COMP4418, 2017 – Assignment 3

Chunnan Sheng

Student Code: **z5100764**

1. Social Choice and Game Theory

(a)

Alternatives	Nodes can be reached within 2 steps
a	b,d,c,e,f,g
b	c,d,a,e,f,g
c	a,d,g,b,e,f
d	e,f,g,a,b
e	a,f,g,b,d
f	b,g,c,d,e
g	

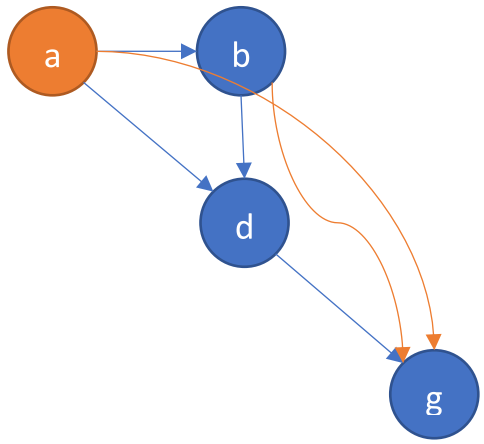
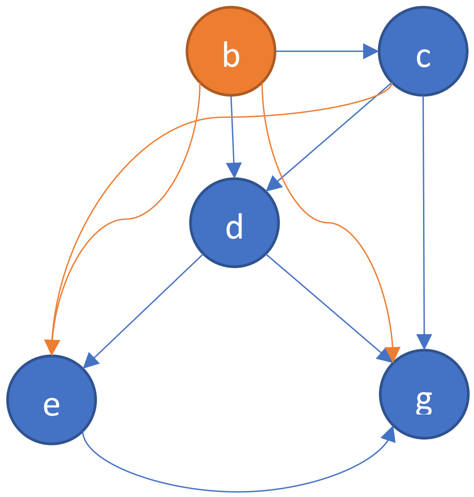
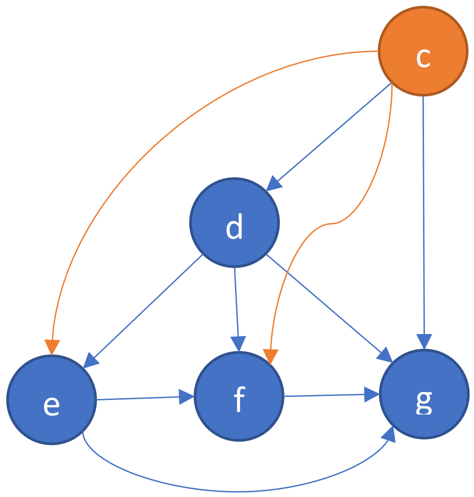
The uncovered set: {a, b, c}

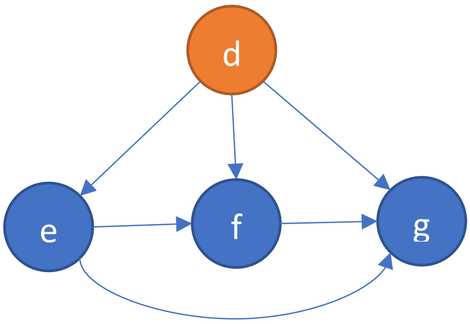
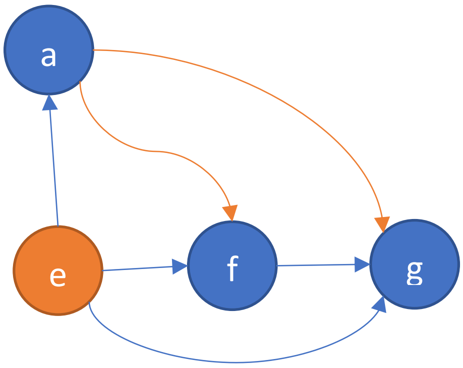
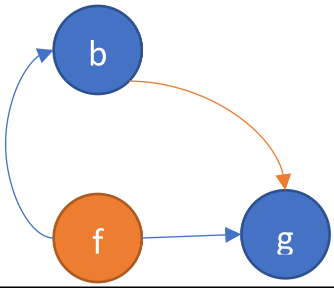
Alternatives	One of the paths to visit all other nodes
a	a, b, c, d, e, f, g
b	b, c, a, d, e, f, g
c	c, a, b, d, e, f, g
d	d, e, a, b, c, f, g
e	e, a, b, c, d, f, g
f	f, b, c, a, d, e, g
g	

The top cycle:
…->b->c->d->e->a->f->…

Alternatives	Dominates:	Copeland Score
a	(a,b), (a,d), (a, g), (a, f)	4
b	(b,c), (b,d), (b,e), (b,g)	4
c	(c,a), (c,d), (c,g), (c,e), (c,f)	5
d	(d,e), (d,f), (d,g)	3
e	(e,a), (e,f), (e,g)	3
f	(f,b), (f,g)	2
g		0

The set of Copeland winners: {c}

Alternatives	Max acyclic sub graph where this alternative is on top
a	
b	
c	

d	
e	
f	
g	

The set of Banks winner: {b, c}

Alternatives	Has parent(s)
a	yes
b	yes
c	yes
d	yes
e	yes
f	yes
g	yes

Condorcet winner doesn’t allow parents, which means the set of Condorcet winners is:
{} = ∅

(b) Compute all the Nash equilibria of the following two player game

	D	E
A	2,4	8,5
B	6,6	4,4

(A, D)

$u_1(A,D)=2$

$u_1(B,D)=6$

then

$u_1(A,D)<u_1(B,D)$

So (A, D) is not a Nash equilibrium.

(A, E)

$u_1(A,E)=8$

$u_1(B,E)=4$

$u_2(A,E)=5$

$u_2(A,D)=4$

Then

$u_1(A,E)>u_1(B,E)$

$u_2(A,E)>u_2(A,D)$

So (A, E) is a Nash equilibrium.

(B, D)

$u_1(B,D)=6$

$u_1(A,D)=2$

$u_2(B,D)=6$

$u_2(B,E)=4$

Then

$u_1(B,D)>u_1(A,D)$

$u_2(B,D)>u_2(B,E)$

So, (B, D) is a Nash equilibrium.

(B, E)

$u_1(B,E)=4$

$u_1(A,E)=8$

$u_1(B,E)<u_1(A,E)$

So, (B, E) is not a Nash equilibrium.

The set of Nash equilibria:

{(A, E), (B, D)}

<table>
<tr><td colspan="3" rowspan="3"></td><td colspan="2">Player2</td></tr>
<tr><td>q</td><td>1-q</td></tr>
<tr><td>D</td><td>E</td></tr>
<tr><td rowspan="2">Player1</td><td>p</td><td>A</td><td>2,4</td><td>8,5</td></tr>
<tr><td>1-p</td><td>B</td><td>6,6</td><td>4,4</td></tr>
</table>

Rewards of player1's actions if probability of player2's action D is q :

$$2q+8(1-q)=6q+4(1-q)$$

Rewards of player2's actions if probability of player1's action A is p :

$$4p+6(1-p)=5p+4(1-p)$$

$$p=\frac{2}{3}$$

$$q=\frac{1}{2}$$

Then the mixed policy is:

$$Player1=(2/3,1/3),\ Player2=(1/2,1/2)$$

2. Decision Making

(a)

Games	Model
Blackjack	(B) Markov decision process (MDP)
Candy Crush	(B) Markov decision process (MDP)
Chess	(E) None/Other
Minesweeper	(D) Partially-observable Markov decision process (POMDP)
Snakes and Ladders	(A) Markov chain
Texas Holdem Poker	(E) None/Other

(b)

Policies Stay=S Leave=L	**Expected utility**	**When δ is 0.999**
π_{SSS}	$V(s_1)=\frac{1}{1-\delta}$	**1000**
	$V(s_2)=0$	**0**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2000**
π_{SSL}	$V(s_1)=\frac{1}{1-\delta}$	**1000**
	$V(s_2)=0$	**0**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2000**
π_{SLS}	$V(s_1)=\frac{1}{1-\delta}$	**1000**
	$V(s_2)=5+\delta(0.5V(s_1)+0.5V(s_3))=5-\frac{0.5\delta}{1-\delta}$	**-494.5**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2000**
π_{SLL}	$V(s_1)=\frac{1}{1-\delta}$	**1000**
	$V(s_2)=5+\delta(0.5V(s_1)+0.5V(s_3))=5-\frac{0.5\delta}{1-\delta}$	**-494.5**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2000**
π_{LSS}	$V(s_1)=\delta V(s_2)=0$	**0**
	$V(s_2)=0$	**0**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2000**

π_{LSL}	$V(s_1)=\delta V(s_2)=0$	**0**
	$V(s_2)=0$	**0**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2000**
π_{LLS}	$V(s_1)=\delta V(s_2)=\frac{\delta(5-6\delta)}{(1-\delta)(1-0.5\delta^2)}$	**-1982**
	$V(s_2)=5+\delta(0.5V(s_1)+0.5V(s_3))=\frac{5-6\delta}{(1-\delta)(1-0.5\delta^2)}$	**-1984**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2000**
π_{LLL}	$V(s_1)=\delta V(s_2)=\frac{\delta(5-6\delta)}{(1-\delta)(1-0.5\delta^2)}$	**-1982**
	$V(s_2)=5+\delta(0.5V(s_1)+0.5V(s_3))=\frac{5-6\delta}{(1-\delta)(1-0.5\delta^2)}$	**-1984**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2000**

π_{SSS} (stay-stay-stay) and π_{SSL} (stay-stay-leave) are the best policies in this case. If δ is close to 1, long term benefits would be more important than short term rewards. Thus, though the reward of leaving S_2 is attractive, it is risky. To avoid getting trapped in S_3 , the best choice is to stay at S_1 and S_2 .

(c)

Policies **Stay=S** **Leave=L**	**Expected utility**	**When δ is 0.001**
π_{SSS}	$V(s_1)=\frac{1}{1-\delta}$	**1.001**
	$V(s_2)=0$	**0**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2.002**
π_{SSL}	$V(s_1)=\frac{1}{1-\delta}$	**1.001**
	$V(s_2)=0$	**0**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2.002**
π_{SLS}	$V(s_1)=\frac{1}{1-\delta}$	**1.001**

	$V(s_2)=5+\delta(0.5\mathrm{V}(s_1)+0.5\mathrm{V}(s_3))=5-\frac{0.5\delta}{1-\delta}$	**4.999**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2.002**
π_{SLL}	$V(s_1)=\frac{1}{1-\delta}$	**1.001**
	$V(s_2)=5+\delta(0.5\mathrm{V}(s_1)+0.5\mathrm{V}(s_3))=5-\frac{0.5\delta}{1-\delta}$	**4.999**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2.002**
π_{LSS}	$V(s_1)=\delta V(s_2)=0$	**0**
	$V(s_2)=0$	**0**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2.002**
π_{LSL}	$V(s_1)=\delta V(s_2)=0$	**0**
	$V(s_2)=0$	**0**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2.002**
π_{LLS}	$V(s_1)=\delta V(s_2)=\frac{\delta(5-6\delta)}{(1-\delta)(1-0.5\delta^2)}$	**0.005**
	$V(s_2)=5+\delta(0.5\mathrm{V}(s_1)+0.5\mathrm{V}(s_3))=\frac{5-6\delta}{(1-\delta)(1-0.5\delta^2)}$	**4.999**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2.002**
π_{LLL}	$V(s_1)=\delta V(s_2)=\frac{\delta(5-6\delta)}{(1-\delta)(1-0.5\delta^2)}$	**0.005**
	$V(s_2)=5+\delta(0.5\mathrm{V}(s_1)+0.5\mathrm{V}(s_3))=\frac{5-6\delta}{(1-\delta)(1-0.5\delta^2)}$	**4.999**
	$V(s_3)=\frac{-2}{1-\delta}$	**-2.002**

π_{SLS} (stay-leave-stay) and π_{SLL} (stay-leave-leave) are the best policies in this case. If δ is close to zero, short term benefits would be more important than long term rewards. Thus, the probability of going into the trap of s_3 is very low, the agent can make most rewardable decisions without considering any risks.

(d)

A short Python program is implemented to do the calculation

```python
def v_func(mdp, step, state, cache):
    """
    The value function of a state
    """
    if step == 0:
        return 0.0

    next_val1 = cache[step - 1][state - 1][0]
    if next_val1 is None:
        next_val1 = v_func_act(mdp, step - 1, state, 'stay', cache)
    next_val2 = cache[step - 1][state - 1][1]
    if next_val2 is None:
        next_val2 = v_func_act(mdp, step - 1, state, 'leave', cache)

    return max(next_val1, next_val2)


def v_func_act(mdp, step, state, act, cache):
    """
    The value function of a state and an action
    """
    reward, result = mdp[state][act]
    next_val = reward
    sum = 0.0
    for prob, next_state in result:
        sum += prob * v_func(mdp, step, next_state, cache)

    next_val += sum * 0.6

    if act == 'stay':
        cache[step][state - 1][0] = next_val
    else:
        cache[step][state - 1][1] = next_val

    return next_val


if __name__ == "__main__":
    # mdp defines the Markov Decision Process
    mdp = {1:{'stay':(1.0, [(1, 1)]),  'leave':(0.0, [(1, 2)])},\
           2:{'stay':(0.0, [(1, 2)]),  'leave':(5.0, [(0.5, 1), (0.5, 3)])},\
           3:{'stay':(-2.0, [(1, 3)]), 'leave':(-2.0, [(1, 3)])}}

    # This cache is to help improve performance
    # The time complexity would be O(2^n) without this cache
    cache = [];
    for i in range(1000):
        states = []
        for j in range(3):
            actions = []
            for k in range(2):
                actions.append(None)
            states.append(actions)
        cache.append(states)
    # Print the table of value functions
    for state in range(1, 4):
        for step in range(4):
            print('V{0}(s{1}): {2}'.format(step, state, round(v_func(mdp, step, state, cache), 5)), end='\t')
            print('V{0}(s{1}, S): {2}'.format(step, state, round(v_func_act(mdp, step, state, 'stay', cache), 5)), end='\t')
            print('V{0}(s{1}, L): {2}'.format(step, state, round(v_func_act(mdp, step, state, 'leave', cache), 5)), end='\t')
        print()
    # Check the value function after many steps
    for state in range(1, 4):
        print('V{0}(s{1}): {2}'.format(500, state, v_func(mdp, 500, state, cache)))
```

	$V_0(s)$	$V_0(s,S)$	$V_0(s,L)$	$V_1(s)$	$V_1(s,S)$	$V_1(s,L)$	$V_2(s)$	$V_2(s,S)$	$V_2(s,L)$	$V_3(s)$
S_1	0.0	1.0	0.0	1.0	1.6	3.0	3.0	2.8	2.82	2.82
S_2	0.0	0.0	5.0	5.0	3.0	4.7	4.7	2.82	4.94	4.94
S_3	0.0	-2.0	-2.0	-2.0	-3.2	-3.2	-3.2	-3.92	-3.92	-3.92

According to the Python program, the values finally converge to:

$V(s_1)=2.5610$

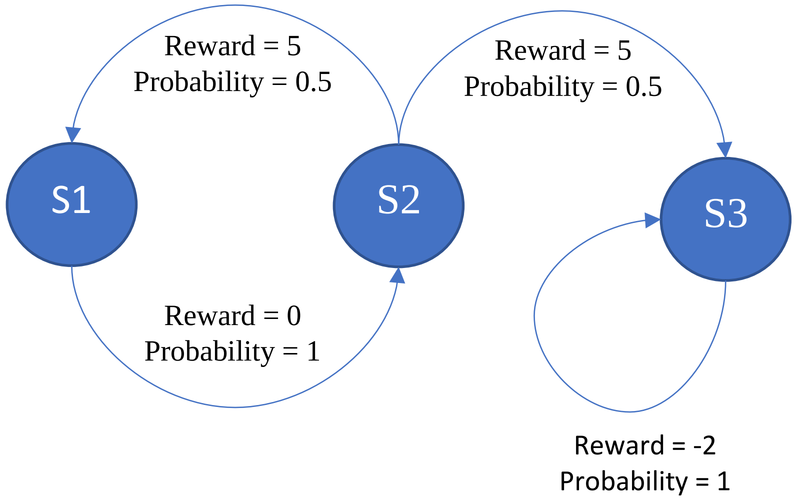
$V(s_2)=4.2683$

$V(s_3)=-5.000$

These values are consistent with π_{LLS} (leave-leave-stay), and π_{LLL} (leave-leave-leave).

(e)

The leave-leave-stay policy:



(f)

Value functions of all policies are listed in answers of (b) and (c).

Value functions of leave-leave-stay policy:

π_{LLS}	$V(s_1)=\delta V(s_2)=\frac{\delta(5-6\delta)}{(1-\delta)(1-0.5\delta^2)}$
	$V(s_2)=5+\delta(0.5V(s_1)+0.5V(s_3))=\frac{5-6\delta}{(1-\delta)(1-0.5\delta^2)}$
	$V(s_3)=\frac{-2}{1-\delta}$

According to evaluations of all policies in circumstances (b) and (c), leave-leave-stay policy is neither optimal in (b), nor optimal in (c).