Artificial Intelligence

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

Oakland University

Homework 2

§ Reinforcement Learning §

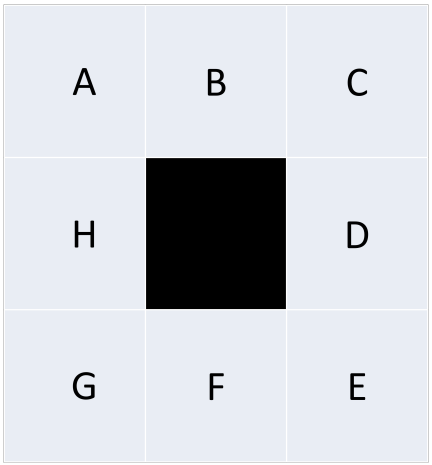


Figure 1: A 3x3 grid world with the center cell not accessible (colored in shade), there are states A through H.

Given 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 (right, left, up, down) and exit. An agent acting in this environment has recorded the following episodes:

Episode 1

s	a	s'	r
A	right	B	1
B	Right	C	-10
C	Down	D	-10
D	Down	E	50

Episode 2

s	A	s'	r
A	Right	B	1
B	Left	A	1
A	Right	B	1
B	Right	C	-10
C	Down	D	-10
D	Down	E	50

Episode 3

s	A	s'	r
A	down	H	1
H	down	G	1
G	Right	F	1
F	Right	E	1

Episode 4

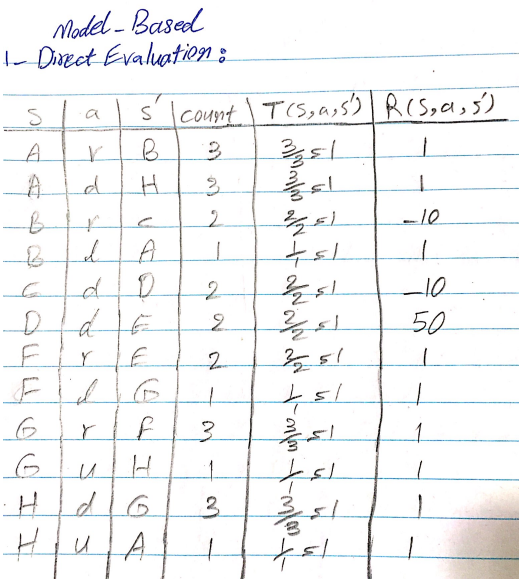
s	A	s'	R
A	down	H	1
H	up	A	1
A	down	H	1
H	down	G	1
G	up	H	1
H	down	G	1
G	Right	F	1
F	Left	G	1
G	Right	F	1
F	Right	E	1

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

Problem 1: Model-Based RL, Direct Evaluation

Consider the agent running model-based reinforcement learning (direct evaluation) based on the four episodes collected above, calculate the transition probabilities, and reward functions.

Answer:



Problem 2: Model-Based RL, Temporal Difference

Consider the agent running model-based reinforcement learning (temporal difference) based on the four episodes collected above, calculate the values of each state. We assume the discount factor $\gamma = 1/2$, and learning rate $\alpha = 1/2$.

Answer:
I've written a code to solve this problem as follows:

Value Iteration Code

```python
#!/usr/bin/env python
from math import gamma

alpha = 0.5
gamma = 0.5

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

# episode1 = [['A', 'r', 'B', 1],
#             ['B', 'r', 'C', -10],
#             ['C', 'd', 'D', -10],
#             ['D', 'd', 'E', 50]]

# episode2 = [['A', 'r', 'B', 1],
#             ['B', 'l', 'A', 1],
#             ['A', 'r', 'B', 1],
#             ['B', 'r', 'C', -10],
#             ['C', 'd', 'D', -10],
#             ['D', 'd', 'E', 50]]

# episode3 = [['A', 'd', 'H', 1],
#             ['H', 'd', 'G', 1],
#             ['G', 'r', 'F', 1],
#             ['F', 'r', 'E', 1]]

# episode4 = [['A', 'd', 'H', 1],
#             ['H', 'u', 'A', 1],
#             ['A', 'd', 'H', 1],
#             ['H', 'd', 'G', 1],
#             ['G', 'u', 'H', 1],
#             ['H', 'd', 'G', 1],
#             ['G', 'r', 'F', 1],
#             ['F', 'l', 'G', 1],
#             ['G', 'r', 'F', 1],
#             ['F', 'r', 'E', 1]]

episodes = [[['A', 'r', 'B', 1],
             ['B', 'r', 'C', -10],
             ['C', 'd', 'D', -10],
             ['D', 'd', 'E', 50]],
            [['A', 'r', 'B', 1],
             ['B', 'l', 'A', 1],
             ['A', 'r', 'B', 1],
             ['B', 'r', 'C', -10],
             ['C', 'd', 'D', -10],
             ['D', 'd', 'E', 50]],
            [['A', 'd', 'H', 1],
             ['H', 'd', 'G', 1],
             ['G', 'r', 'F', 1],
             ['F', 'r', 'E', 1]],
            [['A', 'd', 'H', 1],
```

```
            ['H', 'u', 'A', 1],
            ['A', 'd', 'H', 1],
            ['H', 'd', 'G', 1],
            ['G', 'u', 'H', 1],
            ['H', 'd', 'G', 1],
            ['G', 'r', 'F', 1],
            ['F', 'l', 'G', 1],
            ['G', 'r', 'F', 1],
            ['F', 'r', 'E', 1]]]

V = {state : 0 for state in all_states}

k = 0
for episode in episodes:
    for samples in episode:
        oldV = V.copy()
        for state in samples[0]:
            for action in samples[1]:
                for successor in samples[2]:
                    reward = samples[3]
                    V[state] = (1-alpha)*oldV[state] + (alpha * (reward + gamma *
                     ↪ V[successor]))
                    print('Value of States:',V)
                    print('Value of',successor,'as the successor
                     ↪ for',state,'is:',V[successor])
```

And the result is:

Code output for Problem 2

```
('Value of States:', {'A': 0.5, 'C': 0, 'B': 0, 'E': 0, 'D': 0, 'G': 0, 'F': 0, 'H':
 ↪ 0, 'O': 0})
('Value of', 'B', 'as the successor for', 'A', 'is:', 0)
('Value of States:', {'A': 0.5, 'C': 0, 'B': -5.0, 'E': 0, 'D': 0, 'G': 0, 'F': 0,
 ↪ 'H': 0, 'O': 0})
('Value of', 'C', 'as the successor for', 'B', 'is:', 0)
('Value of States:', {'A': 0.5, 'C': -5.0, 'B': -5.0, 'E': 0, 'D': 0, 'G': 0, 'F':
 ↪ 0, 'H': 0, 'O': 0})
('Value of', 'D', 'as the successor for', 'C', 'is:', 0)
('Value of States:', {'A': 0.5, 'C': -5.0, 'B': -5.0, 'E': 0, 'D': 25.0, 'G': 0,
 ↪ 'F': 0, 'H': 0, 'O': 0})
('Value of', 'E', 'as the successor for', 'D', 'is:', 0)
('Value of States:', {'A': -0.5, 'C': -5.0, 'B': -5.0, 'E': 0, 'D': 25.0, 'G': 0,
 ↪ 'F': 0, 'H': 0, 'O': 0})
('Value of', 'B', 'as the successor for', 'A', 'is:', -5.0)
('Value of States:', {'A': -0.5, 'C': -5.0, 'B': -2.125, 'E': 0, 'D': 25.0, 'G': 0,
 ↪ 'F': 0, 'H': 0, 'O': 0})
('Value of', 'A', 'as the successor for', 'B', 'is:', -0.5)
('Value of States:', {'A': -0.28125, 'C': -5.0, 'B': -2.125, 'E': 0, 'D': 25.0, 'G':
 ↪ 0, 'F': 0, 'H': 0, 'O': 0})
('Value of', 'B', 'as the successor for', 'A', 'is:', -2.125)
('Value of States:', {'A': -0.28125, 'C': -5.0, 'B': -7.3125, 'E': 0, 'D': 25.0,
 ↪ 'G': 0, 'F': 0, 'H': 0, 'O': 0})
('Value of', 'C', 'as the successor for', 'B', 'is:', -5.0)
('Value of States:', {'A': -0.28125, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D': 25.0,
 ↪ 'G': 0, 'F': 0, 'H': 0, 'O': 0})
('Value of', 'D', 'as the successor for', 'C', 'is:', 25.0)
('Value of States:', {'A': -0.28125, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D': 37.5,
 ↪ 'G': 0, 'F': 0, 'H': 0, 'O': 0})
('Value of', 'E', 'as the successor for', 'D', 'is:', 0)
('Value of States:', {'A': 0.359375, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D': 37.5,
 ↪ 'G': 0, 'F': 0, 'H': 0, 'O': 0})
('Value of', 'H', 'as the successor for', 'A', 'is:', 0)
('Value of States:', {'A': 0.359375, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D': 37.5,
```

```
 ↪ 'G': 0, 'F': 0, 'H': 0.5, 'O': 0})
('Value of', 'G', 'as the successor for', 'H', 'is:', 0)
('Value of States:', {'A': 0.359375, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D': 37.5,
 ↪ 'G': 0.5, 'F': 0, 'H': 0.5, 'O': 0})
('Value of', 'F', 'as the successor for', 'G', 'is:', 0)
('Value of States:', {'A': 0.359375, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D': 37.5,
 ↪ 'G': 0.5, 'F': 0.5, 'H': 0.5, 'O': 0})
('Value of', 'E', 'as the successor for', 'F', 'is:', 0)
('Value of States:', {'A': 0.8046875, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D': 37.5,
 ↪ 'G': 0.5, 'F': 0.5, 'H': 0.5, 'O': 0})
('Value of', 'H', 'as the successor for', 'A', 'is:', 0.5)
('Value of States:', {'A': 0.8046875, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D': 37.5,
 ↪ 'G': 0.5, 'F': 0.5, 'H': 0.951171875, 'O': 0})
('Value of', 'A', 'as the successor for', 'H', 'is:', 0.8046875)
('Value of States:', {'A': 1.14013671875, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D':
 ↪ 37.5, 'G': 0.5, 'F': 0.5, 'H': 0.951171875, 'O': 0})
('Value of', 'H', 'as the successor for', 'A', 'is:', 0.951171875)
('Value of States:', {'A': 1.14013671875, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D':
 ↪ 37.5, 'G': 0.5, 'F': 0.5, 'H': 1.1005859375, 'O': 0})
('Value of', 'G', 'as the successor for', 'H', 'is:', 0.5)
('Value of States:', {'A': 1.14013671875, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D':
 ↪ 37.5, 'G': 1.025146484375, 'F': 0.5, 'H': 1.1005859375, 'O': 0})
('Value of', 'H', 'as the successor for', 'G', 'is:', 1.1005859375)
('Value of States:', {'A': 1.14013671875, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D':
 ↪ 37.5, 'G': 1.025146484375, 'F': 0.5, 'H': 1.30657958984375, 'O': 0})
('Value of', 'G', 'as the successor for', 'H', 'is:', 1.025146484375)
('Value of States:', {'A': 1.14013671875, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D':
 ↪ 37.5, 'G': 1.1375732421875, 'F': 0.5, 'H': 1.30657958984375, 'O': 0})
('Value of', 'F', 'as the successor for', 'G', 'is:', 0.5)
('Value of States:', {'A': 1.14013671875, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D':
 ↪ 37.5, 'G': 1.1375732421875, 'F': 1.034393310546875, 'H': 1.30657958984375, 'O':
 ↪ 0})
('Value of', 'G', 'as the successor for', 'F', 'is:', 1.1375732421875)
('Value of States:', {'A': 1.14013671875, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D':
 ↪ 37.5, 'G': 1.3273849487304688, 'F': 1.034393310546875, 'H': 1.30657958984375,
 ↪ 'O': 0})
('Value of', 'F', 'as the successor for', 'G', 'is:', 1.034393310546875)
('Value of States:', {'A': 1.14013671875, 'C': -1.25, 'B': -7.3125, 'E': 0, 'D':
 ↪ 37.5, 'G': 1.3273849487304688, 'F': 1.0171966552734375, 'H': 1.30657958984375,
 ↪ 'O': 0})
('Value of', 'E', 'as the successor for', 'F', 'is:', 0)
```

Problem 3: Model-Free RL, Q-Learning

Consider the agent running model-free reinforcement learning (Q-learning) based on the four episodes being repeatedly in an infinite sequence E1, E2, E3, E4, E1, E2, E3, E4.... , calculate the Q-values of each <state, action>. We assume the discount factor $\gamma = 1$, and learning rate $\alpha = 1/2$. [You are encouraged to compute this question with programming assistance.]

Answer:
I've written a code for this problem as follows:

Value Iteration Code

```python
#!/usr/bin/env python
from math import gamma


alpha = 0.5
gamma = 1
```

```

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

episodes = [[['A', 'r', 'B', 1],
             ['B', 'r', 'C', -10],
             ['C', 'd', 'D', -10],
             ['D', 'd', 'E', 50]],
            [['A', 'r', 'B', 1],
             ['B', 'l', 'A', 1],
             ['A', 'r', 'B', 1],
             ['B', 'r', 'C', -10],
             ['C', 'd', 'D', -10],
             ['D', 'd', 'E', 50]],
            [['A', 'd', 'H', 1],
             ['H', 'd', 'G', 1],
             ['G', 'r', 'F', 1],
             ['F', 'r', 'E', 1]],
            [['A', 'd', 'H', 1],
             ['H', 'u', 'A', 1],
             ['A', 'd', 'H', 1],
             ['H', 'd', 'G', 1],
             ['G', 'u', 'H', 1],
             ['H', 'd', 'G', 1],
             ['G', 'r', 'F', 1],
             ['F', 'l', 'G', 1],
             ['G', 'r', 'F', 1],
             ['F', 'r', 'E', 1]]]

Qmax = {}
Q = {}
for state in all_states:
    for action in all_actions:
        Qmax[state,action] = 0
        Q[state,action] = 0

k = 0
while k < 100:
    for episode in episodes:
        for samples in episode:
            Old_Q = Q.copy()
            for state in samples[0]:
                for action in samples[1]:
                    for successor in samples[2]:
                        reward = samples[3]
                        Q[state,action] = (1-alpha)*Old_Q[state,action] +
                         ↪ alpha*(reward + gamma * Qmax[successor,action])
                        for act in all_actions:
                            if Q[state,act] > Qmax[state,action]:
                                Qmax[state,action] = Q[state,act]
                    print(Q)
    k += 1
```

And the result after 100 iteration is:

Code output for Problem 3

```
{('F', 'u'): 0, ('H', 'l'): 0, ('C', 'r'): 0, ('D', 'd'): 50.0, ('E', 'r'): 0, ('E',
 ↪ 'e'): 0, ('B', 'u'): 0, ('E', 'l'): 0, ('F', 'r'): 1.0, ('C', 'l'): 0, ('A',
 ↪ 'e'): 0, ('H', 'd'): 1.0, ('H', 'r'): 0, ('G', 'e'): 0, ('D', 'e'): 0, ('O',
 ↪ 'd'): 0, ('B', 'l'): 1.0, ('E', 'd'): 0, ('C', 'u'): 0, ('F', 'd'): 0, ('A',
 ↪ 'd'): 2.0, ('B', 'r'): -10.0, ('H', 'e'): 0, ('O', 'u'): 0, ('E', 'u'): 0, ('G',
 ↪ 'd'): 0, ('B', 'd'): 0, ('G', 'r'): 2.0, ('A', 'r'): 2.0, ('F', 'l'): 1.0, ('A',
 ↪ 'l'): 0, ('F', 'e'): 0, ('G', 'u'): 2.0, ('D', 'u'): 0, ('G', 'l'): 0, ('B',
```

```
↪ 'e'): 0, ('C', 'e'): 0, ('H', 'u'): 1.0, ('A', 'u'): 0, ('O', 'e'): 0, ('D',
↪ 'l'): 0, ('O', 'l'): 0, ('D', 'r'): 0, ('O', 'r'): 0, ('C', 'd'): 40.0}
```

Looks like numbers will be converged after like 54 iterations.