# Artificial Intelligence

CSI 4130/5130 Pourya Shahverdi *Instructor: Prof. Qu*

Oakland University

Homework 1

## § MDP §

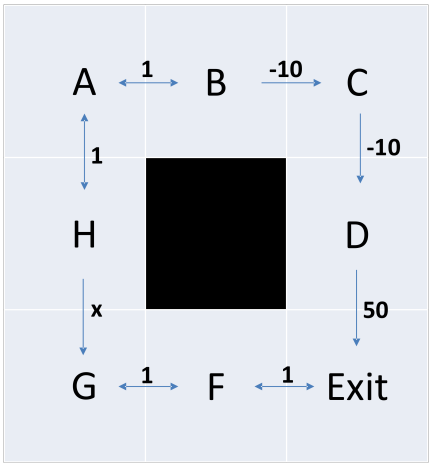


**Figure 1:** A 3x3 grid world with the center cell not accessible (colored in shade), there are states A through H, and Exit.

Given a MDP representing a 3x3 grid world with the center cell not accessible (colored in shade), there are states A through H, and Exit. The actions at each state include moving to its neighbors as indicated by the arrow and exit. The number along with the arrow is the immediate reward the agent would receive [Pay attention, the reward from H to G is x]. For exit action, there is no reward.

**Codes are publicly available at** GitHub: MDP-Problem-AI-CSI-5130

## Problem 1: Value Iteration with $\gamma$=1

Assume all the actions are deterministic. If the discount factor $\gamma$=1, please calculate the V*(A), V*(C), V*(G), V*(Exit).

**Answer:**

I've written a code (Appendix I) for calculation of the value of each state and finding the optimal policy. When $\gamma$=1, the value iteration does not converge and the code output would be like this for 3 iterations:

Code output for Problem 1

```
After 3 iterations:
The final value for each state is:
 {'A': 3, 'B': 30, 'C': 41, 'D': 52, 'E': 3, 'F': 3, 'G': 3, 'H': 3, 'O': 0}
And, the optimal policy is:
 {'A': 'r', 'B': 'r', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u',
  ↪ 'O': 'r'}
```

## Problem 2: Value Iteration with $\gamma$=0.5

Assume all the actions are deterministic. If the discount factor $\gamma$=0.5, please calculate the V*(A), V*(C), V*(G), V*(Exit).

**Answer:**
For $\gamma$=0.5, the value iteration will be converged in 55 iterations and the code output would be like this:

Code output for Problem 2

```
After 55 iterations:
The final value for each state is:
 {'A': 2.0, 'B': 2.0, 'C': 15.5, 'D': 51.0, 'E': 2.0, 'F': 2.0, 'G': 2.0, 'H': 2.0,
  ↪ 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u',
  ↪ 'O': 'r'}
```

## Problem 3: Q*(A, down) and Q*(A, right) with $\gamma$=1

Assume all the non-exit actions with a success probability of p that it transitions to next state and acquires the reward along the edge, otherwise, the agent would state in the same state with a reward of 0. If the discount factor $\gamma$=1, please calculate the Q*(A, down) and Q*(A, right), where action 'down' refers that the agent move from A to H, while action 'right' refers that the agent moves from A to B. **Answer:**
For answering this question, I have Q* for all State-Actions as well as A and it's possible actions (e.g., right and down). I see that value iteration with $\gamma$=1 and p = 0.5 will be converged in 55 iterations. If we increase the probability to 0.99, it takes 3235 iterations to be converged, and if we set p = 1, it won't be converged, which is interesting.

The output of my code for this problem is presented below:

Code output for Problem 3

```
Q*( A ) for actions right, left, up, down, and exit are = dict_values([1.0, 0.0,
 ↪ 0.0, 1.0, 0.0])
After 55 iterations:
The final value for each state is:
 {'A': 1.0, 'B': 1.0, 'C': 7.75, 'D': 25.5, 'E': 1.0, 'F': 1.0, 'G': 1.0, 'H': 1.0,
  ↪ 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u',
  ↪ 'O': 'r'}

Q*( B ) for actions right, left, up, down, and exit are = dict_values([-1.125, 1.0,
 ↪ 0.0, 0.0, 0.0])
After 55 iterations:
The final value for each state is:
 {'A': 1.0, 'B': 1.0, 'C': 7.75, 'D': 25.5, 'E': 1.0, 'F': 1.0, 'G': 1.0, 'H': 1.0,
  ↪ 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u',
  ↪ 'O': 'r'}

Q*( C ) for actions right, left, up, down, and exit are = dict_values([0.0, 0.0,
 ↪ 0.0, 7.75, 0.0])
After 55 iterations:
The final value for each state is:
 {'A': 1.0, 'B': 1.0, 'C': 7.75, 'D': 25.5, 'E': 1.0, 'F': 1.0, 'G': 1.0, 'H': 1.0,
  ↪ 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u',
  ↪ 'O': 'r'}

Q*( D ) for actions right, left, up, down, and exit are = dict_values([0.0, 0.0,
 ↪ 0.0, 25.5, 0.0])
After 55 iterations:
```

```
The final value for each state is:
 {'A': 1.0, 'B': 1.0, 'C': 7.75, 'D': 25.5, 'E': 1.0, 'F': 1.0, 'G': 1.0, 'H': 1.0,
  ↪ 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u',
  ↪ 'O': 'r'}

Q*( E ) for actions right, left, up, down, and exit are = dict_values([0.0, 1.0,
 ↪ 0.0, 0.0, 0.0])
After 55 iterations:
The final value for each state is:
 {'A': 1.0, 'B': 1.0, 'C': 7.75, 'D': 25.5, 'E': 1.0, 'F': 1.0, 'G': 1.0, 'H': 1.0,
  ↪ 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u',
  ↪ 'O': 'r'}

Q*( F ) for actions right, left, up, down, and exit are = dict_values([1.0, 1.0,
 ↪ 0.0, 0.0, 0.0])
After 55 iterations:
The final value for each state is:
 {'A': 1.0, 'B': 1.0, 'C': 7.75, 'D': 25.5, 'E': 1.0, 'F': 1.0, 'G': 1.0, 'H': 1.0,
  ↪ 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u',
  ↪ 'O': 'r'}

Q*( G ) for actions right, left, up, down, and exit are = dict_values([1.0, 0.0,
 ↪ 0.0, 0.0, 0.0])
After 55 iterations:
The final value for each state is:
 {'A': 1.0, 'B': 1.0, 'C': 7.75, 'D': 25.5, 'E': 1.0, 'F': 1.0, 'G': 1.0, 'H': 1.0,
  ↪ 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u',
  ↪ 'O': 'r'}

Q*( H ) for actions right, left, up, down, and exit are = dict_values([0.0, 0.0,
 ↪ 1.0, 1.0, 0.0])
After 55 iterations:
The final value for each state is:
 {'A': 1.0, 'B': 1.0, 'C': 7.75, 'D': 25.5, 'E': 1.0, 'F': 1.0, 'G': 1.0, 'H': 1.0,
  ↪ 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u',
  ↪ 'O': 'r'}

Q*( O ) for actions right, left, up, down, and exit are = dict_values([0.0, 0.0,
 ↪ 0.0, 0.0, 0.0])
After 55 iterations:
The final value for each state is:
 {'A': 1.0, 'B': 1.0, 'C': 7.75, 'D': 25.5, 'E': 1.0, 'F': 1.0, 'G': 1.0, 'H': 1.0,
  ↪ 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u',
  ↪ 'O': 'r'}
```

## Problem 4: Q*(A, down) and Q*(A, right) with $\gamma$=0.5

Assume all the non-exit actions with a success probability of p that it transitions to next state and acquires the reward along the edge, otherwise, the agent would state in the same state with a reward of 0. If the discount factor $\gamma$=0.5, please calculate the Q*(A, down) and Q*(A, right), where action 'down' refers that the agent move from A to H, while action 'right' refers that the agent moves from A to B.

**Answer:**

And the output of the code for value iteration with $\gamma$=0.5 and p = 0.5 will be:

Code output for Problem 3

```
Q*( A ) for actions right, left, up, down, and exit are = dict_values([0.6666666666666666, 0.0, 0.0, 0.6666666666666666, 0.0])
After 28 iterations:
The final value for each state is:
 {'A': 0.6666666666666666, 'B': 0.6666666666666666, 'C': 1.291666666666667, 'D': 25.166666666666668, 'E': 0.6666666666666666, 'F': 0.6666666666666666, 'G': 0.6666666666666666, 'H': 0.6666666666666666, 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u', 'O': 'r'}

Q*( B ) for actions right, left, up, down, and exit are = dict_values([-4.677083333333333, 0.6666666666666666, 0.0, 0.0, 0.0])
After 28 iterations:
The final value for each state is:
 {'A': 0.6666666666666666, 'B': 0.6666666666666666, 'C': 1.291666666666667, 'D': 25.166666666666668, 'E': 0.6666666666666666, 'F': 0.6666666666666666, 'G': 0.6666666666666666, 'H': 0.6666666666666666, 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u', 'O': 'r'}

Q*( C ) for actions right, left, up, down, and exit are = dict_values([0.0, 0.0, 0.0, 1.291666666666667, 0.0])
After 28 iterations:
The final value for each state is:
 {'A': 0.6666666666666666, 'B': 0.6666666666666666, 'C': 1.291666666666667, 'D': 25.166666666666668, 'E': 0.6666666666666666, 'F': 0.6666666666666666, 'G': 0.6666666666666666, 'H': 0.6666666666666666, 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u', 'O': 'r'}

Q*( D ) for actions right, left, up, down, and exit are = dict_values([0.0, 0.0, 0.0, 25.166666666666668, 0.0])
After 28 iterations:
The final value for each state is:
 {'A': 0.6666666666666666, 'B': 0.6666666666666666, 'C': 1.291666666666667, 'D': 25.166666666666668, 'E': 0.6666666666666666, 'F': 0.6666666666666666, 'G': 0.6666666666666666, 'H': 0.6666666666666666, 'O': 0.0}
And, the optimal policy is:
 {'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u', 'O': 'r'}

Q*( E ) for actions right, left, up, down, and exit are = dict_values([0.0, 0.6666666666666666, 0.0, 0.0, 0.0])
After 28 iterations:
The final value for each state is:
 {'A': 0.6666666666666666, 'B': 0.6666666666666666, 'C': 1.291666666666667, 'D': 25.166666666666668, 'E': 0.6666666666666666, 'F': 0.6666666666666666, 'G': 0.6666666666666666, 'H': 0.6666666666666666, 'O': 0.0}
And, the optimal policy is:
```

```
{'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u', 'O': 'r'}

Q*( F ) for actions right, left, up, down, and exit are = dict_values([0.6666666666666666, 0.6666666666666666, 0.0, 0.0, 0.0])
After 28 iterations:
The final value for each state is:
{'A': 0.6666666666666666, 'B': 0.6666666666666666, 'C': 1.291666666666667, 'D': 25.166666666666668, 'E': 0.6666666666666666, 'F': 0.6666666666666666, 'G': 0.6666666666666666, 'H': 0.6666666666666666, 'O': 0.0}
And, the optimal policy is:
{'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u', 'O': 'r'}

Q*( G ) for actions right, left, up, down, and exit are = dict_values([0.6666666666666666, 0.0, 0.0, 0.0, 0.0])
After 28 iterations:
The final value for each state is:
{'A': 0.6666666666666666, 'B': 0.6666666666666666, 'C': 1.291666666666667, 'D': 25.166666666666668, 'E': 0.6666666666666666, 'F': 0.6666666666666666, 'G': 0.6666666666666666, 'H': 0.6666666666666666, 'O': 0.0}
And, the optimal policy is:
{'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u', 'O': 'r'}

Q*( H ) for actions right, left, up, down, and exit are = dict_values([0.0, 0.0, 0.6666666666666666, 0.6666666666666666, 0.0])
After 28 iterations:
The final value for each state is:
{'A': 0.6666666666666666, 'B': 0.6666666666666666, 'C': 1.291666666666667, 'D': 25.166666666666668, 'E': 0.6666666666666666, 'F': 0.6666666666666666, 'G': 0.6666666666666666, 'H': 0.6666666666666666, 'O': 0.0}
And, the optimal policy is:
{'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u', 'O': 'r'}

Q*( O ) for actions right, left, up, down, and exit are = dict_values([0.0, 0.0, 0.0, 0.0, 0.0])
After 28 iterations:
The final value for each state is:
{'A': 0.6666666666666666, 'B': 0.6666666666666666, 'C': 1.291666666666667, 'D': 25.166666666666668, 'E': 0.6666666666666666, 'F': 0.6666666666666666, 'G': 0.6666666666666666, 'H': 0.6666666666666666, 'O': 0.0}
And, the optimal policy is:
{'A': 'r', 'B': 'l', 'C': 'd', 'D': 'd', 'E': 'l', 'F': 'r', 'G': 'r', 'H': 'u', 'O': 'r'}
```

# Appendix I

Value iteration code for the desired MDP problem.

Value Iteration Code

```
#!/usr/bin/env python

'''Pourya Shahverdi
    CSI-5130 Artificial Intelligence
    MDP
    '''

import sys

p = 1 #Probabiliy
GAMMA = 0.5 #Discount Factor
x = 1 #Reward on the edge from H to G (i.e.,  H ---- x ---> G)

all_states = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'O']
all_actions = ['r', 'l', 'u', 'd', 'e']

#Defining the Environment: environment(s, a) receives state-action pairs as
# input and returns their probabilities and rewards for the corresponding successor.
def environment(s, a):
    if   s == 'A' and a == 'd':
        return p, 1,'H'
    elif s == 'A' and a == 'd':
        return 1-p, 0, 'A'
    elif s == 'A' and a == 'r':
        return p, 1, 'B'
    elif s == 'A' and a == 'r':
        return 1-p, 0, 'A'
    elif s == 'B' and a == 'l':
        return p, 1, 'A'
    elif s == 'B' and a == 'l':
        return 1-p, 0, 'B'
    elif s == 'B' and a == 'r':
        return p, -10, 'C'
    elif s == 'B' and a == 'r':
        return 1-p, 0, 'B'
    elif s == 'C' and a == 'd':
        return p, -10, 'D'
    elif s == 'C' and a == 'd':
        return 1-p, 0, 'C'
    elif s == 'D' and a == 'd':
        return p, 50, 'E'
    elif s == 'D' and a == 'd':
        return 1-p, 0, 'D'
    elif s == 'E' and a == 'l':
        return p, 1, 'F'
    elif s == 'E' and a == 'l':
        return 1-p, 0, 'E'
    elif s == 'F' and a == 'r':
        return p, 1,  'E'
    elif s == 'F' and a == 'r':
        return 1-p, 0, 'F'
    elif s == 'F' and a == 'l':
        return p, 1, 'G'
    elif s == 'F' and a == 'l':
        return 1-p, 0, 'F'
    elif s == 'G' and a == 'r':
        return p, 1, 'F'
    elif s == 'G' and a == 'r':
```

```
59          return 1-p, 0, 'G'
60      elif s == 'H' and a == 'd':
61          return p, x, 'G'
62      elif s == 'H' and a == 'd':
63          return 1-p, 0, 'H'
64      elif s == 'H' and a == 'u':
65          return p, 1, 'A'
66      elif s == 'H' and a == 'u':
67          return 1-p, 0, 'H'
68      else:
69          return p, 0, 'O'
70
71  # Value iteration function:
72  def value_iteration(all_states, all_actions, s_a__p_r_suc):
73      V = {s: 0 for s in all_states}
74      k = 0
75      optimal_policy = {s: 0 for s in all_states}
76      f = open('value_iteration.txt', "w") #To clear the txt file.
77      f.close()
78      while True:
79          k = k+1
80          oldV = V.copy()
81          for s in all_states:
82              Q = {}
83              for a in all_actions:
84                  for suc in environment(s,a)[2]:
85                      Q[a] = environment(s,a)[0]*(environment(s,a)[1] + GAMMA *
                         ↪ oldV[suc])
86              # print("Q*(",s,") for actions right, left, up, down, and exit are
                 ↪ =",Q.values())
87              V[s] = max(Q.values())
88              optimal_policy[s] = max(Q, key=Q.get)
89          # print("Q*[A] = ", V['A'],"\n")
90          # print("Values of all states at the end of iteration #",k,"= \n", V)
91          # print("\n")
92          f = open('value_iteration.txt', '+a')
93          f.seek(0)
94          data = f.read()
95          if len(data) > 0:
96              f.write("\n")
97          print(V,optimal_policy, file = f)
98          f.close()
99          if all(oldV[s] == V[s] for s in all_states):
100             break
101     return V,k,optimal_policy
102
103
104 if __name__ == "__main__":
105     print("After",value_iteration(all_states, all_actions,
          ↪ environment)[1],"iterations:")
106     print("The final value for each state is:\n",value_iteration(all_states,
          ↪ all_actions, environment)[0])
107     print("And, the optimal policy is:\n", value_iteration(all_states, all_actions,
          ↪ environment)[2])
108
109     f = open('value_iteration.txt', '+a')
110     f.seek(0)
111     data = f.read()
112     if len(data) > 0:
113         f.write("\n")
114     print("After",value_iteration(all_states, all_actions,
          ↪ environment)[1],"iterations:",file=f)
115     print("The final value for each state is:\n",value_iteration(all_states,
          ↪ all_actions, environment)[0],file=f)
```

```
116     print("And, the optimal policy is:\n", value_iteration(all_states, all_actions,
          ↪ environment)[2],file=f)
```