RMIT UNIVERSITY

COSC2192 Artificial Intelligence

Mentor: Ms. Thuy Nguyen

Assignment 3

Reinforcement Learning & Bayesian Network

(Practical and Written, Individual)

Student ID:

Huynh Ngoc Tai - s3978680

Question 1: MDP and Reinforcement Learning

1.1 What is the optimal policy for this grid?

According to this grid, we have:

- Start State (S_0) = (1,1).
- States = (1,1), (1,2), (1,3), (2,1), (2,2), (2,3).
- Terminal states:
  + (2,3) with reward + 5.
  + (1,3) with reward -5.
- Possible actions: North, South, East, West.
- The reward for non-terminals is 0.

According to the given grid (figure 1), the most optimal policy is as follows:

- Π*: State -> Action
- Π*: (1,1) -> N (Move North)
- Π*: (1,2) -> E (Move East)
- Π*: (2,1) -> W (Move West)
- Π*: (2,2) -> E (Move East)
- Π*: (3,2) -> Terminal State (+5)
- Π*: (3,1) -> Terminal State (-5)

1.2 Suppose the agent knows the transition probabilities. Give the first two rounds of value iteration updates for each state, with a discount of 0.6. (Assume V0 is 0 everywhere and compute Vi for times i = 1, 2).

Value Iteration Formula:

$$U_{i+1}(s) = R(s) + \gamma \left(\max_a \sum_{s' \in S} T(s, a, s') U_i(s') \right)$$

Value iteration for I = 1:

- State (3,2): $U_1(2,3)$ = +5 (Reward and Terminal state)

- State (3,2): $U_1(1,3)$ = -5 (Reward and Terminal state)

- State (1,1): $U_1(1,1)$ can move 4 directions, so:

Direction	
North:	$U = 0.8 \cdot U_0(1,2) + 0.1 \cdot U_0(1,1) + 0.1 \cdot U_0(2,1) = 0$
East:	$U = 0.8 \cdot U_0(2,1) + 0.1 \cdot U_0(1,1) + 0.1 \cdot U_0(1,2) = 0$
South:	$U = 0.8 \cdot U_0(1,1) + 0.1 \cdot U_0(1,1) + 0.1 \cdot U_0(2,1) = 0$
West:	$U = 0$

Then $U_1(1,1) = R(1,1) + \text{discount} * \max(U_{North}, U_{South}, U_{East}, U_{West})$

$= 0 + 0.6 * 0 = 0$

- State (1,2): $U_1(1,2)$ can move 4 directions, so:

Direction	
North:	$U = 0.8 \cdot U_0(1,2) + 0.1 \cdot U_0(1,2) + 0.1 \cdot U_0(2,2) = 0$
East:	$U = 0.8 \cdot U_0(2,2) + 0.1 \cdot U_0(1,2) + 0.1 \cdot U_0(1,1) = 0$
South:	$U = 0.8 \cdot U_0(1,1) + 0.1 \cdot U_0(1,2) + 0.1 \cdot U_0(2,2) = 0$
West:	$U = 0$

Then $U_1(1,2) = R(1,2) + \text{discount} * \max(U_{North}, U_{South}, U_{East}, U_{West})$

$= 0 + 0.6 * 0 = 0$

- State (2,1): $U_1(2,1)$ can move 4 directions, so:

Direction	
North:	$U = 0.8 \cdot U_0(2,2) + 0.1 \cdot U_0(1,1) + 0.1 \cdot U_0(3,1) = 0 + 0 - 5 * 0.1 = -0.5$
East:	$U = 0.8 \cdot U_0(3,1) + 0.1 \cdot U_0(2,1) + 0.1 \cdot U_0(2,2) = 0.8*-5 + 0 + 0 = -4$
South:	$U = 0.8 \cdot U_0(2,1) + 0.1 \cdot U_0(1,1) + 0.1 \cdot U_0(3,1) = 0 + 0 - 5 * 0.1 = -0.5$
West:	$U = 0.8 \cdot U_0(1,1) + 0.1 \cdot U_0(2,1) + 0.1 \cdot U_0(2,2) = 0$

Then $U_1(2,1) = R(2,1) + \text{discount} * \max(U_{North}, U_{South}, U_{East}, U_{West})$

$= 0 + 0.6 * 0 = 0$

- State (2,2): $U_1(2,2)$ can move 4 directions, so:

Direction	
North:	$U = 0.8 \cdot U_0(2,2) + 0.1 \cdot U_0(1,2) + 0.1 \cdot U_0(3,2) = 0 + 0 + 0.1*5 = 0.5$
East:	$U = 0.8 \cdot U_0(3,2) + 0.1 \cdot U_0(2,2) + 0.1 \cdot U_0(2,1) = 0.8*5 + 0 + 0 = 4$
South:	$U = 0.8 \cdot U_0(2,1) + 0.1 \cdot U_0(1,2) + 0.1 \cdot U_0(3,2) = 0 + 0 + 0.1*5 = 0.5$
West:	$U = 0.8 \cdot U_0(1,2) + 0.1 \cdot U_0(2,2) + 0.1 \cdot U_0(2,1) = 0$

Then $U_1(2,2) = R(2,2) + \text{discount} * \max(U_{North}, U_{South}, U_{East}, U_{West})$

$= 0 + 0.6 * 4 = 2.4$

Value iteration for i = 2:

- State (3,2): $U_2(2,3) = +5$ (Reward and Terminal state)

- State (3,1): $U_2(1,3) = -5$ (Reward and Terminal state)

- State (1,1): $U_2(1,1)$ can move 4 directions, so:

Direction	
North:	$U = 0.8 \cdot U_1(1,2) + 0.1 \cdot U_1(1,1) + 0.1 \cdot U_1(2,1) = 0$
East:	$U = 0.8 \cdot U_1(2,1) + 0.1 \cdot U_1(1,1) + 0.1 \cdot U_1(1,2) = 0$
South:	$U = 0.8 \cdot U_{1(}1,1) + 0.1 \cdot U_1(1,1) + 0.1 \cdot U_1(2,1) = 0$
West:	$U = 0$

Then $U_2(1,1) = R(1,1) + \text{discount} * \max(U_{North}, U_{South}, U_{East}, U_{West})$

$= 0 + 0.6 * 0 = 0$

- State (1,2): $U_2(1,2)$ can move 4 directions, so:

Direction	
North:	$U = 0.8 \cdot U_1(1,2) + 0.1 \cdot U_1(1,2) + 0.1 \cdot U_1(2,2) = 0 + 0 + 0.1 * 2.4 = 0.24$
East:	$U = 0.8 \cdot U_1(2,2) + 0.1 \cdot U_1(1,2) + 0.1 \cdot U_1(1,1) = 0.8 * 2.4 + 0 + 0 = 1.92$
South:	$U = 0.8 \cdot U_1(1,1) + 0.1 \cdot U_1(1,2) + 0.1 \cdot U_1(2,2) = 0 + 0 + 0.1 * 2.4 = 0.24$
West:	$U = 0$

Then $U_2(1,2) = R(1,2) + \text{discount} * \max(U_{North}, U_{South}, U_{East}, U_{West})$

$= 0 + 0.6 * 1.92 = 1.152$

- State (2,1): $U_2(2,1)$ can move 4 directions, so:

Direction	
North:	$U = 0.8 \cdot U_1(2,2) + 0.1 \cdot U_1(1,1) + 0.1 \cdot U_1(3,1) = 0.8 * 2.4 + 0 - 5 * 0.1 = 1.42$
East:	$U = 0.8 \cdot U_1(3,1) + 0.1 \cdot U_1(2,1) + 0.1 \cdot U_1(2,2) = 0.8*(-5) + 0 + 0.1 * 2.4 = -3.76$
South:	$U = 0.8 \cdot U_1(2,1) + 0.1 \cdot U_1(1,1) + 0.1 \cdot U_1(3,1) = 0 + 0 - 5 * 0.1 = -0.5$
West:	$U = 0.8 \cdot U_1(1,1) + 0.1 \cdot U_1(2,1) + 0.1 \cdot U_1(2,2) = 0 + 0 + 0.1 * 2.4 = 0.24$

Then $U_2(2,1) = R(2,1) + \text{discount} * \max(U_{North}, U_{South}, U_{East}, U_{West})$

$= 0 + 0.6 * 1.42 = = 0.852$

- State (2,2): $U_2(2,2)$ can move 4 directions, so:

Direction	
North:	$U = 0.8 \cdot U_1(2,2) + 0.1 \cdot U_1(1,2) + 0.1 \cdot U_1(3,2) = 0.8 * 2.4 + 0 + 0.1*5 = 2.42$
East:	$U = 0.8 \cdot U_1(3,2) + 0.1 \cdot U_1(2,2) + 0.1 \cdot U_1(2,1) = 0.8*5 + 0.1 * 2.4 + 0 = 4.24$

South:	$U = 0.8 \cdot U_1(2,1) + 0.1 \cdot U_1(1,2) + 0.1 \cdot U_1(3,2) = 0 + 0 + 0.1*5 = 0.5$
West:	$U = 0.8 \cdot U_1(1,2) + 0.1 \cdot U_1(2,2) + 0.1 \cdot U_1(2,1) = 0.1 * 2.4 = 0.24$

Then $U_2(2,2) = R(2,2) + \text{discount} * \max(U_{North}, U_{South}, U_{East}, U_{West})$

$= 0 + 0.6 * 4.24 = 2.544$

Question 2: Reinforcement Learning

2.1 Model scenarios as reinforcement learning problems (states, actions, type of rewards etc.)

i. Learning to play chess

Q-learning Algorithm for Playing Chess:

$Q(s, a) \leftarrow Q(s, a) + \alpha[R(s) + \gamma \max a\,0\, Q(s\,0\,, a0\,) - Q(s, a)]$

- Q-Learning is like teaching a chess bot through trial and error. The bot learns by:

+ Seeing a board position (state).

+ Choosing a move (action).

+ Getting feedback (reward).

+ Improving its decision-making over time (updating the Q-Table).

+etc.

-States (S): Each state represents what's happening on the chessboard, how many chess pieces that both players have, or positions of every remaining pieces of both players. (Could be a grid of 8x8 array or strings)

-Action(A): The legal moves that the players can do according to the chess rules from the current state (move their chess pieces, get enemy chess pieces, etc)

-Reward (R):

+ Win the game = + 10000 (Terminal state reward)

+ Lost the game = – 10000 (Terminal state reward)

+ Draw = 0 (Terminal state)

+ Losing a chess piece = -10 (Nonterminal state reward)

+ Moving chess piece = -1 (Nonterminal state reward)

+ Get a chess piece = +100 (Nonterminal state reward)

+ etc.

- Q-table for storing the value of each action.

- Hyperparameters:

+Learning rate (α): How much we update the Q-value.

+Discount factor (γ): How much future rewards matter.

+Exploration rate (ε): Balances exploration vs. exploitation.

- Goal: Get as much reward as possible and avoid losing points to win the game.

Q-learning Algorithm:

```
# Initialize Q-table (empty dictionary)
Q = {}
# Parameters
alpha = 0.1      # How much we update the Q-value.
gamma = 0.9      # How much future rewards matter.
epsilon = 0.2    # Balances exploration vs. exploitation
# Loop for each episode (game)
for each game:
    state = starting position ()
    # Until the game end (terminal)
    while game is not over:
        # Choose an action using epsilon-greedy
        if random () < epsilon:
```

```
        action = random_legal_move(state)  # Explore
    else:
        action = best_move_from_Q(state)   # Exploit
    # Take action and observe next state and reward
    next_state, reward = make_move(state, action)
    # Update Q-value
    Q[state, action] = Q.get((state, action), 0) + alpha * (
        reward + gamma * max_Q_value(next_state) - Q.get((state, action), 0)
    ) # Q(s, a) ← Q(s, a) + α[R(s) + γ max a 0 Q(s 0 , a0 ) − Q(s, a)]
    # Move to the next state
    state = next state
```

---

ii. You are about to graduate, and you decide to plan your finances for the next 30 years (until retirement). Consider what a reinforcement model might look like for financial planning. What can be challenges for the problem?

- States (S): My heath status (age, sickness, strength, etc.)

- Action (A): Movement that can affect financial plan, including investment of any kinds, keeping relationship with the coworker, learning to expand knowledge, etc.

- Reward (R):

+ Depending on your age, the reward could be different (More money when young gives better reward than more money when old).

+ Doing investment earlier would give better rewards.

+ Keeping good relationships would give a long-term reward.

+ Terminal state: died, broke, retired, etc.

- Goal: Stay alive as long as possible to get as much reward as possible.

- Challenges in this problem could be:

+ Random event (uncertainty) in the environment, such as unpredictable factors, or being tricked by someone, …

+ Poor health status (cancer, old, backache, etc.)

+ Complex state space and long period of time.

2.2 Consider chess. If we wanted to approximate the utility of the states using a sum of weighted features, discuss the type of features that might be useful for playing chess.

In chess, utility represents how "useful" a given state is—essentially evaluating whether a position could lead to a winning or losing strategy. To assess this, feature as follows could be useful to evaluate the value of action when playing chess:

+ The number of pieces the agent has on the board.

+ The number of pieces the opponent has.

+ The total value of pieces (e.g., Queen = 9, Rook = 5) for both the agent and the opponent.

+ The mobility of the King (i.e., how many squares the King can move to; limited mobility could indicate vulnerability, such as being stuck in a check).

+ The number of moves needed for a pawn to reach promotion.

+ How close or possibility that enemy chess piece can reach your king.

+ Etc.

- Utility Formula: $U(s) = \Sigma i\ (w_i \times f_i(s))$

Where:

$f_i(s)$: The i^{th} feature for a given state s.

w_i: The weight assigned to feature i, which can be manually adjust or learned over tests.

Question 3: Bayesian Network

3.1 Write down the joint probability table specified by the Bayesian network.

In which F = Bad weather and C = Traffic Congestion

Bad Weather (F):

P(F)	0.1	(Given)
P(¬F)	0.9	=> P (¬F) = 1 – P(F) = 1 – 0.1 = 0.9

Traffic Congestion (C):

P(C | F)	0.8
P(C |¬F)	0.3

=>

P(¬C | F)	0.2	=> P(¬C|F) = 1 − P(C|F) = 0.2
P(¬C |¬F)	0.7	=> P(¬C|¬F) = 1 − P(C|¬F) = 0.7

=> Joint Probability:

P (C, F) = P(F) · P(C|F) = 0.1 * 0.8 = 0.08

P (¬C, F) = P(F) · P(¬C|F) = 0.2 * 0.1 = 0.02

P (C, ¬F) = P(¬F) · P(C|¬F) = 0.9 * 0.3 = 0.27

P (¬C, ¬F) = P(¬F) · P(¬C|¬F) = 0.9 * 0.7 = 0.63

Event	C	¬C
F	0.08	0.02

¬F	0.27	0.63

3.2 Are C and F independent in the given Bayesian network.

No, C and F are not independent as both variables do affecting each other.

For C and F to be independent (C ⊥ F):

=> P(F|C) = P (F) but:

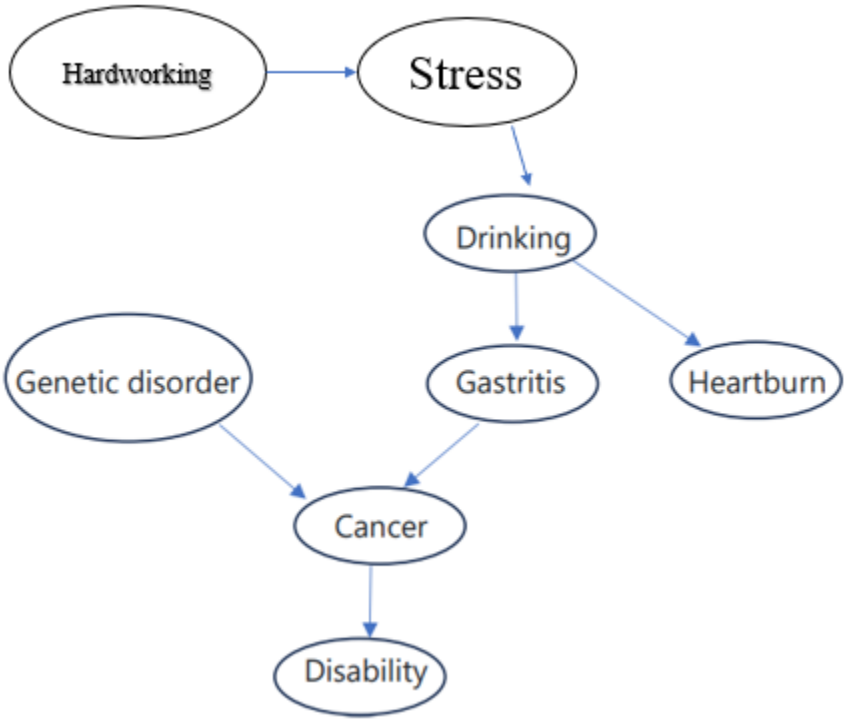
+ P (C, F) = 0.08

+ P(F)·P(C) = 0.1·0.35 = 0.035

Since P (C, F) not equal to P(F)·P(C), therefore, C and F are not independent.

Question 4: Bayesian Network

4.1. Extend the network with the Boolean variables Hardworking and Stress. State your assumptions and reason how components might affect each other.



- Added variables:

+ Hardworking: states whether an individual is hardworking or not (True or False). Hardworking affects stress

+ Stress: represents whether an individual is stressed or not (True or False). Stress affects gastritis

-Assumptions and Reasonings:

+ Hardworking affects stress, when you're working hard for something over a long period of time, you will be tired and therefore it's also lead to stress.

+ Stress affects gastritis, when you are stressed, you usually want to go to drink something (maybe alcohol?) to relieve the stress you are having, which will then increase the probability of having gastritis, which will indirectly affect the health status.

4.2. Give reasonable conditional probability tables for all the nodes.

Hardworking:

Event	P(Hardworking)
Hardworking	0.65
¬ Hardworking	0.35

Genetic Disorder:

Event	P(Genetic Disorder)
Genetic Disorder	0.2
¬ Genetic Disorder	0.8

Stress | Hardworking:

Event	Hardworking	¬Hardworking
Stress	0.7	0.4
¬ Stress	0.3	0.6

Drinking | Stress:

Event	Stress	¬Stress
Drinking	0.6	0.3
¬ Drinking	0.4	0.7

Gastritis | Drinking:

Event	Drinking	¬Drinking
Gastritis	0.5	0.2
¬ Gastritis	0.5	0.8

Heartburn | Drinking:

Event	Drinking	¬Drinking
Heartburn	0.4	0.1
¬ Heartburn	0.6	0.9

Cancer | Gastritis, Genetic Disorder:

Event	Gastritis	¬ Gastritis	Genetic Disorder	¬ Genetic Disorder
Cancer	0.6	0.4	0.3	0.1
¬ Cancer	0.4	0.6	0.7	0.9

Disability | Cancer:

Event	Cancer	¬Cancer
Disability	0.7	0.2
¬ Disability	0.3	0.8

4.3. Assuming that no conditional independence relations are known to hold among the Boolean nodes, how many independent values are contained in the joint probability distribution?

Formula: Number of independent probabilities = $2^n - 1$

Where n is the total number of Boolean variables (nodes) in the network.

There are 8 Boolean nodes (Hardworking, Genetic Disorder, Stress, Drinking, Gastritis, Heartburn, Cancer, Disability)

=> $2^n - 1 = 2^8 - 1 = 256 - 1 = 255$ (Subtract 1 for normalization)

So, there's 255 independent values.