Assignment 1: Solutions
IS711: Learning and Planning in Intelligent Systems

SHI Jieke
jkshi.2022@phdcs.smu.edu.sg

1 Question #1

1.1 Answer to Question #1.1

We can formulate the problem with the following definitions: The state space is the set of all possible configurations.

- **State representation**: A state can be represented by one or several piles. Each pile is represented by a stack of blocks denoted as color+label. For example, the Configuration 1 of this question is in the state of {[*Green A, Grey B, Orange A*], [*Yellow B, Green C*]}. There are five blocks available. The first block of each pile is the top and the last block is on the table.
- **State space**: The state space is the set of all possible configurations of piles.
- **Initial state**: {[*Green A, Grey B, Orange A*], [*Yellow B, Green C*]}.
- **Goal state**: {[*Green C, Grey B, Yellow B, Orange A, Green A*]}.
- **Possible Operators**: denoting x and y as two blocks at the top of two piles, the possible operators are:
  - *Move(x, y)*: move the block x to atop y.
  - *MoveToTable(x)*: move the block x to the table.
- **Path cost**: The cost of each call to operator *Move(x, y)* is 1. The cost of each call to operator *MoveToTable(x)* is determined by color, namely, Orange block costs 2, Gray block costs 3, Yellow block costs 1, Green block costs 4. The path cost is the sum of the costs of all calls to operators.

1.2 Answer to Question #1.2

The heuristic function is $h(s) = \#blocks\_not\_in\_goal\_place(s)$, i.e., the number of blocks that do not arrive goal place in the state s. The heuristic function is admissible. When a block is not at the goal place, it costs at least 1 to call the operator *Move* or more to call the *MoveToTable*. Thus, for each state, $h(s)$ is always less than or equal to the actual cost to reach the goal state. The expansion of first 4 nodes of the search tree are shown as Figure 1.

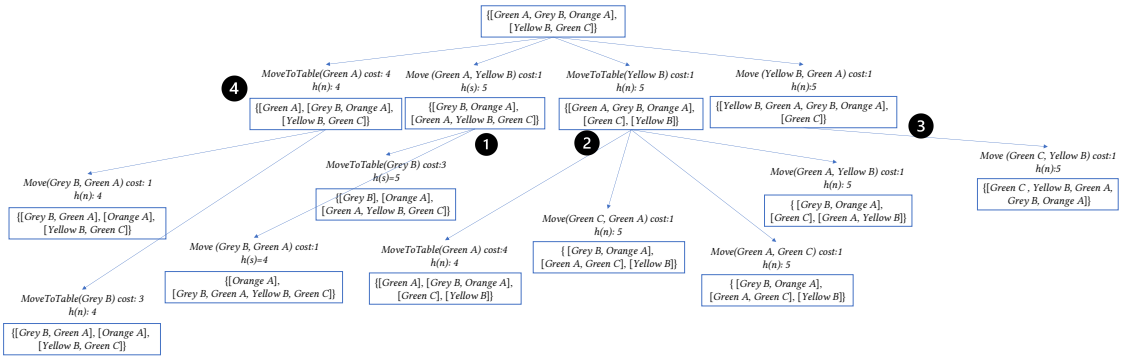

Figure 1: *The expansion of first 4 nodes.*

2 Question #2

The trip with the least possible cost would need 6 large buses and 5 small buses. The calculation using linear programming is as follows:

- **Variables**: x_1: the number of large buses; x_2: the number of small buses.
- **Objective**: Minimize rental costs, i.e.,

$$\min 800x_1 + 600x_2 \tag{1}$$

- **Constraints**: Subject to the following constraints:

$$\begin{aligned} 50x_1 + 40x_2 &\geq 500 \\ x_1 + x_2 &\leq 11 \\ 0 \leq x_1 &\leq 10 \\ 0 \leq x_2 &\leq 8 \end{aligned} \tag{2}$$

- **Solution**: As shown in Figure 2, the optimal solution is $x_1 = 6$ and $x_2 = 5$. The minimum cost is $800 \times 6 + 600 \times 5 = 7800$.

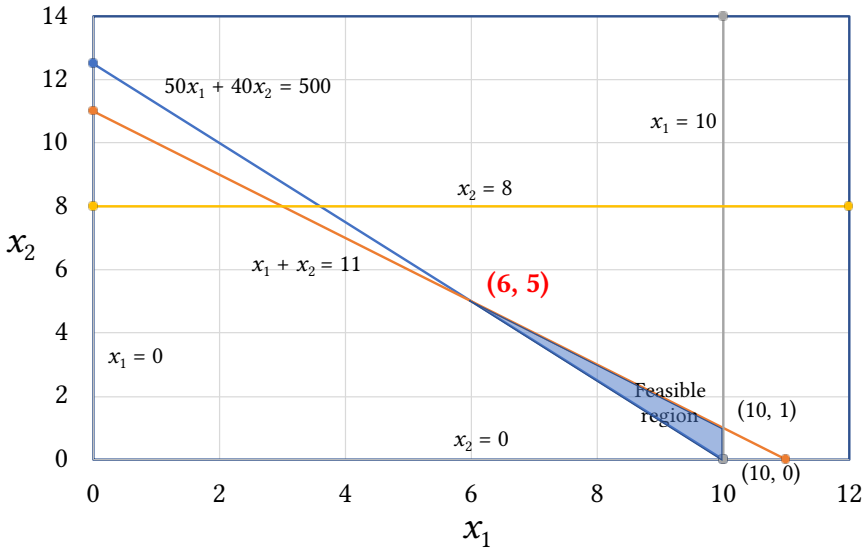

Figure 2: *Solution to Question #2*

The drawbacks of linear programming are:

- To the above question, we can observe redundant computation in linear programming, i.e., if we remove the constraint $x_2 \leq 8$, the answer will be unchanged. The other constraints are already sufficient conditions for the answer, but linear programming still needs to compute the redundant constraint $x_2 \leq 8$.
- In the general cases of using linear programming,
  - When facing real-life problems, it is possible that the constraints may not be directly expressible as linear inequalities. Non-linear relations may be involved, e.g., $x_1^2 + x_2^2 \leq 100$;
  - When solving linear programming, there is no guarantee of getting integer values. A non-integer valued solution will be meaningless to lots of problems.

– Linear programming can only handle single-objective problems, while real-life decision-making problems often have multiple objectives.
– The solution of linear programming can involve massive mathematical calculations. In some cases, the solution may not be available due to the complexity of the problem.

3 Question #3

I will play the game as the winning probability is 0.517 > 0.5, and the expected value is 0.279 > 0. The calculation is as follows:

For each round of rolling the dice, there exist 6^3 possible outcomes. Among them,

- There is 1 outcome whose total is 3.
- There are 10 outcomes whose total is 6.
- There are 21 outcomes whose total is 8.
- There are 25 outcomes whose total is 9.
- There are 27 outcomes whose total is 11.
- There are 25 outcomes whose total is 12.
- There are 10 outcomes whose total is 15.
- There are 1 outcomes whose total is 18.

Therefore, the probability of winning \$1 in the first round is

$$P(\text{winning \$1 in first round}) = \frac{1 + 21 + 27 + 10}{6^3} = \frac{59}{216} \approx 0.273 \tag{3}$$

The probability of losing \$1 in the first round is

$$P(\text{losing \$1 in first round}) = \frac{25 + 1}{6^3} = \frac{26}{216} \approx 0.120 \tag{4}$$

The probability of winning \$2 in the second round is

$$P(\text{winning \$2 in second round}) = (1 - \frac{59}{216} - \frac{26}{216}) \times \frac{10 + 25 + 27 + 25}{6^3} = \frac{11397}{46656} \approx 0.244 \tag{5}$$

The probability of losing \$1 in the second round is

$$P(\text{losing \$1 in the second round}) = 1 - \frac{59}{216} - \frac{26}{216} - \frac{11397}{46656} = \frac{16899}{46656} \approx 0.362 \tag{6}$$

Thus, the winning probability of this game is

$$P(\text{winning}) = 0.273 + 0.244 = 0.517 > 0.5 \tag{7}$$

The expected value of the game is

$$E = \frac{59}{216} \times 1 + \frac{26}{216} \times (-1) + \frac{11397}{46656} \times 2 + \frac{16899}{46656} \times (-1) = \frac{13023}{46656} \approx 0.279 > 0 \tag{8}$$

The winning probability is 0.517 > 0.5, and the expected value is 0.279 > 0, thus, I will play the game.

4 Question #4

4.1 Answer to Question #4.1

Figure 3 shows the Bayes Network to identify fraudulent transactions.

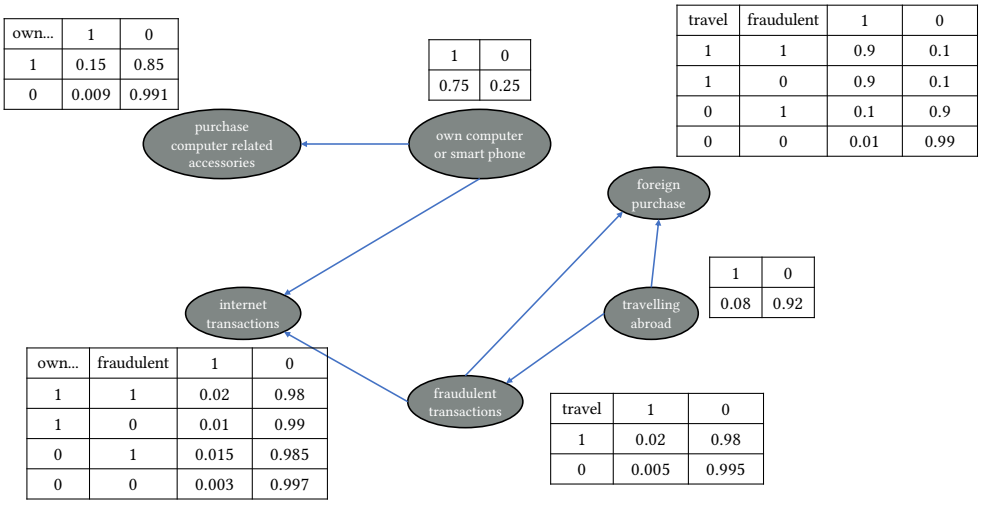

Figure 3: *Bayes Network to Question #4*

4.2 Answer to Question #4.2

The prior probability that the current transaction is legitimate is 0.9938. The calculation is as follows:

Abbreviations: $fraudulent \longrightarrow fr, travel \longrightarrow tr$

$$P(fr = 0) = \sum_{tr} P(fr = 0|tr) = 0.08 \times 0.98 + 0.92 \times 0.995 = 0.9938 \tag{9}$$

The probability that the current transaction is a legitimate one is 0.98726326. The calculation is as follows:

Abbreviations: $fraudulent \longrightarrow fr, travel \longrightarrow tr, foreign\_purchase \longrightarrow fp, internet\_purchase \longrightarrow ip,$ $purchase\_computer \longrightarrow pc, own\_computer \longrightarrow oc$

$$\begin{aligned}
P(fr = 0|fp = 0, ip = 1, pc = 1) &= \frac{P(fr = 0, fp = 0, ip = 1, pc = 1)}{P(fp = 0, ip = 1, pc = 1)} \\
&= \frac{\sum_{oc}\sum_{tr} P(oc)P(tr)P(fr = 0|tr)P(fp = 0|fr = 0, tr)P(ip = 1|fr = 0, oc)P(pc = 1|oc)}{\sum_{oc}\sum_{tr}\sum_{fr} P(oc)P(tr)P(fr|tr)P(fp = 0|fr, tr)P(ip = 1|fr, oc)P(pc = 1|oc)} \\
&= \frac{\sum_{oc} P(oc)P(ip = 1|fr = 0, oc)P(pc = 1|oc)\sum_{tr} P(tr)P(fr = 0|tr)P(fp = 0|fr = 0, tr)}{\sum_{oc}\sum_{tr} P(oc)P(tr)P(fr = 0|tr)P(fp = 0|fr = 0, tr)P(ip = 1|fr = 0, oc)P(pc = 1|oc)} \\
&\quad + \sum_{oc}\sum_{tr} P(oc)P(tr)P(fr = 1|tr)P(fp = 0|fr = 1, tr)P(ip = 1|fr = 1, oc)P(pc = 1|oc) \\
&= \frac{(0.01 \times 0.75 + 0.003 \times 0.25) \times (0.15 \times 0.75 + 0.009 \times 0.25) \times (0.08 \times 0.98 + 0.92 \times 0.995) \times (0.1 \times 0.08 + 0.99 \times 0.92)}{(0.08 \times 0.98 + 0.92 \times 0.995) \times (0.1 \times 0.08 + 0.99 \times 0.92) \times (0.01 \times 0.75 + 0.003 \times 0.25) \times (0.15 \times 0.75 + 0.009 \times 0.25)} \\
&\quad + (0.02 \times 0.08 + 0.92 \times 0.005) \times (0.1 \times 0.08 + 0.9 \times 0.92) \times (0.02 \times 0.75 + 0.015 \times 0.25) \times (0.15 \times 0.75 + 0.009 \times 0.25) \\
&= \frac{0.0008644236129}{0.0008755755916} = 0.98726326
\end{aligned} \tag{10}$$

4.3 Answer to Question #4.3

As her spouse confirms that she is currently traveling, i.e, $tr = 1$, the probability that the current transaction is legitimate would reduce into 0.9556737589, i.e., the probability of a fraud increases a bit. The calculation is as follows:

$$
\begin{aligned}
P(fr=0|fp=0,ip=1,pc=1,tr=1) &= \frac{P(fr=0,fp=0,ip=1,pc=1,tr=1)}{P(fp=0,ip=1,pc=1,tr=1)} \\
&= \frac{\sum_{oc} P(oc)P(fr=0|tr=1)P(fp=0|fr=0,tr=1)P(ip=1|fr=0,oc)P(pc=1|oc)}{\sum_{oc}\sum_{fr} P(oc)P(fr|tr=1)P(fp=0|fr,tr=1)P(ip=1|fr,oc)P(pc=1|oc)} \\
&= \frac{0.98\times 0.1\times(0.01\times 0.75+0.003\times 0.25)\times(0.15\times 0.75+0.009\times 0.25)}{\begin{array}{l}0.98\times 0.1\times(0.01\times 0.75+0.003\times 0.25)\times(0.15\times 0.75+0.009\times 0.25)\\ \quad +0.02\times 0.1\times(0.02\times 0.75+0.015\times 0.25)\times(0.15\times 0.75+0.009\times 0.25)\end{array}} \\
&= \frac{0.000092775375}{0.0000970785} = 0.9556737589
\end{aligned}
\tag{11}
$$

5 Question #5

The Most Probable Explanation for $G=g1$ and $L=l0$ is $I=i1$, $D=d0$, and $S=s1$. The probability is 0.01296. The calculation is as follows:

The joint distribution of the given Bayesian network is $P(I,D,G,S,L)=P(I)P(D)P(G|I,D)P(S|I)P(L|G)$.

- **Forward pass:**

$$
\begin{aligned}
&\max_{D,I,S} P(I,D,G=g1,S,L=l0) \\
&= \max_{D,I,S} P(I)P(D)P(G=g1|I,D)P(S|I)P(L=l0|G=g1) \\
&= \max_{I} P(I) \max_{D} P(D)P(G=g1|I,D) \max_{S} P(S|I) * 0.1 \\
&= \max_{I} \begin{bmatrix} i0 & i1 \\ 0.7 & 0.3 \end{bmatrix} \begin{bmatrix} i0 & i1 \\ 0.6*0.3 & 0.6*0.9 \end{bmatrix} \begin{bmatrix} i0 & i1 \\ 0.95 & 0.8 \end{bmatrix} * 0.1 \\
&= \max_{I} \begin{bmatrix} i0 & i1 \\ 0.01197 & 0.01296 \end{bmatrix} \\
&= 0.01296
\end{aligned}
\tag{12}
$$

- **Backward pass:** $MPE = \arg\max_{D,I,S} P(I,D,G=g1,S,L=l0) = (I=i1, D=d0, S=s1)$, i.e., the most probable explanation for $G=g1$ and $L=l0$ is $I=i1$, $D=d0$, and $S=s1$.

6 Question #6

6.1 Answer to Question #6.a

The MDP model is defined as follows:

- **States**: $S=\{L1, L2, L3, L4\}$.
- **Actions**: $A=\{Pickup, MoveToL1, MoveToL2, MoveToL3, MoveToL4\}$
- **transition probability matrices**: Shown in the 4^{th} column of Table 1.
- **Reward function**: Shown in the 5^{th} column of Table 1.

6.2 Answer to Question #6.b

The code of value iteration algorithm is shown in Listing 1. The code is also available via this link: https://github.com/jiekeshi/IS711/blob/main/Value_iteration.py.

Also, Figure 4 shows the first two steps of value iteration in the solution.

Table 1: *MDP model in Question #6*

State	Action	State′	$P(s'\|s,a)$	$Reward(s,a,s')$
L1	Pickup	L1	0.7	0
L1	Pickup	L2	0.15	19
L1	Pickup	L3	0.09	8.5
L1	Pickup	L4	0.06	7.75
L1	MoveToL2	L2	1	-1
L1	MoveToL3	L3	1	-1.5
L1	MoveToL4	L4	1	-1.25
L2	Pickup	L2	0.5	0
L2	Pickup	L1	0.2	9
L2	Pickup	L3	0.3	6.25
L2	MoveToL1	L1	1	-1
L2	MoveToL3	L3	1	-1.75
L2	MoveToL4	L4	1	-1
L3	Pickup	L3	0.4	0
L3	Pickup	L1	0.18	13.5
L3	Pickup	L4	0.42	7.8
L3	MoveToL1	L1	1	-1.5
L3	MoveToL2	L2	1	-1
L3	MoveToL4	L4	1	-1.2
L4	Pickup	L4	0.2	0
L4	Pickup	L1	0.32	9.75
L4	Pickup	L2	0.48	4
L4	MoveToL1	L1	1	-1.25
L4	MoveToL2	L2	1	-1
L4	MoveToL3	L3	1	-1

Listing 1: *Core Code of Value Iteration*

```
from time import time

convergence_tol = 0.01
discount_factor = 0.95

### MDP classes
class State:
  def __init__(self, pos):
    self.pos = pos
    self.value = 0
    self.value_old = 0
    self.is_converged = True
    self.policy = "--"

class Transition:
  def __init__(self, start, end, transition_prob, reward):
    self.start = start
    self.end = end
    self.transition_prob = transition_prob
```

```
    self.reward = reward

class Action:
  def __init__(self, transitions):
    self.transitions = transitions

def main():
  t0 = time()

  ### Create states
  states = {}
  places = ["L1", "L2", "L3", "L4"]
  for i in places:
    states[i] = State(i)

  ### Create actions
  actions = {}
  actions["Pickup"] = Action(transitions=[
    Transition("L1", "L1", 0.7, 0),
    Transition("L1", "L2", 0.15, 19),
    Transition("L1", "L3", 0.09, 8.5),
    Transition("L1", "L4", 0.06, 7.75),
    Transition("L2", "L2", 0.5, 0),
    Transition("L2", "L1", 0.2, 9),
    Transition("L2", "L3", 0.3, 6.25),
    Transition("L3", "L3", 0.4, 0),
    Transition("L3", "L1", 0.18, 13.5),
    Transition("L3", "L4", 0.42, 7.8),
    Transition("L4", "L4", 0.2, 0),
    Transition("L4", "L1", 0.32, 9.75),
    Transition("L4", "L2", 0.48, 4),
  ])
  actions["MoveToL1"] = Action(transitions=[
    Transition("L2", "L1", 1, -1),
    Transition("L3", "L1", 1, -1.5),
    Transition("L4", "L1", 1, -1.25)
  ])
  actions["MoveToL2"] = Action(transitions=[
    Transition("L1", "L2", 1, -1),
    Transition("L3", "L2", 1, -1),
    Transition("L4", "L2", 1, -1)
  ])
  actions["MoveToL3"] = Action(transitions=[
    Transition("L1", "L3", 1, -1.5),
    Transition("L2", "L3", 1, -1.75),
    Transition("L4", "L3", 1, -1)
  ])
  actions["MoveToL4"] = Action(transitions=[
    Transition("L1", "L4", 1, -1.25),
    Transition("L3", "L4", 1, -1.2),
    Transition("L2", "L4", 1, -1)
  ])

  ### Value Iteration
  # Print initial values
  print(states["L1"].value)
  print(states["L2"].value)
  print(states["L3"].value)
  print(states["L4"].value)
  print("Initial state values" + "\n")

  i = 0
  while True:
    # Value iteration (estimate state values)
```

```
    for state in states.values():
      expected_utils = []
      for action_name, action in actions.items():
        expected_util = 0
        for t in action.transitions:
          if t.start == state.pos:
            state_new = states[t.end]
            expected_util += t.transition_prob * (t.reward + (state_new.value_old * discount_factor))
        expected_utils.append((expected_util, action_name))
      best_action = max(expected_utils, key=lambda x: x[0])   # Action with max value / expected utility
      state.value, state.policy = best_action

    # Update state values
    for state in states.values():
      state.is_converged = abs(state.value_old - state.value) < convergence_tol
      state.value_old = state.value
    # Print
    i += 1
    print("state values L1", states["L1"].value, "action L1", states["L1"].policy)
    print("state values L2", states["L2"].value, "action L2", states["L1"].policy)
    print("state values L3", states["L3"].value, "action L4", states["L1"].policy)
    print("state values L4", states["L4"].value, "action L4", states["L1"].policy)
    print("Iteration: {0}".format(i) + "\n")
    # Convergence check
    is_converged = all([state.is_converged for state in states.values()])
    if is_converged: break

  print("Best policy" + "\n")
  print("Time taken is {:.4f} seconds".format(time() - t0))

if __name__ == "__main__":
  main()
```

```
jieke@Jiekes-iMac IS711 % /opt/homebrew/bin/python3 /Users/jieke/Documents/GitHub/IS711/Value_iteration.py
0
0
0
0
Initial state values

state values L1 4.08 action L1 Pickup
state values L2 3.675 action L2 Pickup
state values L3 5.7059999999999995 action L4 Pickup
state values L4 5.04 action L4 Pickup
Iteration: 1

state values L1 8.0920305 action L1 Pickup
state values L2 7.822035 action L2 Pickup
state values L3 10.58292 action L4 Pickup
state values L4 8.91372 action L4 Pickup
Iteration: 2
```

Figure 4: *Two steps of value iteration*