Markov Processes

Saurabh Himmatlal Mirani (A53319557)
Planning & Learning in Robotics
University of California, San Deigo
smirani@ucsd.edu

I. Introduction

In robotics, we need to solve many shortest path problems. For example, in path planning, to reach to the goal in minimum amount of time or to use minimum fuel, we should find the shortest path to reach the goal quickly.

The path planning problem is one of the most important problems in autonomous robotics. Given the prior information of the environment and knowledge about robot motion capabilities, a path planning problem is then reduced to optimization problem where we want to find the best path under a set of constraints.

The problem of the robot to reach goal in the least possible steps is defined as shortest path problem. This problem doesn't have noise and hence is reduced to Deterministic Shortest Path problem which is then solved using a Label correction algorithm. The rest of the paper is organised as: Section II formulates the problem, Section III desrcibes the technical approach, Section IV has the results and discussions and Section V concludes the paper.

II. Problem Formulation

Consider a Deterministic Shortest Path (DSP) problem with space V, where $V = (x, y, carryingkey, doorstatus)$, weighted edge space C start node $s = (x_o, y_o, carryingkey_o, doorstatus_o) \in V$ and terminal node $\tau = (x_\tau, y_\tau, --, --) \in V$

$$carryingkey = \begin{cases} 0 & if\ not\ carrying\ key \\ 1 & if\ carrying\ key \end{cases}$$

$$doorstatus = \begin{cases} 0 & if\ closed \\ 1 & if\ open \end{cases}$$

Assumption: the optimal path need not have more than $|V|$ elements

Possible controls are: control space U

- Left (-1,0,0,0)
- Right (1,0,0,0)
- Up (0,-1,0,0)
- Down (0,1,0,0)
- Pickup Key (0,0,1,0)
- Unlock Door (0,0,0,1)

We can formulate the DSP problem as a DFS with $T := |V| - 1$ stages:

- State space $X = V$, control space U
- Motion model

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

- Costs:

$$l(x_t, u_t) = \begin{cases} 0 & if\ x_t = \tau \\ 1 & otherwise \end{cases}$$

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

- Planning horizon T: Form mxn grid, T = $m*n*2*2 = 4mn$

Path: a sequence $i_{1:q} = (i_1, i_2, ..., i_q)$ of nodes $i_k \in V$

All paths from $s \in V$ to $\tau \in V$: $\mathbb{I}_{s,\tau} = \{i_{1:q} | i_k \in V, i_1 = s, i_q = \tau\}$

Path length: sum of the arc lengths over the path: $J^{i_{1:q}} = \sum_{k=1}^{q-1} 1 = \frac{q*(q-1)}{2}$

Objective: find a path $i^*_{1:q} = \underset{i_{1:q} \in \mathbb{I}_{s,\tau}}{\arg\min} J^{i_{1:q}}$ that has the smallest length from node $s \in V$ to $\tau \in V$

III. Technical Approach

The problem defined in Section II can be solved using Label Correcting Algorithm.

Algorithm 3 Label Correcting Algorithm

```
OPEN ← {s}, g_s = 0, g_i = ∞ for all i ∈ V \ {s}
while OPEN is not empty do
    Remove i from OPEN
    for j ∈ Children(i) do
        if (g_i + c_ij) < g_j and (g_i + c_ij) < g_τ then    ▷ Only when c_ij ≥ 0 for all i, j ∈ V
            g_j = g_i + c_ij
            Parent(j) = i
            if j ≠ τ then
                OPEN = OPEN ∪{j}
```

There is freedom in selecting the node to be removed from OPEN (set of nodes that can potentially be part of the shortest path) at each iteration, which gives rise to several different methods, one of which is Breadth-first search (BFS) (Bellman-Ford Algorithm)

Breadth-first search (BFS) (Bellman-Ford Algorithm) is used to determine the shortest path. This algorithm employs a "first-in,first-out" policy with OPEN implemented as a queue.

The cost to goal from each possible state of the robot is determined using BFS algorithm. First, the weight if the goal is initialized to zero and other states are initialized to infinity. Cost is assumed to be one unit for each direction hence, all the surrounding free states are updated with cost one. Then these nodes are added to the open node list (queue) and the surrounding nodes to these are update with cost two if it is minimum. This is carried on till all possible states are visited.

BFS is used to calculate the the cost to pickup key, to reach the door and to reach the goal from door

Algorithm 1: Breadth-first search (BFS) Pseudocode

```
Input: A graph Graph and a starting vertex root of
       Graph
Output: Goal state. The parent links trace the shortest
        path back to root
initialization;
while Q is not empty do
    v := Q.dequeue();
    if v is the goal then
        return v;
    end
    forall edges from v to w in G.adjacentEdges(v) do
        if w is not labeled as discovered then
            label w as discovered;
            w.parent := v;
            Q.enqueue(w);
        end
    end
end
```

To find the shortest path from start to goal, the problem is divided into multiple sub-problems.

- If robot is not carrying key and door is closed:
  - Find shortest paths to all possible pickup positions
  - Find shortest paths from all possible pickup positions to unlock door position
  - Find the best pickup position, which minimizes the total cost i.e both from start to key pickup position and from key pickup position to unlock door position
  - Find the shortest path from unlock door position to goal
- If the robot is carrying key and door is closed
  - Find the shortest path from start to door and door to goal
- Find the shortest path from start to goal (cases include:)
  - Direct path exists
  - Door is open
- Find the minimum cost of all possible paths, this gives the optimal path

The obtained path is then converted into optimal sequence by splitting the control inputs, Move right = TR + MF, Move left = TL + MF, Move back = TL + TL + MF. This depends on the robot's current orientation and hence current position, current orientation and next position is used to compute the optimal sequence.

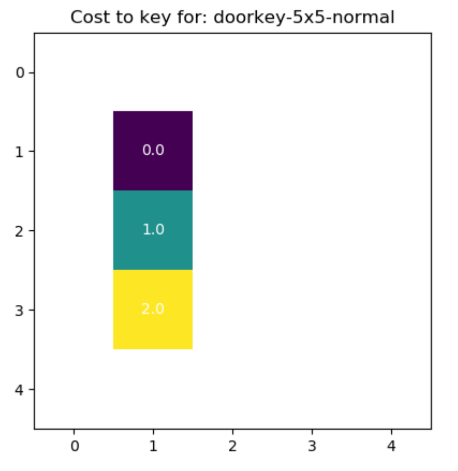
IV. Results and Discussions

Explaination of figures:

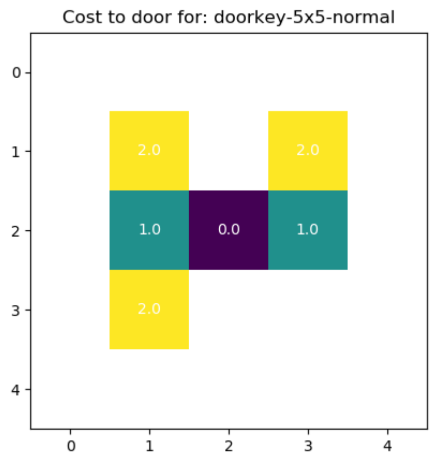
- Cost to XYZ: This figure represents the cost from any possible point in the grid to XYZ. Since, some points do not make sense, for eg cost to key from locked room is infinity and hence appears as white. Similarly if cost from some point to XYZ doesn't exist or is infinity it is displayed as zero.
- - Policy to key: This figure represents policy from any possible points to key pickup position (Pickup positions are colored as blue), in case the robot is not carrying key and wants to pickup key
  - Policy to door: This figure represents policy from all possible points to unlock door position (pink or purple in color) The door position is colored yellow
  - Policy to goal: This has two cases open or closed door. Closed door doesn't always make sense hence it is shown only where it makes sense (direct and shortcut cases). Goal approach positions are dark green in color.

  If arrows are missing in grid it means policy doesn't exist at that position or doesn't make sense. Note: Door can be approached from both the rooms hence policy exists in both the rooms. However it can be unlocked only from unlock position which can be reached only through room where key exists.
- Value Function: Some lines overlap each other and may result in poor visibility. However, important lines are clearly visible in all graphs. refer legend of graph for more information

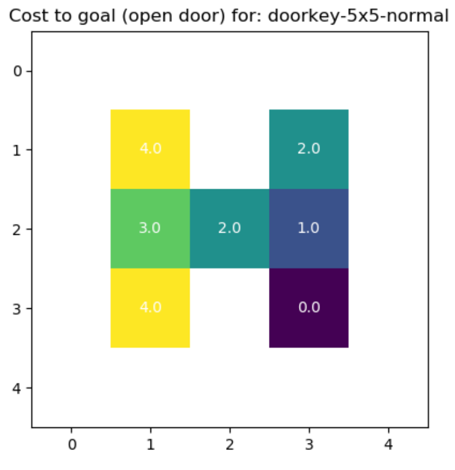
A. Doorkey-5x5-normal



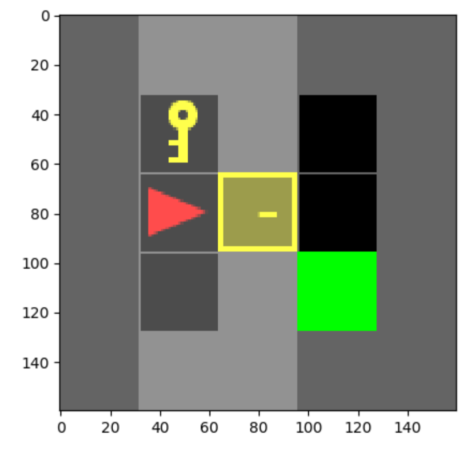
(a)



(b)

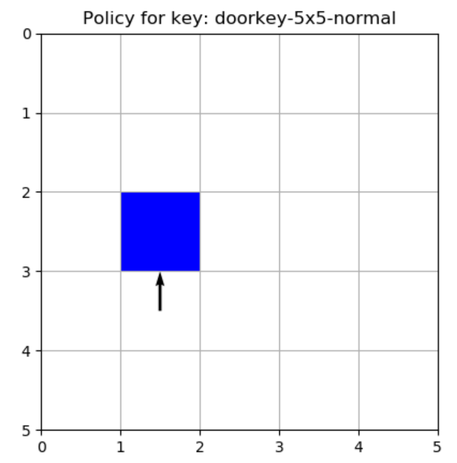


(c)

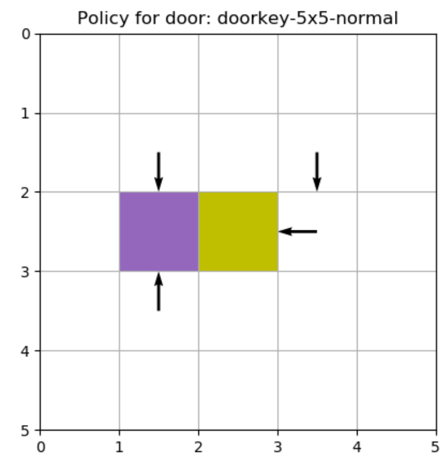


(d)

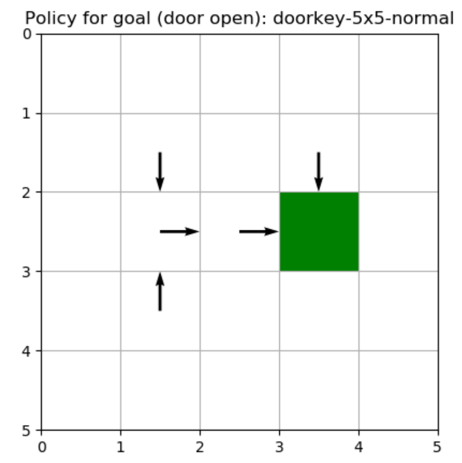
Fig. 1. Costs



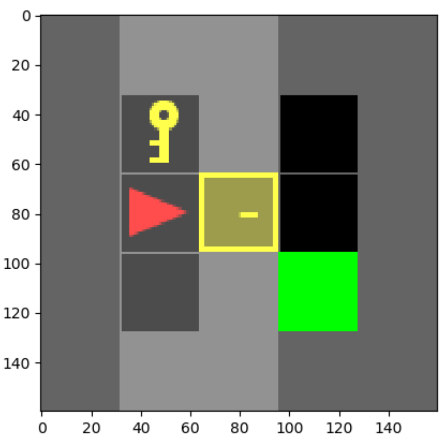
(a)



(b)



(c)



(d)

Fig. 2. Policy

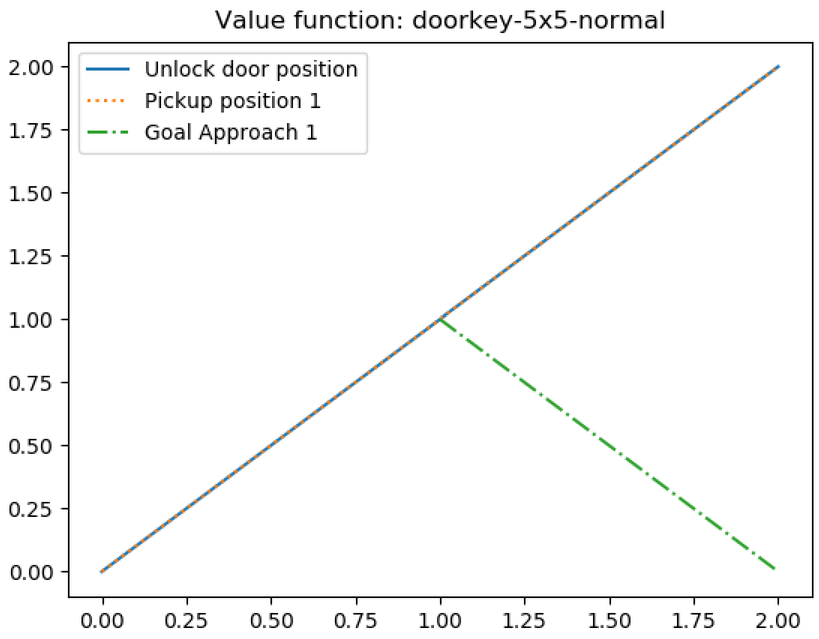
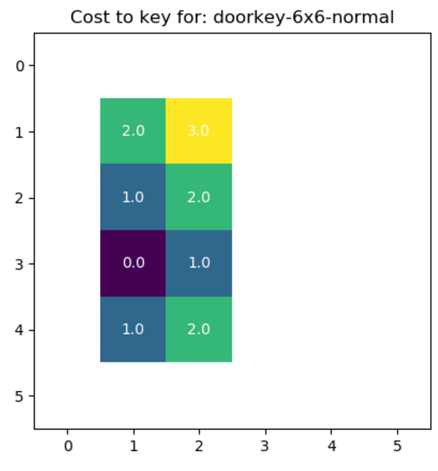
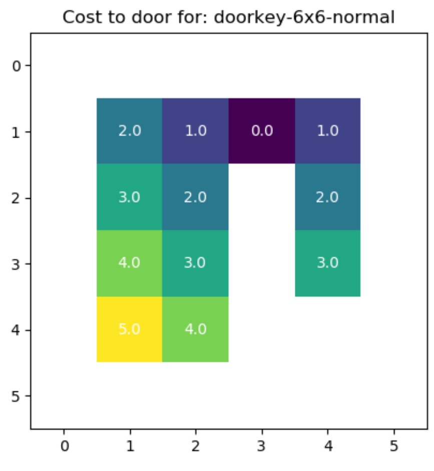


Fig. 3. Value function

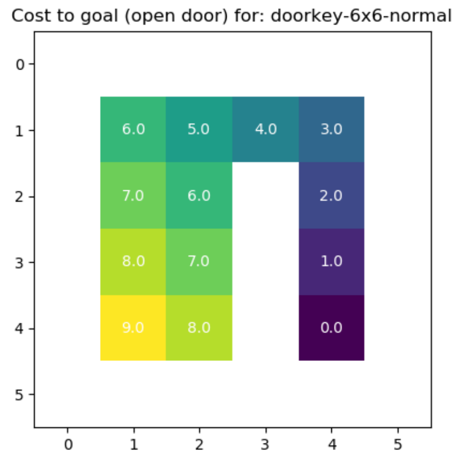
B. Doorkey-6x6-normal



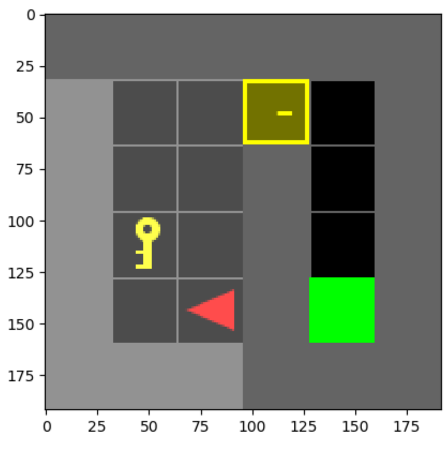
(a)



(b)

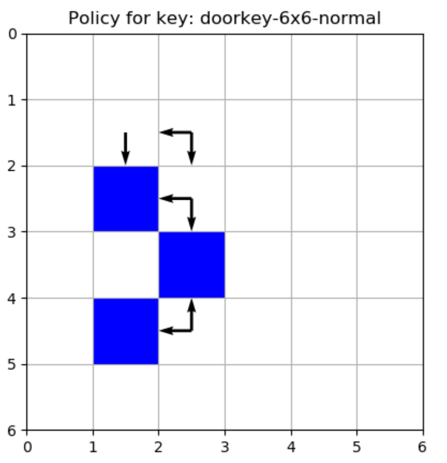


(c)

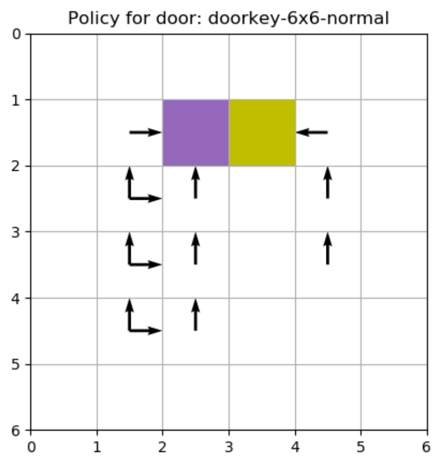


(d)

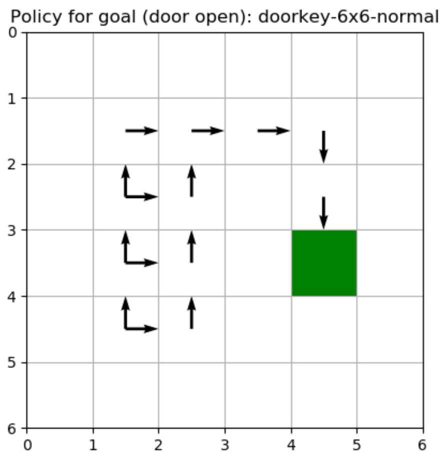
Fig. 4. Costs



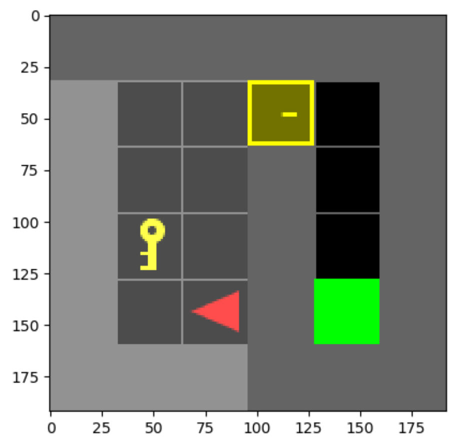
(a)



(b)



(c)



(d)

Fig. 5. Policy

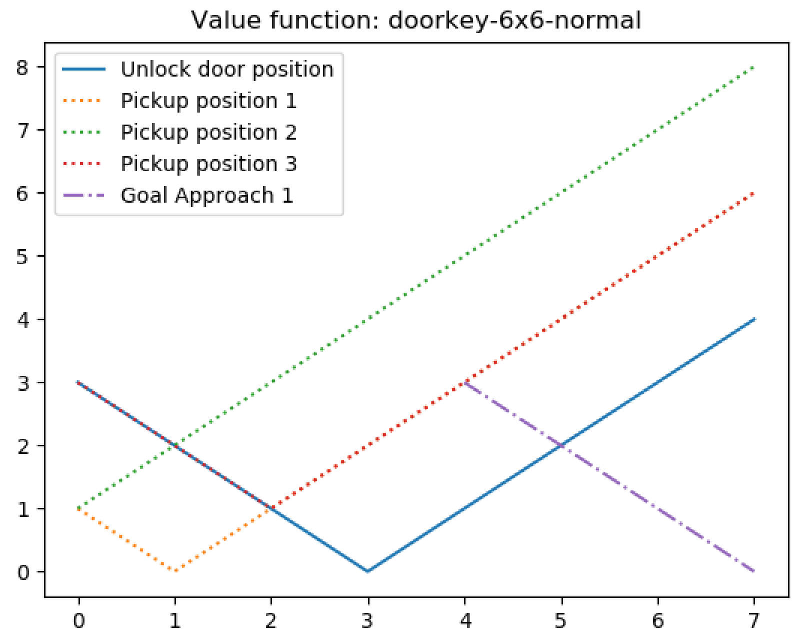
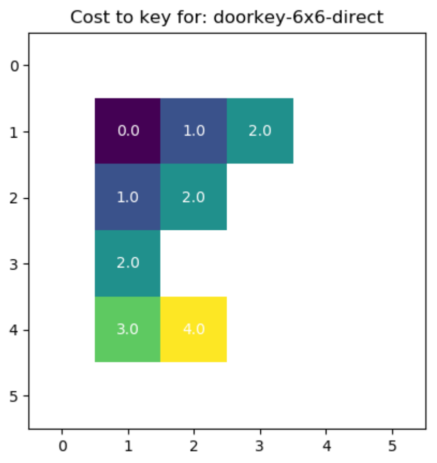
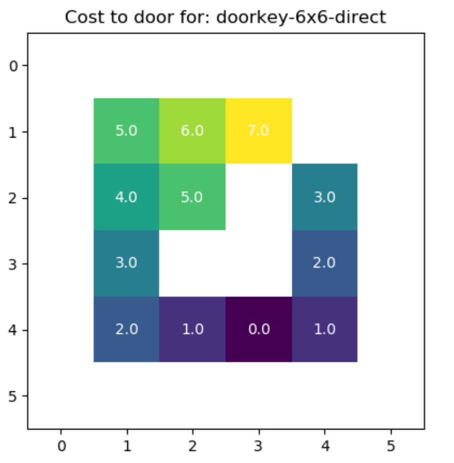


Fig. 6. Value function

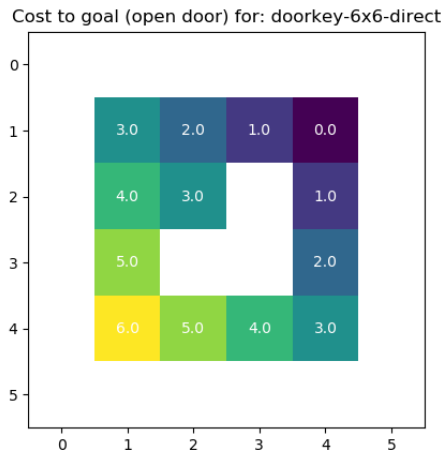
C. Doorkey-6x6-direct



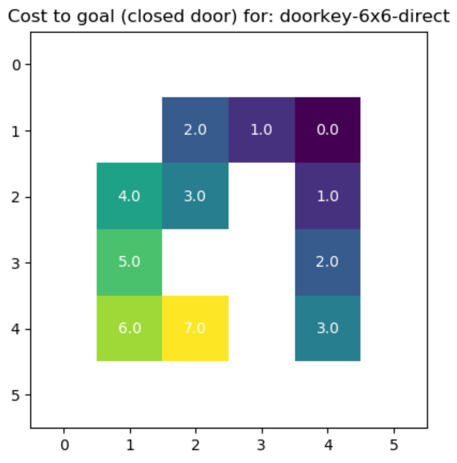
(a)



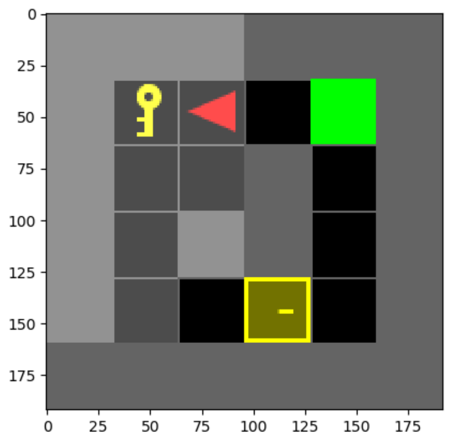
(b)



(c)

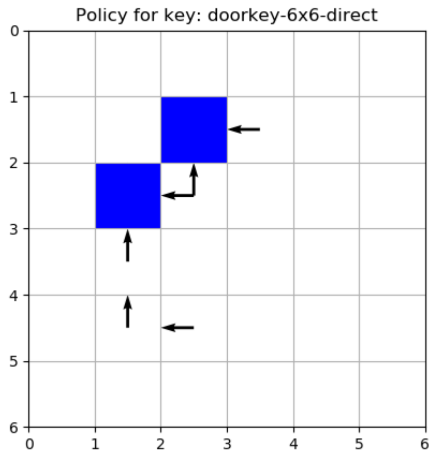


(d)

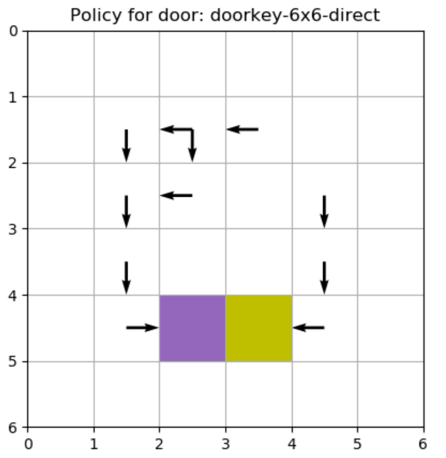


(e)

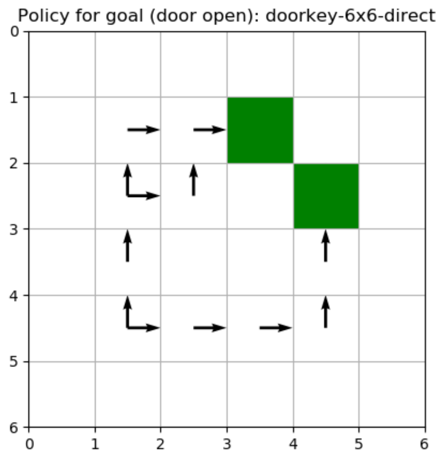
Fig. 7. Costs



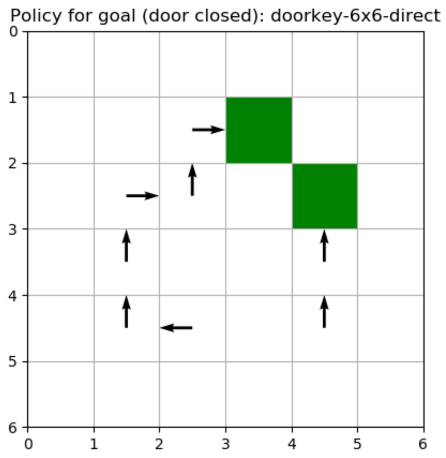
(a)



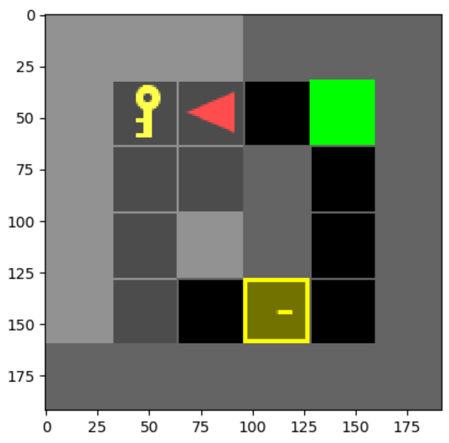
(b)



(c)



(d)



(e)

Fig. 8. Policy

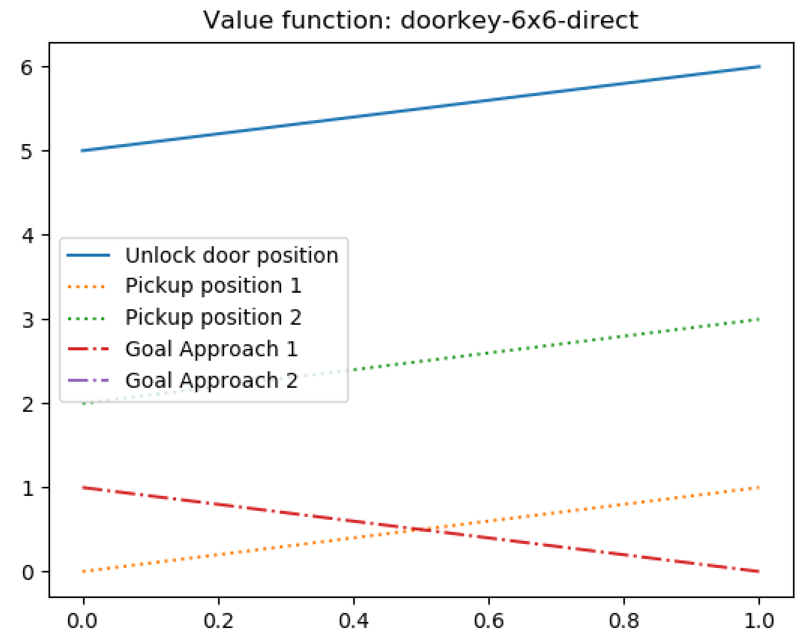
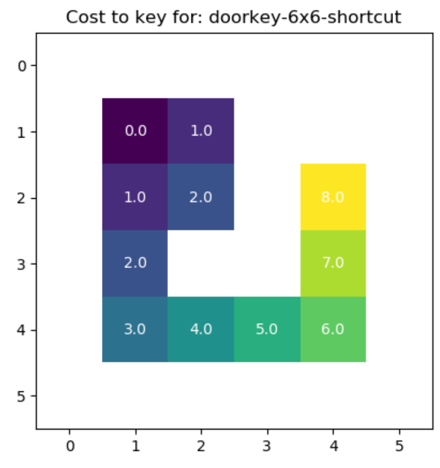
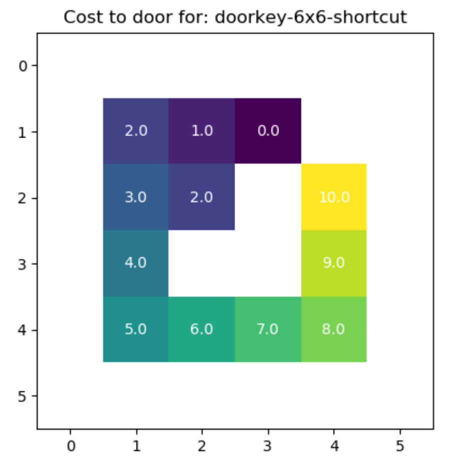


Fig. 9. Value function

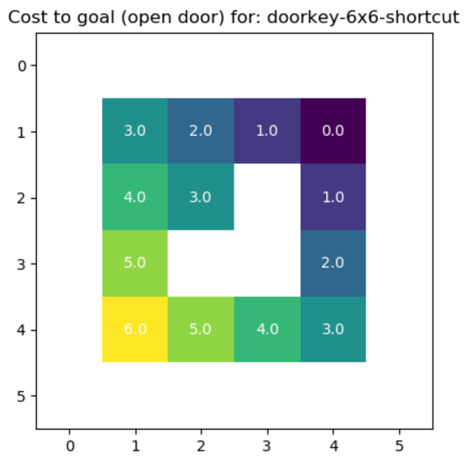
D. *Doorkey-6x6-shortcut*



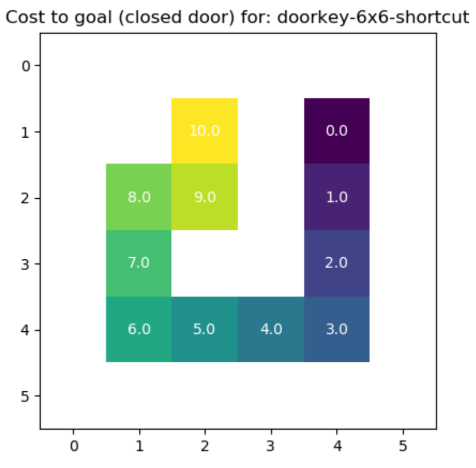
(a)



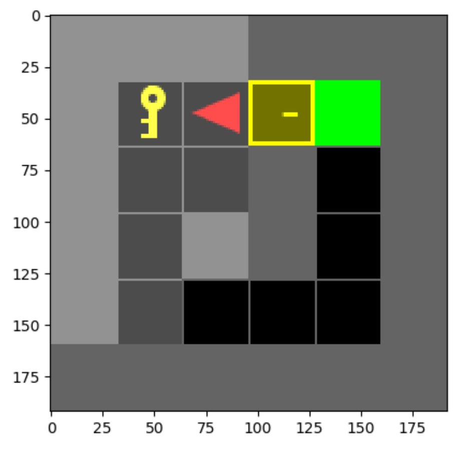
(b)



(c)

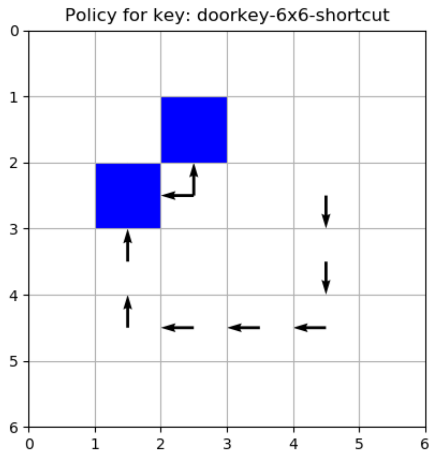


(d)

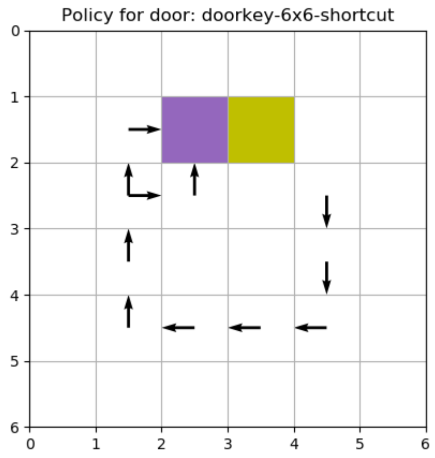


(e)

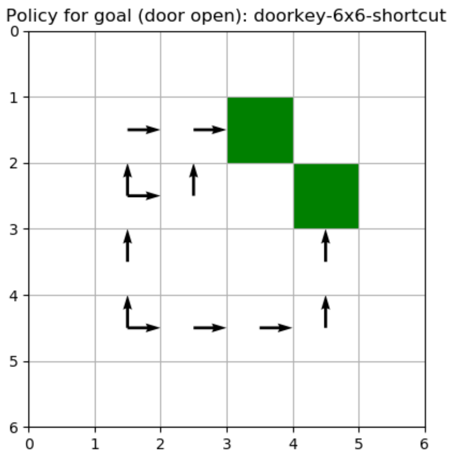
Fig. 10. Costs



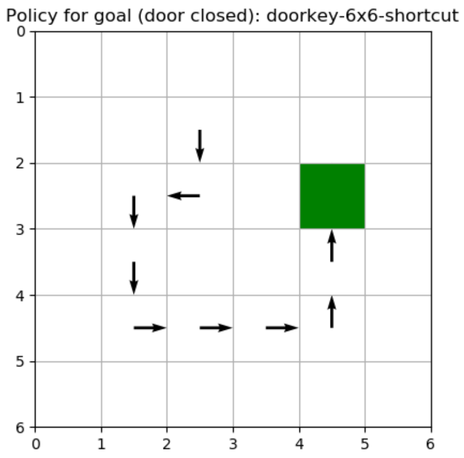
(a)



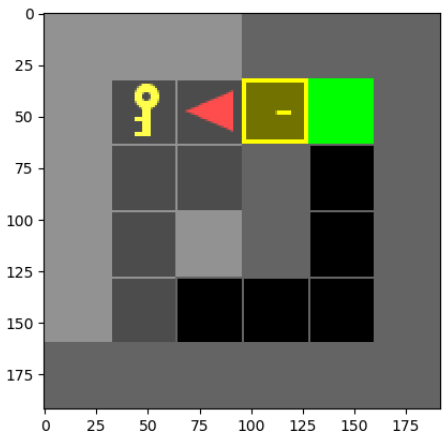
(b)



(c)



(d)



(e)

Fig. 11. Policy

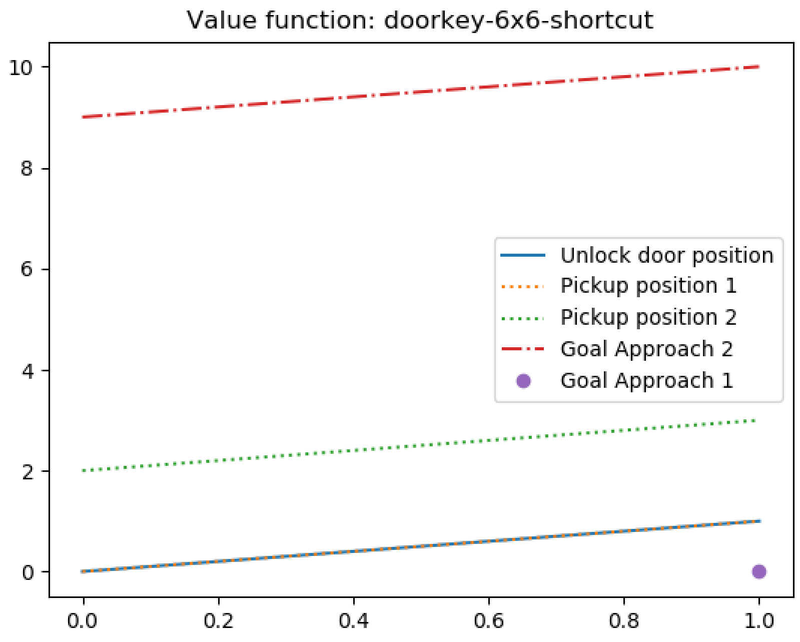
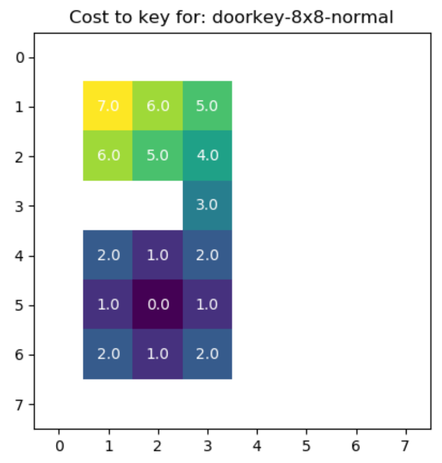
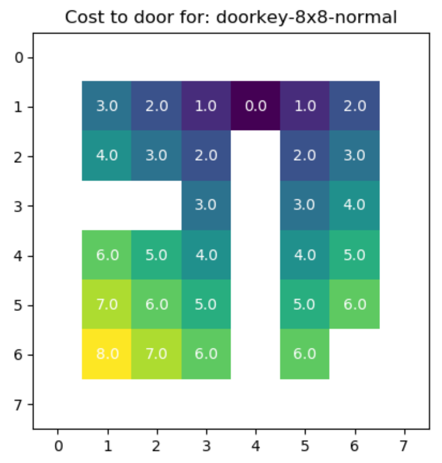


Fig. 12. Value function

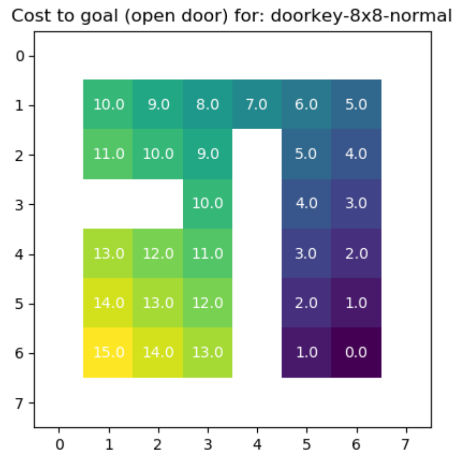
E. Doorkey-8x8-normal



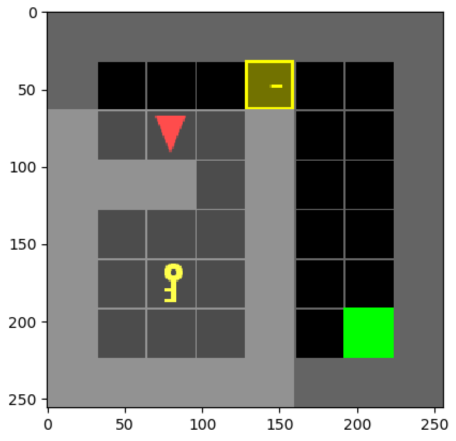
(a)



(b)

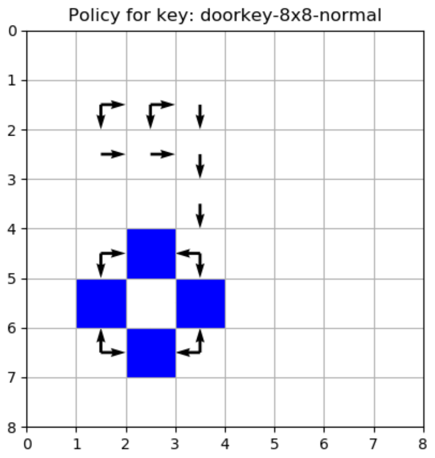


(c)

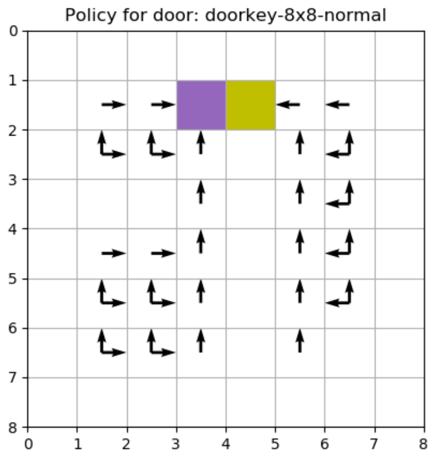


(d)

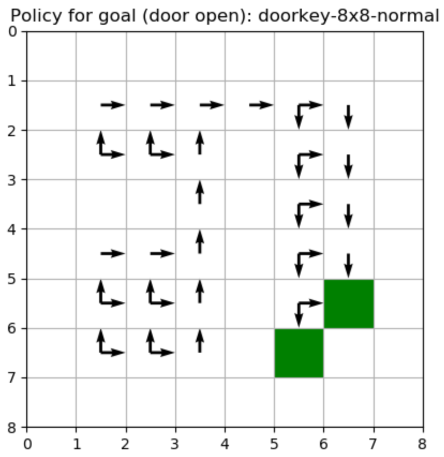
Fig. 13. Costs



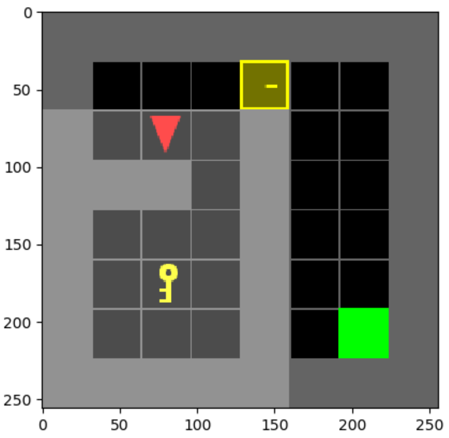
(a)



(b)



(c)



(d)

Fig. 14. Policy

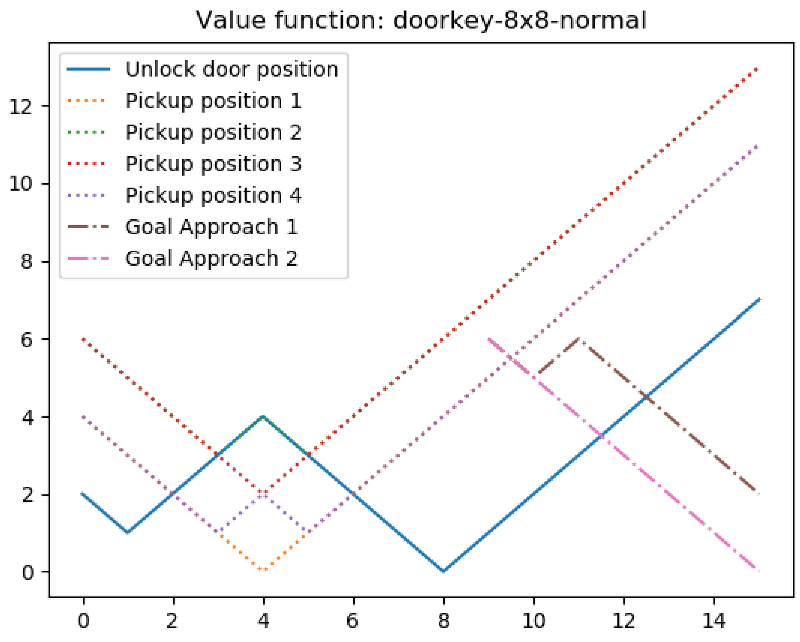
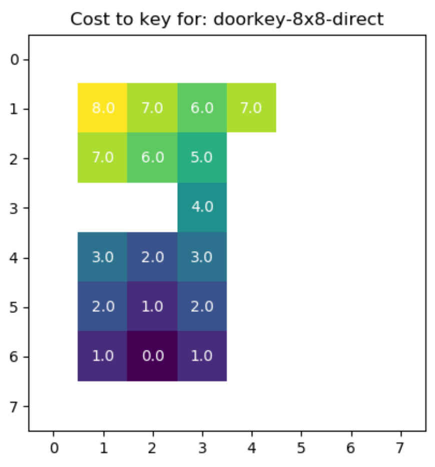
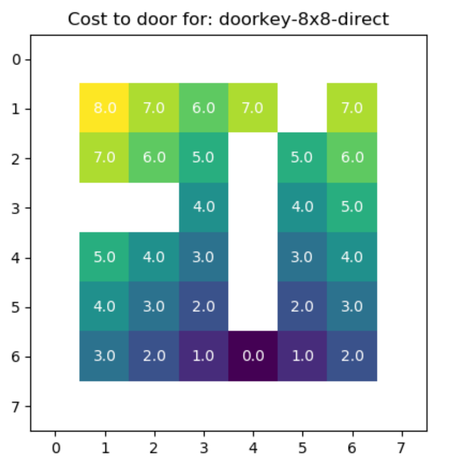


Fig. 15. Value function

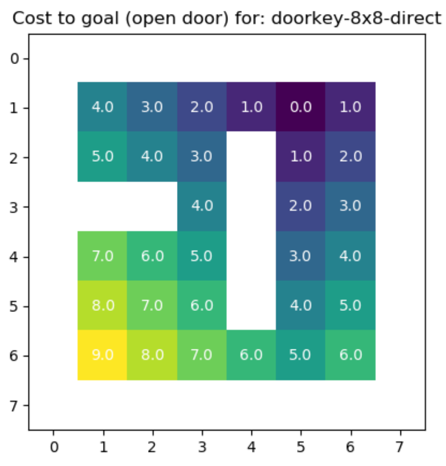
F. *Doorkey-8x8-direct*



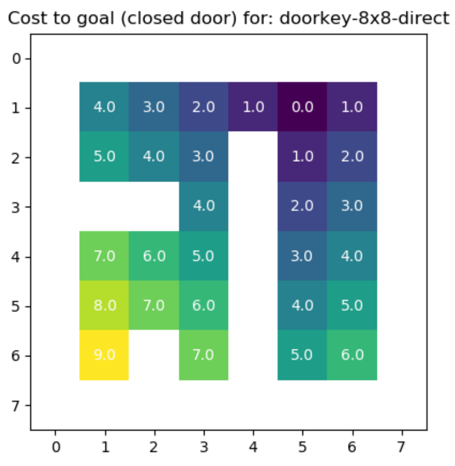
(a)



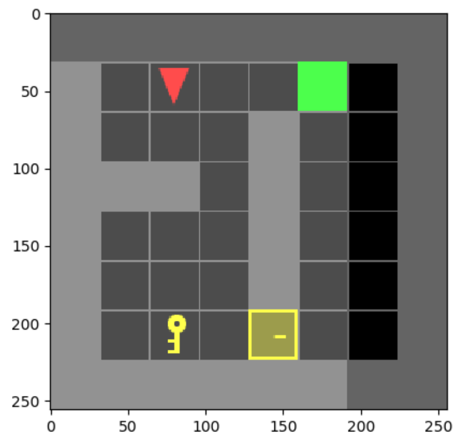
(b)



(c)

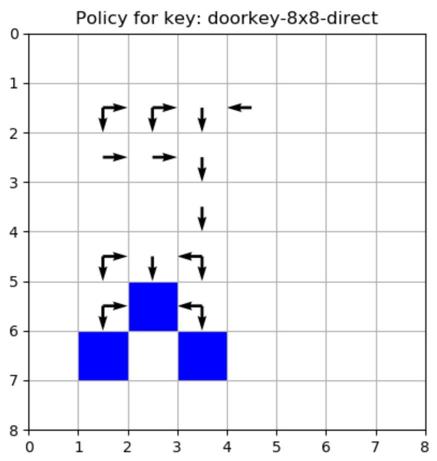


(d)

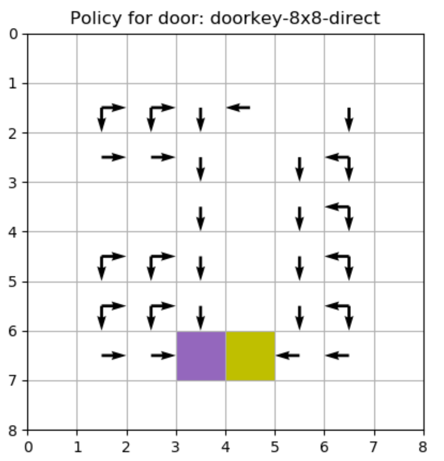


(e)

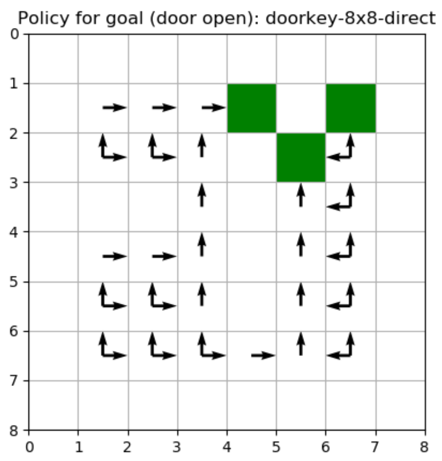
Fig. 16. Costs



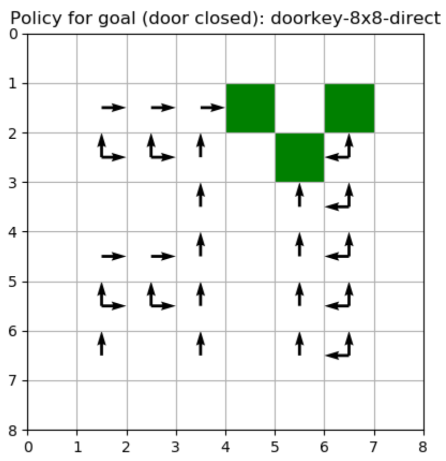
(a)



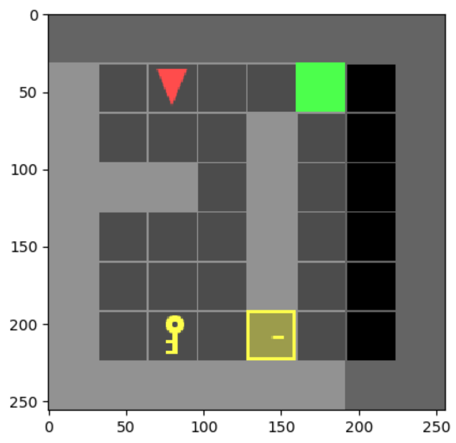
(b)



(c)



(d)



(e)

Fig. 17. Policy

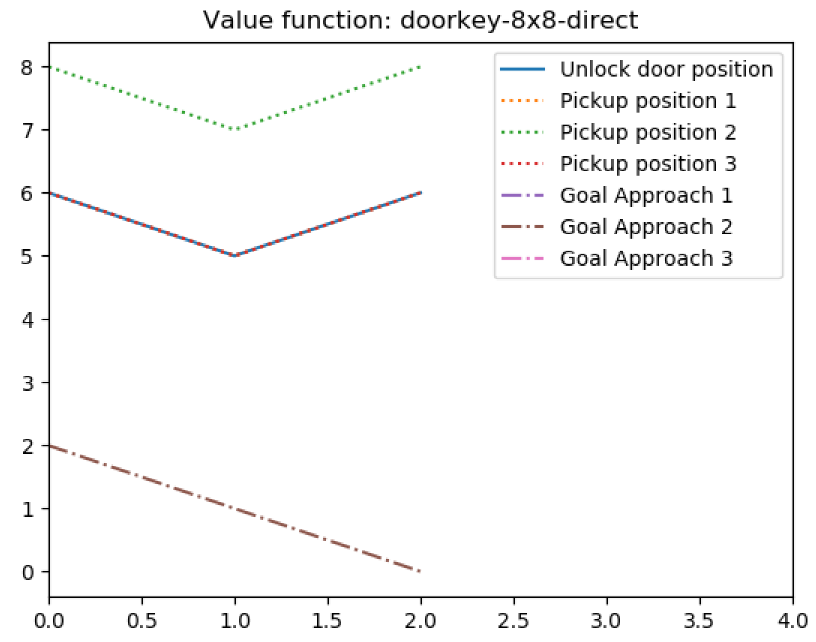
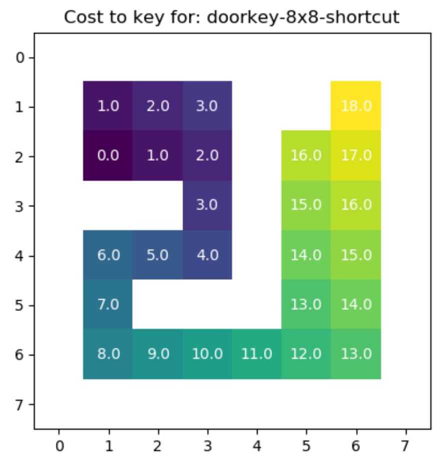
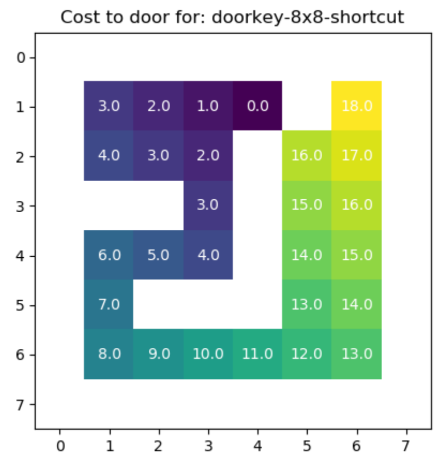


Fig. 18. Value function

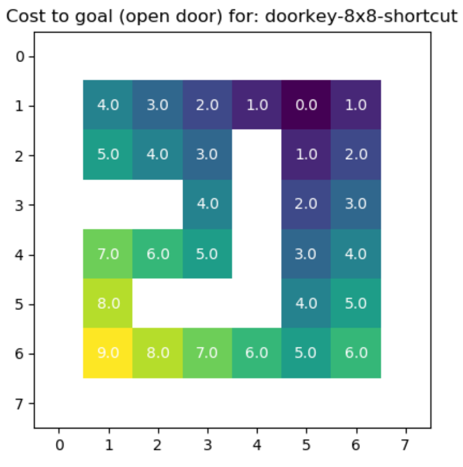
G. *Doorkey-8x8-shortcut*



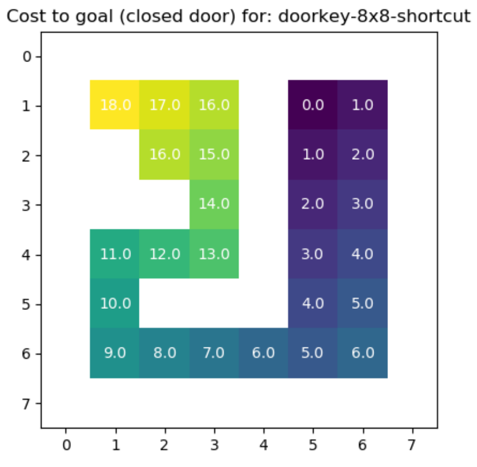
(a)



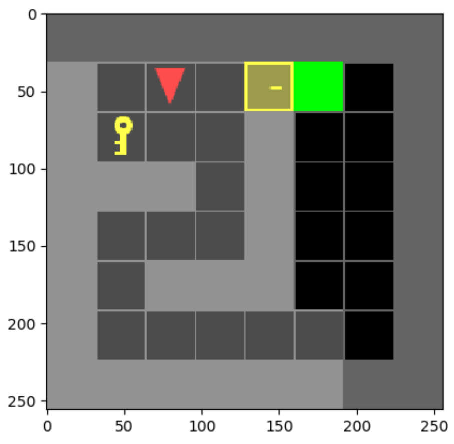
(b)



(c)

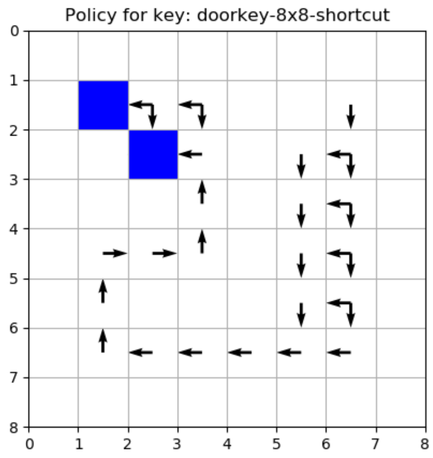


(d)

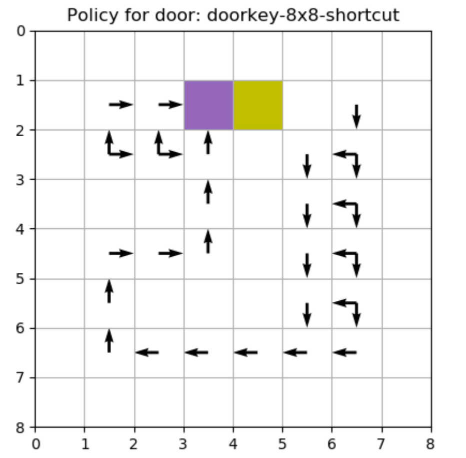


(e)

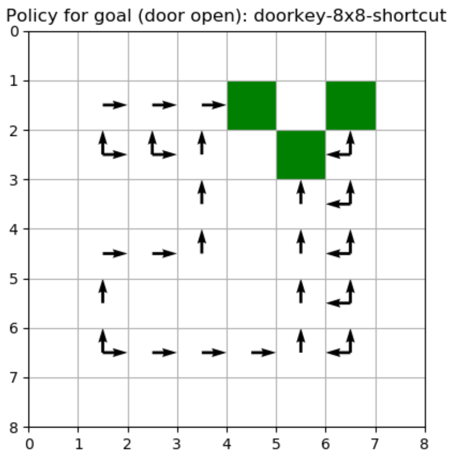
Fig. 19. Costs



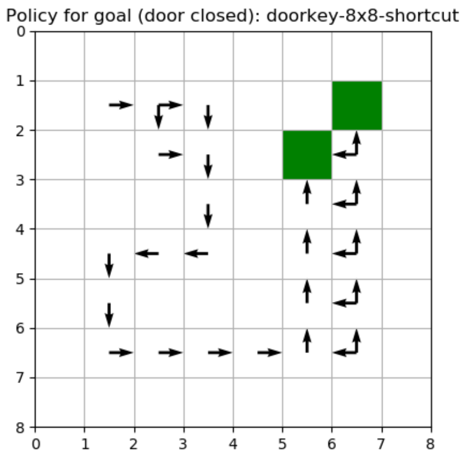
(a)



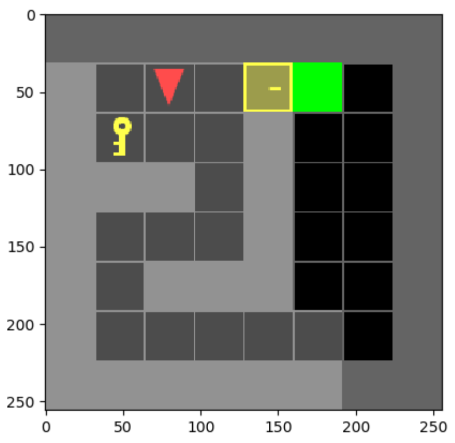
(b)



(c)



(d)



(e)

Fig. 20. Policy

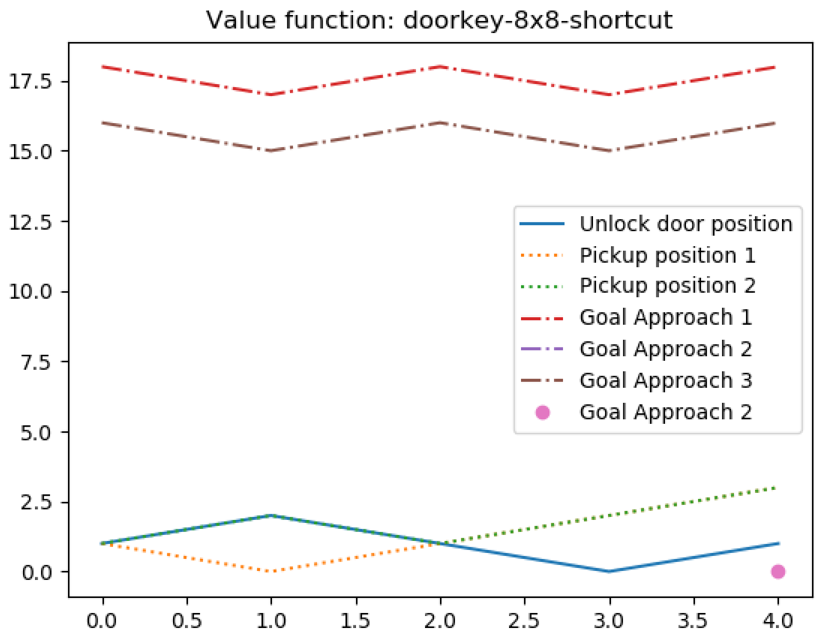


Fig. 21. Value function

Optimal sequence for doorkey-5x5-normal: [TL PK TR UD MF MF TR MF]

Optimal sequence for doorkey-6x6-normal: [TR MF TR PK TL MF UD MF MF TR MF MF MF MF MF]

Optimal sequence for doorkey-6x6-direct: [TL TL MF MF]

Optimal sequence for doorkey-6x6-shortcut: [PK TL TL UD MF MF]

Optimal sequence for doorkey-8x8-normal: [TL MF TR MF MF MF TR PK TR MF MF MF MF TR UD MF MF MF TR MF MF MF MF MF]

Optimal sequence for doorkey-8x8-direct: [TL MF MF MF]

Optimal sequence for doorkey-8x8-shortcut: [MF TR PK TR MF TR MF UD MF MF]

It can be seen from the figures of all the test environments that the algorithm indeed finds the most optimal path always. Even in cases where direct path is available but using key and opening door is less expensive, the robot is able to identify it. The algorithm is expected to work in almost any environment where the conditions remain same i.e. key can be picked up only when facing the key, door can be opened only when facing the door.

V. Conclusion and Future Work

The algorithm works as expected.

However, this algorithm can be modified to include the turning cost. This will reduce calculation of orientation at each step, which is being currently done. The optimality will remain same i.e. same path will be computed, only difference will be policies will be dependent on the orientation of the robot.

Acknowledgment

I thank Mr. Shiladitya Biswas, Mr. Ayon Biswas for collaborating with me on this project. We held many whiteboard discussion sessions.