi_{0:T-1}} \left\{ \mathfrak{q}(\mathbf{x}_T) + \sum_{t=0}^{T-1} l(\mathbf{x}_t, \pi_t(\mathbf{x}_t)) \right\} \end{aligned} \tag{3}$$

and does not change online depending on $\mathbf{x}_t$.

Since this is optimal control problem with no noise (i.e. deterministic and fully-observed) and finite space space ($|\mathcal{X}| < \infty$), this is therefore a **deterministic finite state** (DFS) optimal control problem:

$$\begin{aligned} \min_{\mathbf{u}_{0:T-1}} \quad & \mathfrak{q}(\mathbf{x}_T) + \sum_{t=0}^{T-1} l(\mathbf{x}_t, \mathbf{u}_t) \\ \text{s.t.} \quad & \mathbf{x}_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t), \quad t = 0, \ldots, T-1, \\ & \mathbf{x}_t \in \mathcal{X}, \\ & \mathbf{u}_t \in \mathcal{U} \end{aligned} \tag{4}$$

which is equivalent to the DSP problem, as we will see next.

B. Deterministic Shortest Path

Given a finite, discrete state-space $\mathcal{X}$ and a finite time-horizon T, we can construct a graph representation of the DFS problem.

To see this, consider a DSP problem with vertex space $\mathcal{V}$, weighted edge space $\mathcal{C}$, start node $s \in \mathcal{V}$, and terminal node $\tau \in \mathcal{V}$.

We can then formulate the DSP problem as a DFS with $T := |\mathcal{V}| - 1$ stages. Then:
- State space: $\mathcal{X} = \mathcal{V}$
- Control space: $\mathcal{U} = \mathcal{V}$
- Motion model: $x_{t+1} = f(\mathbf{x}_t, \mathbf{u}_t)$
- Stage costs: $l(\mathbf{x}_t, \mathbf{u}_t)$
- Terminal cost: $\mathfrak{q}(\mathbf{x})$

where

$$f(\mathbf{x}_t, \mathbf{u}_t) = \begin{cases} \mathbf{x}_t & \text{if } x_t = \tau \\ \mathbf{u}_t & \text{else} \end{cases} \tag{5}$$

$$l(\mathbf{x}_t, \mathbf{u}_t) = \begin{cases} 0 & \text{if } x_t = \tau \\ c_{\mathbf{x}_t, \mathbf{u}_t} & \text{else} \end{cases} \tag{6}$$

and

$$\mathfrak{q}(\mathbf{x}) = \begin{cases} 0 & \text{if } x_t = \tau \\ \infty & \text{else} \end{cases} \tag{7}$$

We also make the following assumption: there exist **no negative cycles** in the graph. That is, the optimal path need not have more than $|\mathcal{V}|$ elements. Since the solution to the DSP problem requires a minimization, any negative transition costs would confuse our algorithm.

III. Technical Approach

In this section, we discuss the algorithms implemented in this project.

We did not solve the problem of the shortest path from START to GOAL, but instead solved two simpler subproblems: START to KEY, and KEY to GOAL. The logic here is is that once the agent has the key, the door is traversable, and hence once we are at the KEY we simply need to focus on getting to the GOAL. This leads to sub-optimal solutions, but in most situations not much worse (except for the 'doorkey-8x8-normal' environment, shown in Fig. 2. We will discuss this more in Section IV.

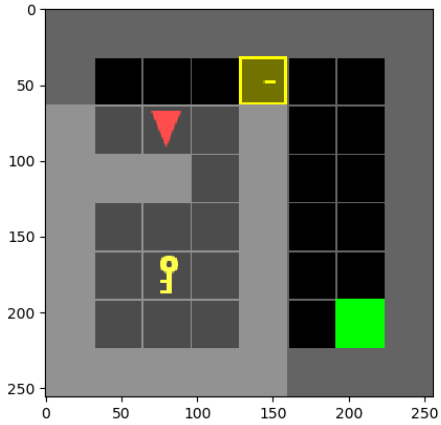


Fig. 2: Worst result obtained was doorkey-8x8-normal.

A. Deterministic Finite State

We begin by defining the states in the state-space.

In this scenario, the state of the agent is defined by the following:

- Position (j, i)
- Orientation $\theta \in \{0, 90, 180, 270\}$
- Flag for KEY
- Flags for DOOR

However, because the state space would be $|\mathcal{V}| = N \times M \times 4 \times 3$, where 4 refers to the number of orientations, and 3 refers to the number of flags, we make some simplifying assumptions. Namely, that we can recreate the changes in orientation from the optimal node sequence returned from the LCA algorithm, and therefore do not to include it in the state description. In this way, we can reduce the number of states in the state space by a factor of 4. This will prove to be a problematic statement, as we shall soon see in Section IV.

Further, we set the cost $c_{\mathbf{x}_t, \mathbf{u}_t}$ in 6 to be 1 for any move forward (MF), and to be 0 for all other actions, including turning left (TL), turning right (TR), pickup key (PK), and unlock door (UD).

B. Deterministic Shortest Path

The label-correcting algorithm is given below:

1) OPEN $\leftarrow \{s\}$
2) $g_s = 0, g_i = \infty \quad \forall i \in \mathcal{V} \setminus \{s\}$
3) While OPEN is not empty:
   a) Remove i from OPEN
   b) for $j \in$ Children(i)
      i) if $(g_i + c_{i,j}) < g_j$ and $(g_i + c_{i,j}) < g_\tau$
         A) $g_j = (g_i + c_{i,j})$
         B) Parent(j) $\leftarrow i$
         C) if $j \neq \tau$ then OPEN $\leftarrow$ OPEN $\cup\{j\}$, else done

We implemented OPEN as a priority queue that pops the node in OPEN with the minimum label value. This means that my implementation of the label-correcting algorithm is equivalent to **Dijkstra's Algorithm**.

IV. Results

A. Discussion

One of the issues with splitting the problem into two sub-problems is that our agent is forced to enter the KEY's cell after picking up the key. We were able to generate an action sequence from the shortest sequence of nodes that avoided this problem, and for the provided example environment generated an identical action sequence as the example action sequence. However the agent did not move in the same path that the label-correcting algorithm dictated it should for some other environments, and in the case of the environment 'doorkey-6x6-normal', the agent failed completely to reach the DOOR, much less the GOAL.

The action sequence we include in this report and generate in our implementation is therefore less optimal than it could have been, but it is guaranteed to reach the GOAL eventually,

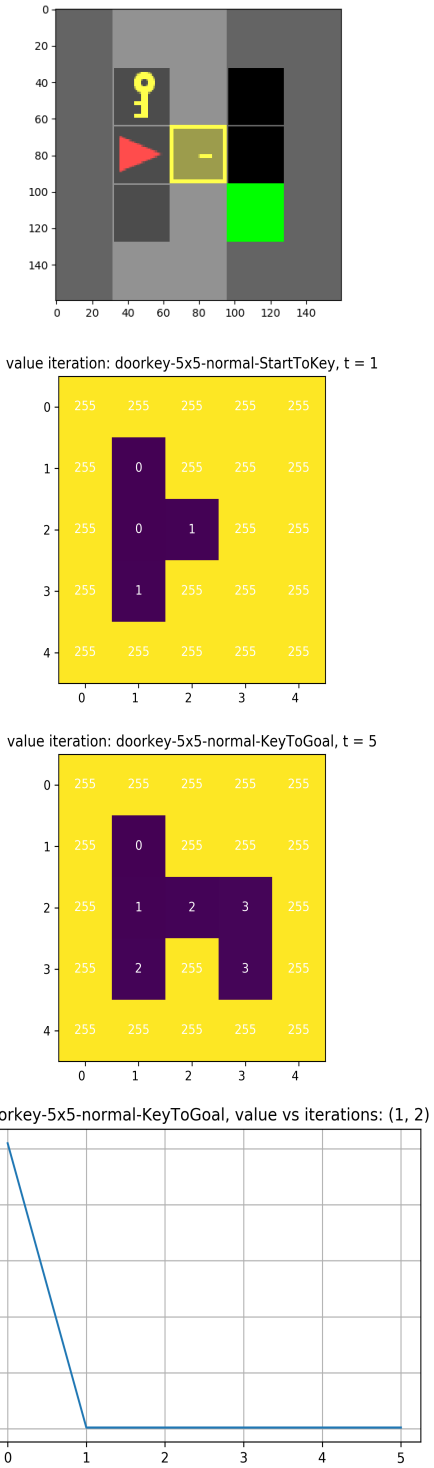


Fig. 3: Result for doorkey-5x5-normal environment. Optimal path: (2, 4) to (2, 5) to (2, 6) to (2, 6) to (2, 5) to (3, 5) to (4, 5) to (5, 5) to (6, 5) to (6, 6). Optimal action sequence: TL, MF, PK, MF, TL, TL, MF, TR, UK, MF, MF, MF, MF, TR, MF. Maze animation, heat-map animation, and value-curves for more states of interest submitted with code.

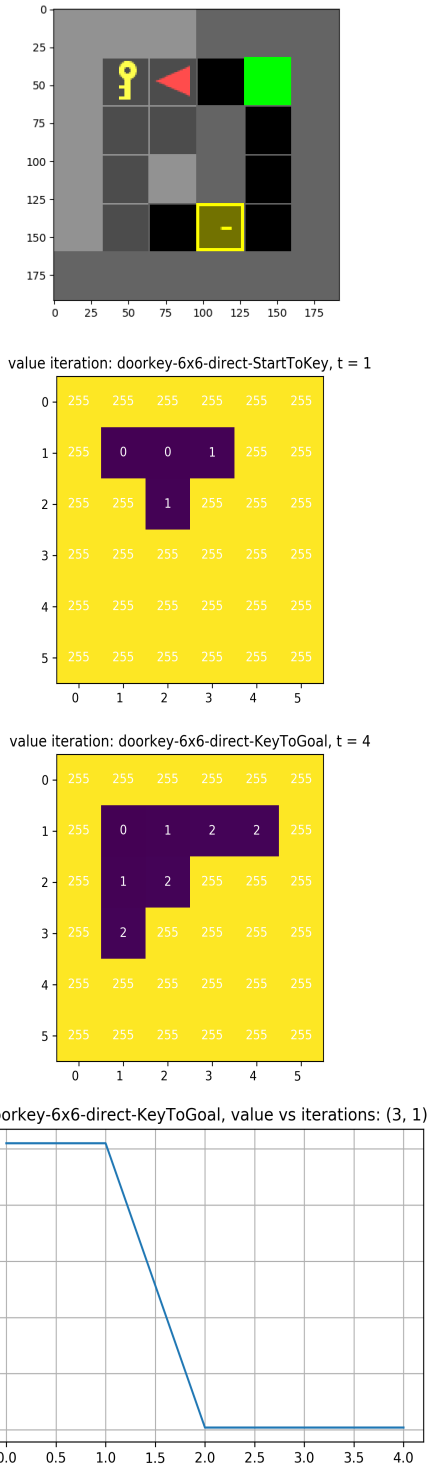
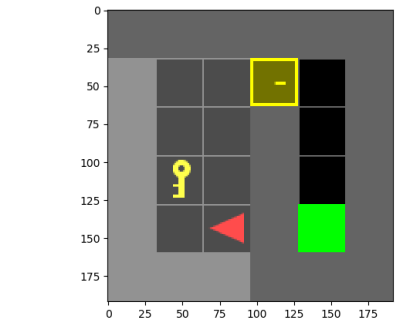
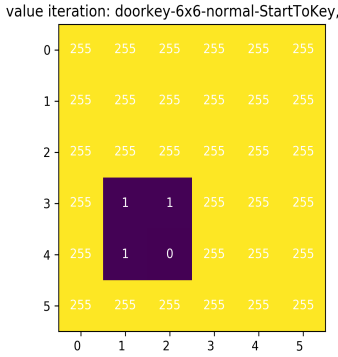
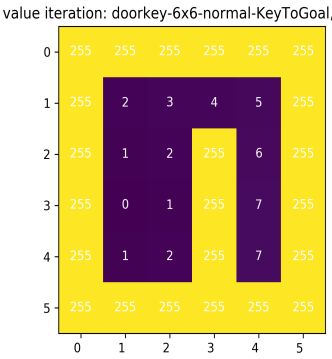


Fig. 4: Result for doorkey-6x6-direct environment. Optimal path: (2, 1) to (1, 1) to (1, 1) to (2, 1) to (3, 1) to (4, 1). Optimal action sequence: PK, MF, TL, TL, MF, MF, MF. Maze animation, heat-map animation, and value-curves for more states of interest submitted with code.

and in most cases, it is able to do this with ϵ-optimality. We suspect we would be able to resolve this issue if we had slightly more time or slightly more brainpower, but nothing in life is ever perfect. We included these in the zip file 'gifs-incorrect.7z' in case it is of interest to the reader.







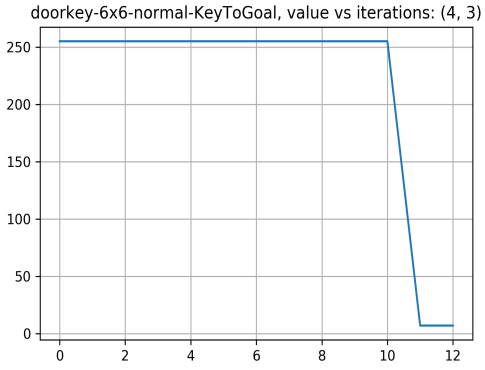
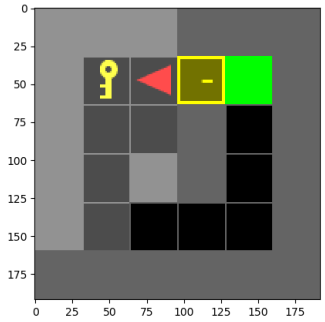
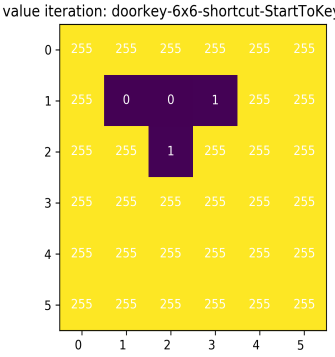
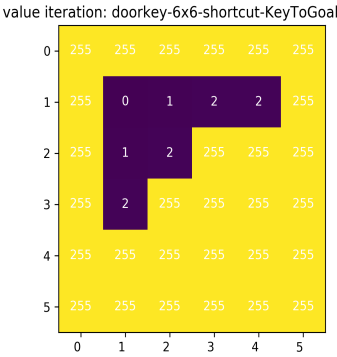


Fig. 5: Result for doorkey-6x6-normal environment. Optimal path: (2, 4) to (1, 4) to (1, 3) to (1, 3) to (2, 3) to (2, 2) to (2, 1) to (3, 1) to (4, 1) to (4, 2) to (4, 3) to (4, 4). Optimal action sequence: MF, TR, PK, MF, TR, MF, TL, MF, MF, TR, UK, MF, MF, TR, MF, MF, MF. Maze animation, heat-map animation, and value-curves for more states of interest submitted with code.







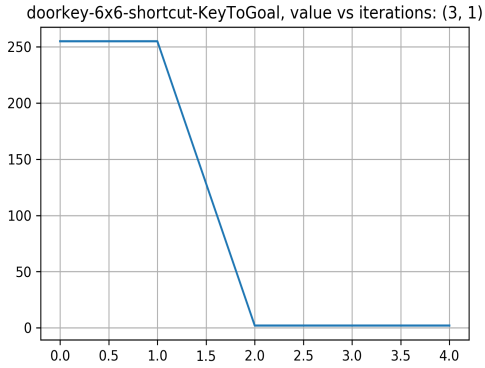
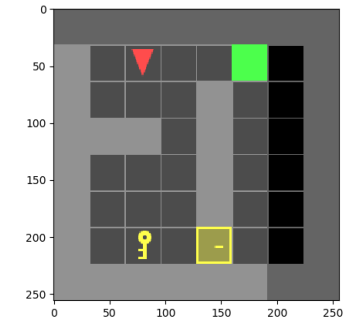
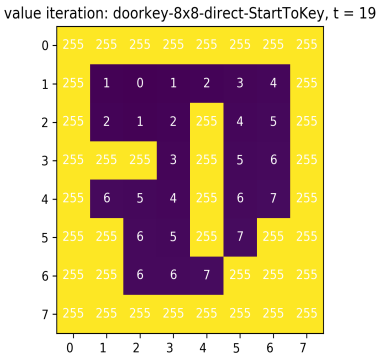
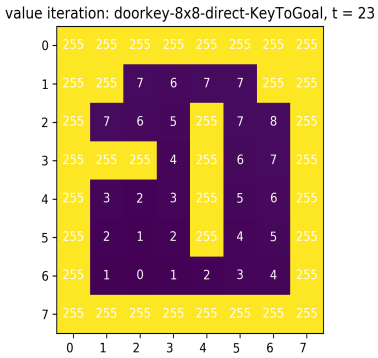


Fig. 6: Result for doorkey-6x6-shortcut environment. Optimal path: (2, 1) to (1, 1) to (1, 1) to (2, 1) to (3, 1) to (4, 1). Optimal action sequence: PK, MF, TL, TL, MF, UK, MF, MF. Maze animation, heat-map animation, and value-curves for more states of interest submitted with code.







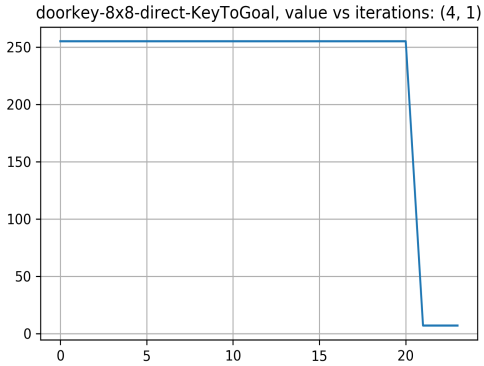
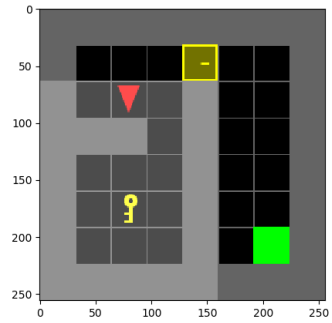
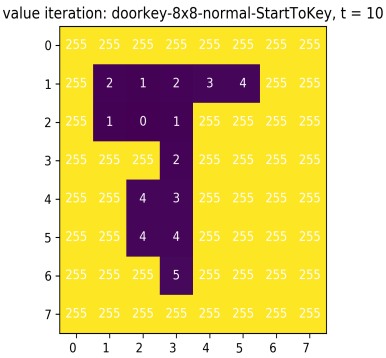
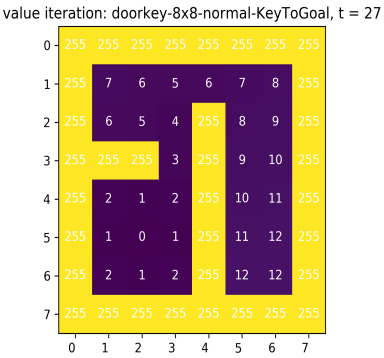


Fig. 7: Result for doorkey-8x8-direct environment. Optimal path: (2, 1) to (3, 1) to (3, 2) to (3, 3) to (3, 4) to (3, 5) to (3, 6) to (2, 6) to (2, 6) to (3, 6) to (4, 6) to (5, 6) to (5, 5) to (5, 4) to (5, 3) to (5, 2) to (5, 1). Optimal action sequence: TL, MF, TR, MF, MF, MF, MF, MF, TR, PK, MF, TL, TL, MF, UK, MF, MF, TL, MF, MF, MF, MF, MF. Maze animation, heat-map animation, and value-curves for more states of interest submitted with code.







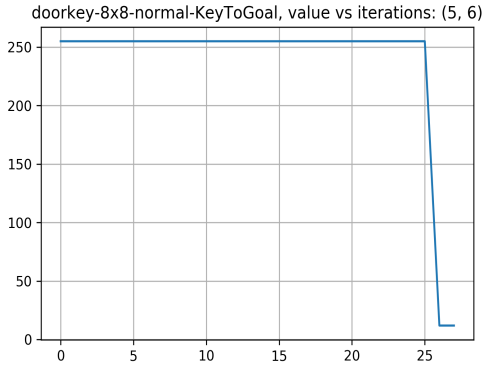
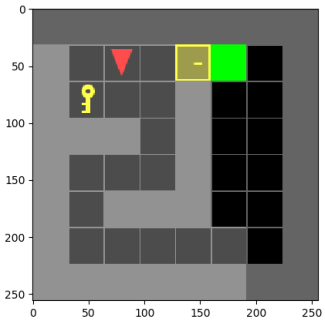
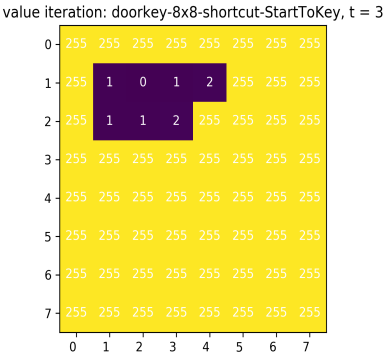
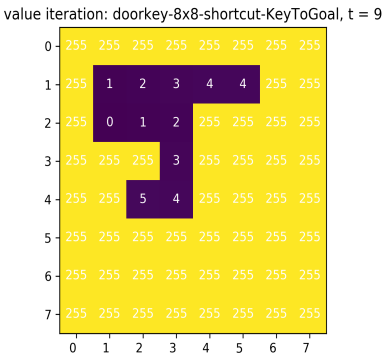


Fig. 8: Result for doorkey-8x8-normal environment. Optimal path: (2, 2) to (3, 2) to (3, 3) to (3, 4) to (3, 5) to (2, 5) to (2, 5) to (3, 5) to (3, 4) to (3, 3) to (3, 2) to (3, 1) to (4, 1) to (5, 1) to (6, 1) to (6, 2) to (6, 3) to (6, 4) to (6, 5) to (6, 6). Optimal action sequence: TL, MF, TR, MF, MF, MF, TR, PK, MF, TL, TL, MF, TL, MF, MF, MF, MF, TR, UK, MF, MF, MF, TR, MF, MF, MF, MF, MF. Maze animation, heat-map animation, and value-curves for more states of interest submitted with code.







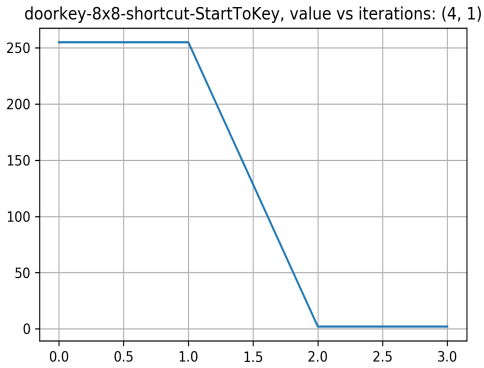


Fig. 9: Result for doorkey-8x8-shortcut environment. Optimal path: (2, 1) to (2, 2) to (1, 2) to (1, 2) to (2, 2) to (3, 2) to (3, 1) to (4, 1) to (5, 1). Optimal action sequence: MF, TR, PK, MF, TL, TL, MF, MF, TL, MF, TR, UK, MF, MF. Maze animation, heat-map animation, and value-curves for more states of interest submitted with code.