ECE 276B Project 1
Markov Process and Dynamic Programming

Shiladitya Biswas
Dept. of Electrical and Computer Engineering
Universrity of California,San Diego
California, USA

I. Introduction

Path planning plays a major role in any robotic setup. For example in a house hold setup the robot should be able to navigate from one location to another to perform several household duties. Hence, for a given environment (i.e. creating a MAP of the environment by using techniques discussed in ECE276A) the robot's software should be able to make proper decisions and generate appropriate control policies to safely navigate from one point to the other. One method to achieve this is to breakdown the robot's position and orientation into discreet states. Each of these states has a reward attached to it. In general the goal/final pose to be reached has the highest reward. The control actions that the robot undertakes at a given state to reach another state is called the control policy. Thus we can construct a connected graph (generally called the **Markov Decision Process MDP graph**), indicating all the inter-state transitions (and their respective costs/reward) possible for a given environment and use techniques like value iteration, label correction, etc. to find the path that has the least cost or the highest reward. In this project, we are given a minigrid environment with an agent in it. The agent can perform certain actions like Move Forward(MF), Turn Left(TL), Turn Right(TR), Pick Key(PK) and Unlock Door(UD). The main goal is to reach a the goal in the shortest path possible. Sometimes it is observed that reaching the goal via the Door costs less as compared to any other path. On the other hand, there might be certain cases where it is impossible to reach the goal without using the door. Similarly, there are cases where the goal can be reached without using the door. Hence we have to take into consideration all such cases and choose the path that gives the highest reward/ lowest cost.

II. Problem Formulation

As discussed in the previous section, we are trying to solve a path planning problem using **Markov Decision Process(MDP)** formulation and Dynamic programming (here I used Label Correction Algorithm). In this section we look into the problem formulation in further depth.

A. Markov Decision Process

As per definition a Markov Decision process is a discreet time stochastic control process which provides a mathematical framework for modelling decision making in circumstances where outcomes are partly random and partly under control of decision maker. In our case here, the problem is completely deterministic in nature i.e. the agent completely obeys the input signals like MOVE FORWARD, TURN LEFT etc with 100% guarantee.In this project a state will be defined as follows,

X= $\begin{bmatrix} Agent\_Location \\ Agent\_orientation \\ Door\_Status \\ Key\_status. \end{bmatrix}$

And the actions are as follows:

U= $\begin{bmatrix} MF & TL & TR & PK & UD. \end{bmatrix}$

Now we have to decide upon the cost of transition from one state to another. But we face a problem in this formulation i.e. we have too many states to keep track of. No of states is roughly equal to $4 \times 2 \times 2 \times N$. Where N=No. of Empty Cells, door open or close, is carrying key or not carrying key. Added to that the value N is state dependent, since once the Key is picked up or once a door is unlocked the value of N increases. In order to reduce the states I have made the following assumptions

1) Cost for pickup key and unlocking door action's cost/rewards equal to zero.
2) Cost is incurred only on the Move Forward Command and it is equal to One.
3) The stage cost/reward for each state(here Cell location) is initialized to zero.
4) The motion control inputs are combined as follows:
   a) Move Right: Turn Right + Move Forward
   b) Move Left: Turn Left + Move Forward
   c) Move back: Turn Right + Turn Right + Move Forward
   d) Pick key and Unlock Door command remains the same.

The new state space is as follows:

X_new= $\begin{bmatrix} Agent\_Location \end{bmatrix}$

And the actions are as follows:

U_new= $\begin{bmatrix} MoveRight & MoveLeft & MoveBack & PK & UD. \end{bmatrix}$

Since the problem at hand is deterministic and we have simplified our state space to a large extent, a simple Deterministic Shortest path algorithm can be employed to find the shortest path from one cell to every cell in the environment map. A general method to do so is the Label Correcting Algorithm.

I would like to thank Saurabh Mirani and Aditya Mishra for the helpful discussions on the project.

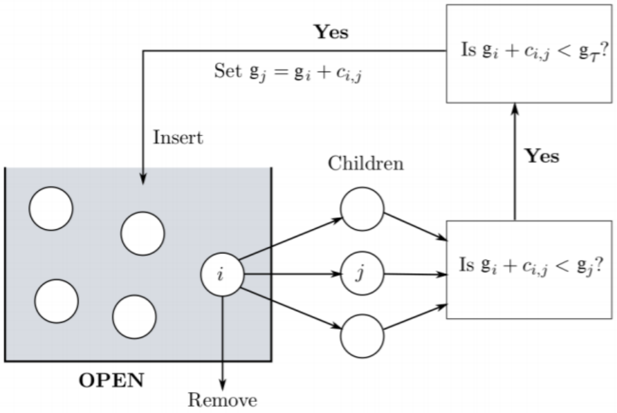
Yes
Set $g_j = g_i + c_{i,j}$
Is $g_i + c_{i,j} < g_\tau$?
Insert
Children
Yes
i
j
Is $g_i + c_{i,j} < g_j$?
OPEN
Remove

Fig. 1. Label Correction Algorithm Flowchart

B. Label Correcting Algorithm

The traditional label correcting algorithm to find the shortest path goes as follows. Consider a graph with vertex/state set V, weighted edge cost set C and starting node $s \ \epsilon \ V$ and an artificial terminal node $\tau \epsilon V$. The optimal path will not have more than $|V|$ elements. We have No of stages $T = |V| - 1$

We start with a node of a graph ”i” and put them in a set called the OPEN set and consider its children nodes ”j”. Next we calculate the cost of going from node i to j and from node i to τ. if:

$$g_i + c_{i,j} < g_j \tag{1}$$

then we check If:

$$g_i + c_{i,j} < g_\tau \tag{2}$$

If both equations 1 and 2 are TRUE then we update the cost of the node ”j” as $g_j = g_i + c_{i,j}$. Once this is done we remove the node ”i” from the OPEN set.
Here, g_k and $c_{i,j}$ is the cost of the total cost to reach node ”k” and the cost to go from node ”i” to ”j” respectively. This algorithm is carried out for all the node in V i.e. for all $i \epsilon V$. The total algorithm can be summarized in figure 1. At the end the OPEN set becomes completely empty indicating that we have visited all the node and the final g values of all the nodes is the shortest distance from node we started from i.e. the first node for which. As mentioned earlier, this project is done using a special case of Label Correction algorithm(LCA) i.e. Breadth first search (BFS). LCA becomes a BFS algorithm when we implement the OPEN set as a queue which uses the First-In-First-Out (FIFO) methodology.

C. Complete formulation

The complete formulation of our project is as follows.
State space: $X\_new = V$ and Control Space: $U\_new = V$
Motion model: $X^{t+1}_{new} = f(X^t_{new}, U^t_{new})$
Cost:

$$\ell(X^t_{new}, U^t_{new}) = 0, \text{ if } X_t = \tau \text{ and 1 otherwise} \tag{3}$$

$$q(X^t_{new}, U^t_{new}) = 0, \text{ if } X_t = \tau \text{ and } \infty \text{ otherwise} \tag{4}$$

Here we have T= N-1. Where. N is the number of empty cells in the environment. Since the cost is 1 for every transition, for a give parent cell on the grid (we begin with the goal cell as the very first parent cell) we find the respective children cells and update their cost value with $1+$Parent cell value (here goal cell value=0). We then iterate for for T times or until all the cells are reached/visited. As a result we get a 2D matrix each of whose cell value represents the least cost it will take to reach the goal from it. We call this grid as the Cost Grid.

III. Technical Approach

A. Decide if key is needed or not

The approach to find the shortest path is broken down into 4 steps. Using the method discussed in II-C we find out the following costs:

1) Cost to reach the goal from Agent's starting position. (Call it C_{direct})
2) Cost to reach the key from the Agents Starting position.(Call it C_{Key})
3) Cost to reach the door from the key position (Call it $C_{key-Door}$)
4) Cost to reach the Goal from the Door Position. (Call it $C_{Door-Goal}$)

We check the following condition.

$$C_{direct} > C_{key} + C_{keytoDoor} + C_{DoortoGoal} \tag{5}$$

if equation 5 is true then we need the key to reach the goal in the shortest path possible. Else if equation 5 is False then we don't need the key to reach the goal in the shortest path. There exists a direct shorter path from the initial Agent position to the goal position.

B. Traverse the shortest path

1) When equation 5 is True, we make the key position as the goal and find the respective cost grid. Hence now we get the total cost to go from the initial agent position to the key position. We then use the agent orientation and the position to move the agent to the key position. For a given agent position we look at its 4 surrounding cells and find the cell with the lowest cost. We then use the agent orientation to transition (generate the appropriate sequence and store them in a list S1) to the cell with the lowest cost amongst the 4 possible surrounding cells. We continue doing this until we reach a cell adjacent to the key position cell. then we orient ourselves and (using the relevant motion command and store them in S1) Pickup the key.

2) Once the key is picked, we again recompute the cost grid, but this time we change the goal position to the Door Position and the initial agent position to the current agent position. Then we use the same strategy as discussed in last paragraph to generate (and store them in a list S2) the optimum sequence

to travel from the key position to the door position. We then orient our agent so that it faces the door cell and unlock the door. The corresponding action sequences are appended into S2.

3) Once the door is unlocked, we again recompute the cost grid, but this time we use the final goal position as Goal and set the initial agent position to the current agent position (this position will be adjacent to the door position). We then use the same strategy as discussed in the last paragraph to generate (and store them in a list S3) the optimum action sequence to travel from the current agent position to the Final goal position.

Finally we club S1,S2 and S3 to get the final action sequence i.e. Seq=[S1,S2,S3]. This Seq vector and the environment variable is given as input to the gif making function. The final gif is stored in a an appropriate folder.

C. Finding the value Functions

Now, that we got the optimum control sequence ”Seq” from the above section, we apply these control actions on the agent in the environment and calculate the value functions of the certain cells (lets call their set W), namely the 4 cells surrounding the Key Position, 4 cells surrounding the goal position and 1 cell from where we unlock the door. In order to do so we keep computing the Cost grid with the goal position = the current agent position. The pseudo code of the Algorithm is shown in Algorithm 1. The output Q matrix is plotted for all complete sequence

Algorithm 1 Finding the value functions of certain cells

Require: Seq,env

```
function LOOP(Action in Seq)
for $Action \leftarrow in\ Seq$ do
    $step(env, Action)\ Apply\_Control\_action$
    $Agent\_Pos \leftarrow env.agent\_pos\ GetCurragentPos$
    $Cost\_Grid \leftarrow Get\_Cost\_Grid(Agent\_pos)$
    $Q \leftarrow$ get_values_of_Cells_in_W(Cost_Grid)
        return $Q$
        =0
```

IV. Results and Discussions

The results of the above algorithm are shown below. The agent reached the final goal position using the shortest path possible.

1) Environment name: doorkey-5x5-normal: The value and Policy function graph is shown in fig 2. The control sequence output: $fTL- > PK- > TR- > UD- > MF- > MF- > TR- > MF$

2) Environment name: doorkey-6x6-direct: The value and Policy function graph is shown in fig 3. The control sequence output: $TR- > TR- > MF- > MF$

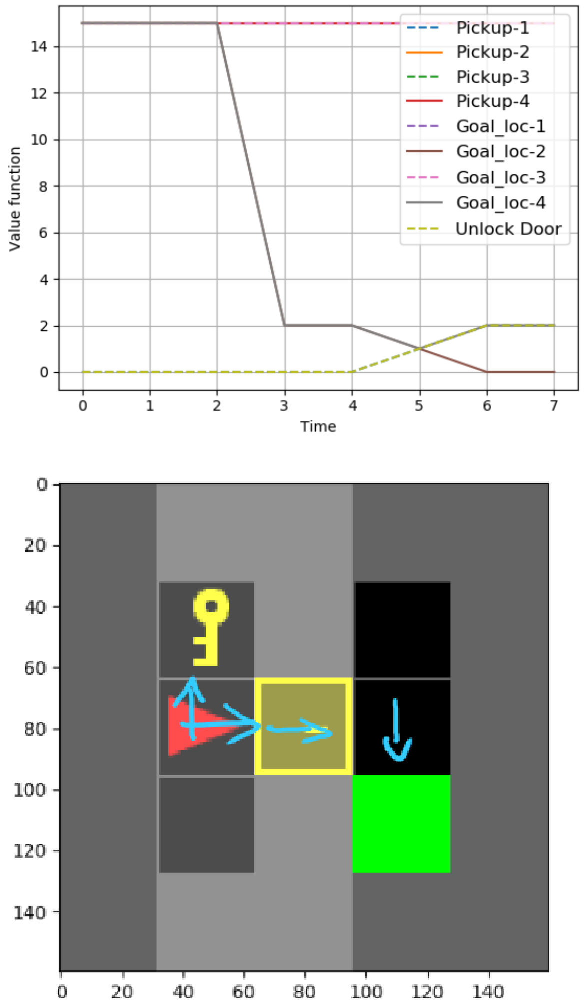

Fig. 2. Value Function and Policy Function

3) Environment name: doorkey-6x6-normal: The value and Policy function graph is shown in fig 4. The control sequence output: $MF- > TR- > PK- > TR- > MF- > TL- > MF- > MF- > MF- > TR- > UD- > MF- > MF- > TR- > MF- > MF- > MF$

4) Environment name: doorkey-6x6-shortcut: The value and Policy function graph is shown in fig 5. The control sequence output: $PK- > TR- > TR- > UD- > MF- > MF$

5) Environment name: doorkey-8x8-direct: The value and Policy function graph is shown in fig 6. The control sequence output: $TL- > MF- > MF- > MF$

6) Environment name: doorkey-8x8-normal: The value and Policy function graph is shown in fig 7. The control sequence output: $TL- > MF- > TR- > MF- > MF- > TR- > MF- > TL- > PK- > TL- > MF- > TL- > MF- > MF- > MF- > TR- > UD- > MF- > MF- > MF- > TR- > MF- > MF- > MF- > MF- > MF$

7) Environment name: doorkey-8x8-shortcut: The value and Policy function graph is shown in fig 8. The control sequence output: $TR- > MF- > TL- > PK- > TL- > MF- >$

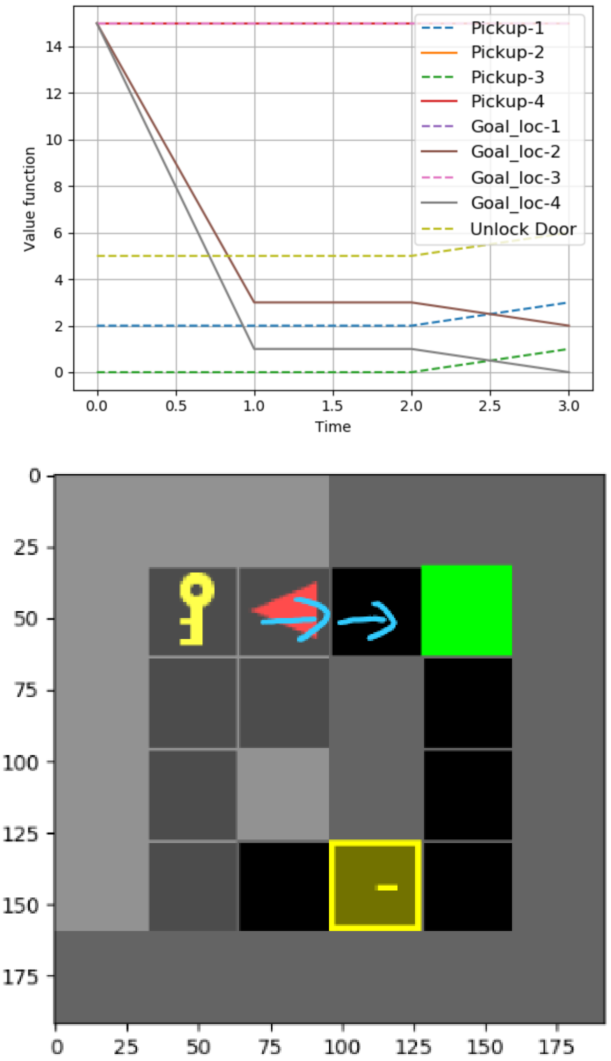

Fig. 3. Value Function and Policy Function

$MF- > UD- > MF- > MF$

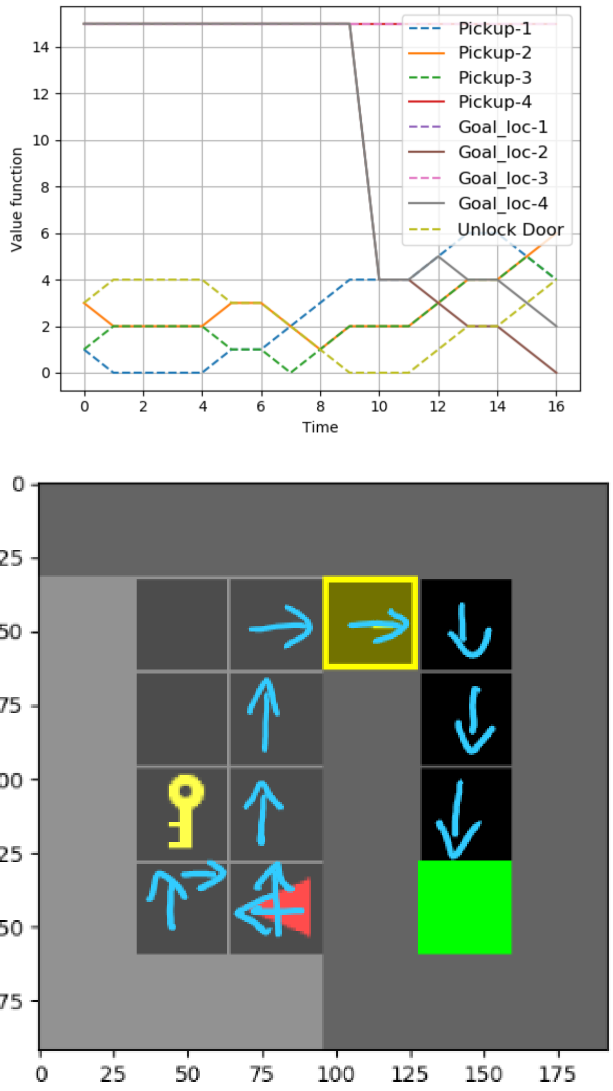

Fig. 4. Value Function and Policy Function

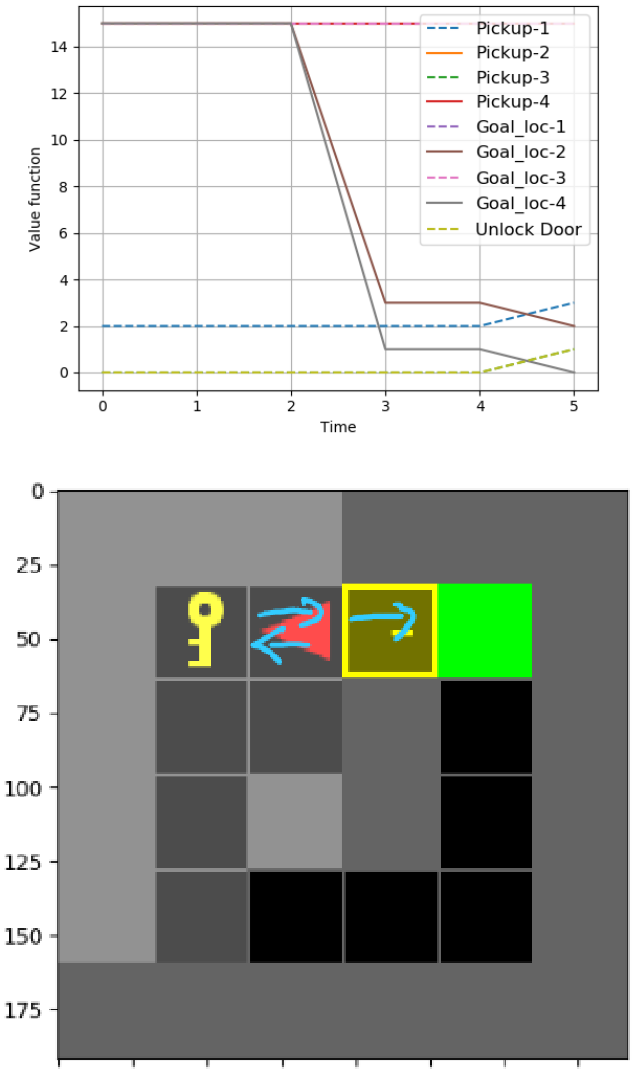

Fig. 5. Value Function and Policy Function

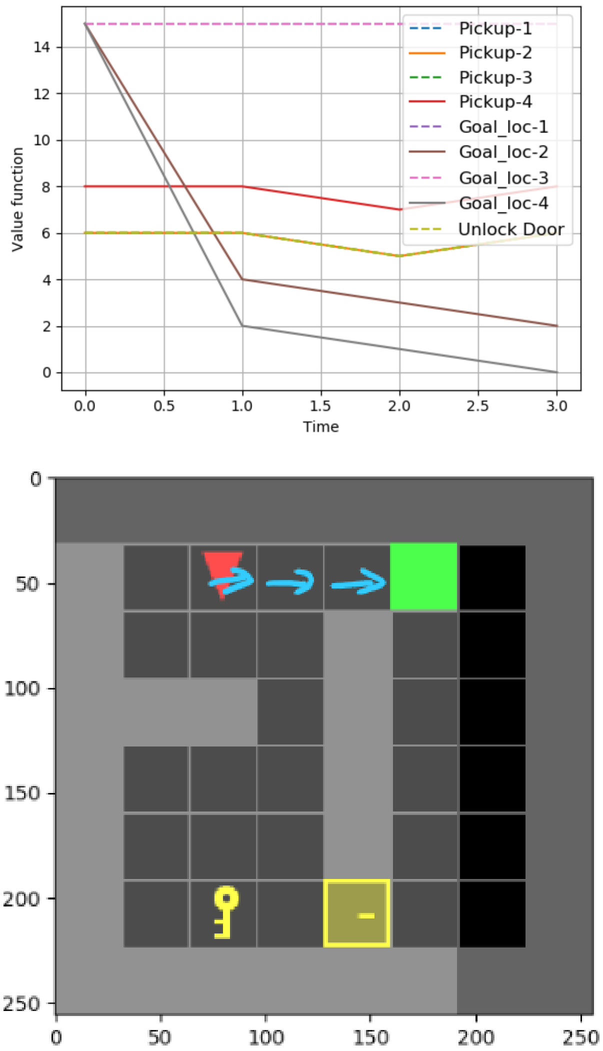

Fig. 6. Value Function and Policy Function

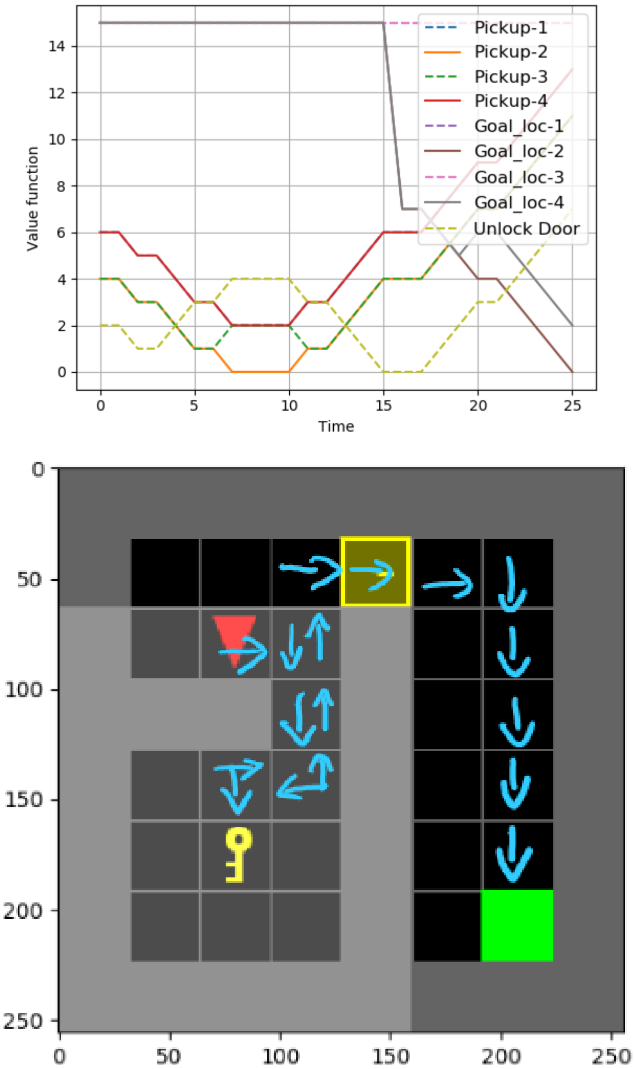

Fig. 7. Value Function and Policy Function

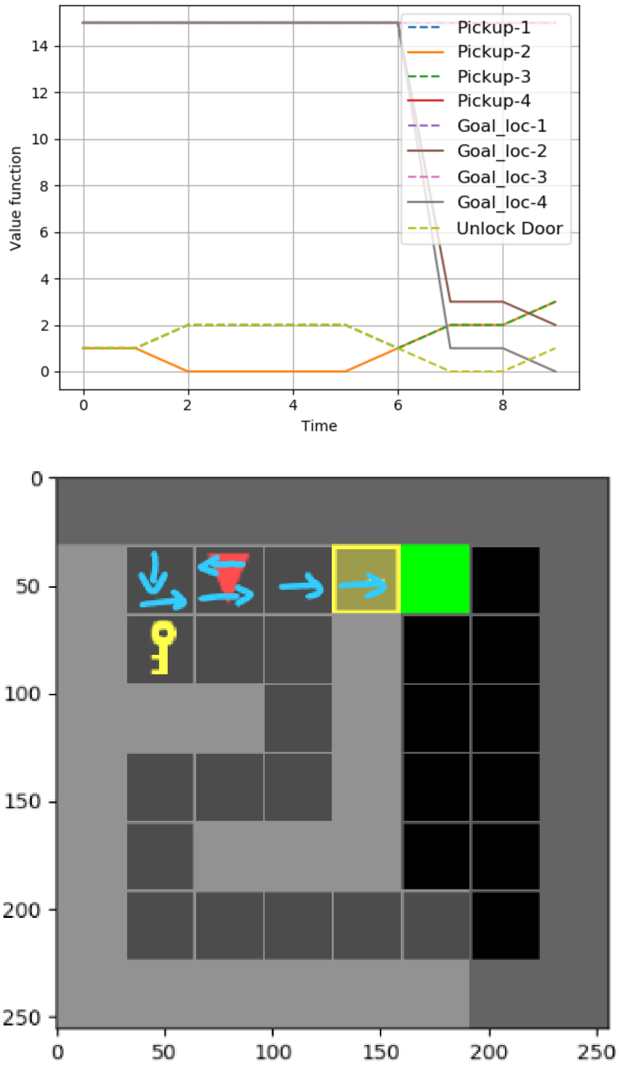

Fig. 8. Value Function and Policy Function