TDT4171 Assignment 4

Bjørn Kåre

March 10, 2020

Contents

1 Problem 1

1.1 A)

- Gabriel is risk-seeking, as his utility function is convex.
- Maria is risk-averse, as her utility function is concave.
- Both Gustav and Sonja are risk-neutral as they have a linear utility function

1.2 B)

	Risk-seeking	Risk-averse
Lottery	[(0.5, 10), (0.5,20)]	[(0.5, -3), (0.5, -6)]
Expected utility of lottery	4500	-94.5
Utility of expected monetary value	3375	-91.125

X^3 is risk-seeking for positive utilities, and risk-averse for negative ones. As we can see, the expected utility of the lottery is higher than the utility of the expected monetary value. And the inverse is true for a lottery with negative rewards. Lottery A has 50% chance of winning 10$ and 50% of 20$. The expected reward is 0.5*10 + 0.5*20 = 15, wich has a utility of 15^3 = 3375. But the expected utility of the lottery is 0.5*10^3 + 0.5*20^3 = 4500 wich is higher, as the utility function is nonlinear and convex for positive numbers, thereby weighting larger values more.

The opposite happens with the negative values, where larger negative numbers are "punished" more by the utility function. Here: $0.5 * (-3)^3 * 0.5(-6)^3 = -94.5 < (-4.5)^3 = -91.125$

2 Problem 2

2.1 A) Decision Network

1 shows the belief network with probability tables for m and b.

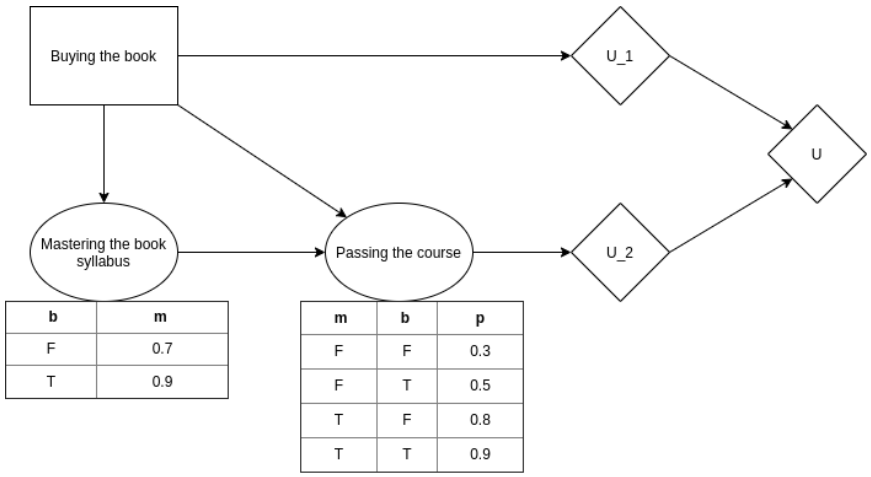

Figure 1: Belief network for task 2A

2.2 B)

2.2.1 Expected utility of buying the book (B=b):

P(m) = P(m|b) = 0.9
P(p) = P(m) * P(p|m,b) + P(¬ m)*P(p|¬ m,b) = 0.9*0.9 + 0.1*0.5 = 0.86

$$EU = EU_1 + EU_2 = U_1(b) + EU_2(p) = -100 + 0.86 * 2000 = 1620 \quad (1)$$

2.2.2 Expected utility of not buying the book (B = ¬ b):

P(m) = P(m|¬ b) = 0.7 P(p) = P(m) * P(p|m,¬ b) + P(¬ m)*P(p|¬ m,¬ b) = 0.7*0.8 + 0.3*0.3 = 0.65

$$EU = EU_1 + EU_2 = U_1(\not{b}) + EU_2(p) = 0 + 0.65 * 2000 = 1300 \quad (2)$$

2.3 C)

Based on the stated preferences, Sam should by the book as this gives a higher expected utility

3 Problem 3

3.1 A)

Step	1	2	3
Initial	U[1] = 0	U[2] = 0	U[3] = 0
Iteration 1	U[1] = 0	U[2] = 1	U[3] = 0
Iteration 2	U[1] = 0.375	U[2] = 1	U[3] = 0.375

3.1.1 Iteration 1:

$$U[1] = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s') = 0 \tag{3}$$

$$U[2] = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s') = 1 + 0 = 1 \tag{4}$$

$$U[3] = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s') = 0 \tag{5}$$

3.1.2 Iteration 2:

$$U[1] = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s') \tag{6}$$

$$= 0 + 0.5 * 0.75 * 1 \tag{7}$$

$$= 0.375 \tag{8}$$

$$U[2] = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s') \tag{9}$$

$$= 1 + 0 \tag{10}$$

$$= 1 \tag{11}$$

$$U[3] = R(s) + \gamma \max_{a \in A(s)} \sum_{s'} P(s'|s,a) U_i(s') \tag{12}$$

$$= 0 + 0.5 * 0.75 * 1 \tag{13}$$

$$= 0.375 \tag{14}$$

3.2 B)

$U_1 = U_3 = 0.5$, $U_2 = 1.25$
Expected utility of R: 0.75 * 1.25 + 0.25 * 0.50 = 1.0625
Expected utility of L: 0.75 * 0.50 + 0.25 * 1.25 = 0.6875
=> R is the best choice

4 Problem 4

4.1 A)

4.1.1 value iteration

```
def value_iteration() -> Any:
    gamma = constants.gamma
    epsilon = constants.epsilon
    tol = epsilon * (1-gamma)/gamma
    number_states = constants.number_states

    U = [0 for i in range(16)] # State utilities U[s] := utility of state s
    U_p =U[:] # U'
    delta = 2*tol  # To make sure first iteration runs without do{...}while();

    while(delta > tol):
        U = U_p[:]
        delta = 0
        for s in range(number_states):
            U_p[s] = get_reward(s) + gamma *
                max( [sum( [
                get_transition_probability(s, a, sp)*U[sp]
                for sp in get_outcome_states(s, a)]) for a in moves
                ])
            delta = max([abs(U_p[s] - U[s]), delta])
    return U
```

4.2 B)

The resulting output printed with

4.2.1 Printing values

```
print(tabulate([['{0:.3f}'.format(v) for v in value_table[i:i+4]]
  for i in range(0,16,4)], showindex="always", tablefmt="orgtbl"))
```

4.2.2 Result

gives the following output (indexes included as the first row, utilities rounded)

0	1.134	0.476	1.457	0.164
1	1.513	-10	2.083	-10
2	3.301	5.784	5.532	-10
3	-10	7.068	8.349	10

4.3 C)

4.3.1 Policy iteration

Implementation of policy iteration:

```
def extract_policy(value_table: Any) -> Any:
    PI = ["left" for i in range(16)]
    unchanged = False
    while not unchanged:
        unchanged = True
        for s in range(constants.number_states):
            for a in moves:
                action_util = sum( [get_transition_probability(s, a, sp)\
                *value_table[sp] for sp in get_outcome_states(s, a)])

                policy_util = sum( [get_transition_probability(s, PI[s], sp)\
                *value_table[sp] for sp in get_outcome_states(s, PI[s])])

                if action_util > policy_util:
                    unchanged = False
                    PI[s] = a
    return PI
```

4.3.2 Printing result

and to print in a pretty way i used the following code in main, replcing holes with Xs and the goal with a G

```
# in main
optimal_policy = extract_policy(value_table)

move_arrow: Dict[str, str] = {"left": "<", "down": "v", "right": ">", "up": "^"}
optimal_arrows = [move_arrow[action] for action in optimal_policy]

for i in range(constants.number_states):
    if value_table[i] < -5:
        optimal_arrows[i] = "x"

optimal_arrows[-1] = "G"

print(tabulate([optimal_arrows[i:i+4] for i in range(0,16,4)],
showindex="always", tablefmt="orgtbl"))
```

4.3.3 Result

This gave the following optimal policy for getting to work

0	v	^	v	^
1	v	x	v	x
2	>	v	v	x
3	x	>	>	G