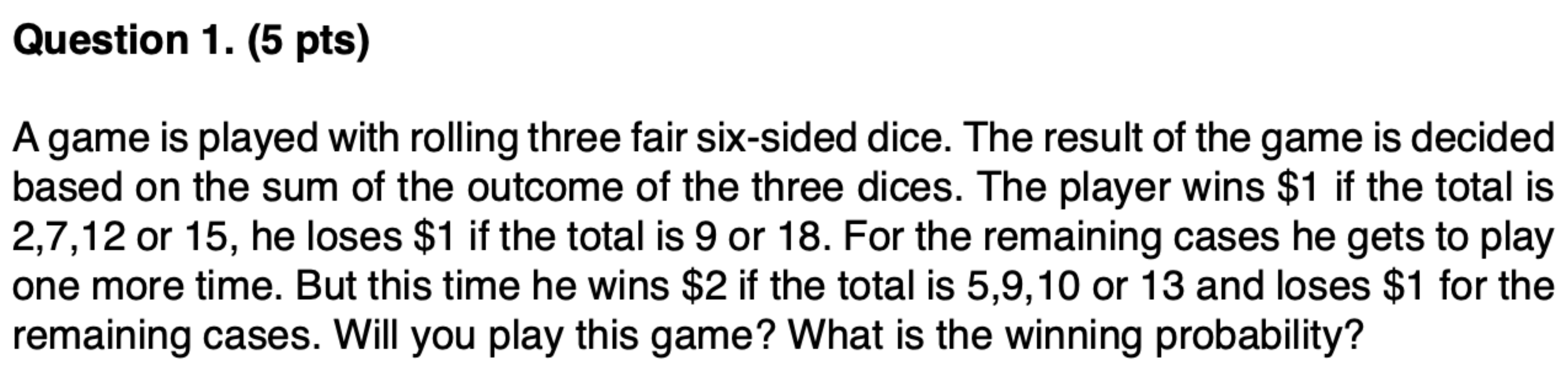
Assignment 1

Tags Assignment 1 - CS422.pdf

Google Colaboratory

https://colab.research.google.com/drive/1qa0tqNvWRwwEwHhcoSxLcAs-JiwY8d08?authuser=0#scrollTo=foqJlHytYQck



Question 1. (5 pts)

A game is played with rolling three fair six-sided dice. The result of the game is decided based on the sum of the outcome of the three dices. The player wins $1 if the total is 2,7,12 or 15, he loses $1 if the total is 9 or 18. For the remaining cases he gets to play one more time. But this time he wins $2 if the total is 5,9,10 or 13 and loses $1 for the remaining cases. Will you play this game? What is the winning probability?

total number of outcomes from rolling 3 fair six-sided dice = 6 x 6 x 6 = 216

total number of outcomes for sum 2 = 0

to get sum = 7:

- (1, 1, 5)
- (1, 2, 4)
- (1, 3, 3)
- (2, 2, 3)

total number of outcomes for sum 7 = 3 + 3! + 3 + 3 = 15

to get sum = 12:

- (1, 6, 5)
- (2, 5, 5)
- (2, 6, 4)
- (3, 6, 3)
- (3, 5, 4)
- (4, 4, 4)

total number of outcomes for sum 12 = 3! + 3 + 3! + 3 + 3! + 1 = 25

to get sum = 15:

- (3, 6, 6)
- (4, 5, 6)
- (5, 5, 5)

total number of outcomes for sum 15 = 3 + 3! + 1 = 10

total probability to win $1 = (15 + 25 + 10) / 216 = 0.23148

to get sum = 9:

- (1, 6, 2)
- (1, 5, 3)
- (1, 4, 4)
- (2, 3, 4)
- (2, 5, 2)
- (3, 3, 3)

total number of outcomes for sum 9 = 3! + 3! + 3 + 3! + 3 + 1 = 25

to get sum = 18:

- (6, 6, 6)

total number of outcomes for sum 18 = 1

total probability to lose $1 = (25 + 1)/216 = 0.12037

total probability to continue to another round = 1 - 0.23148 - 0.12037 = 0.64815

to get sum = 5:

- (1, 1, 3)
- (1, 2, 2)

total number of outcomes for sum 5 = 3 + 3 = 6

total number of outcomes for sum 9 = 3! + 3! + 3 + 3! + 3 + 1 = 25

to get sum = 10:

- (1, 4, 5)
- (1, 6, 3)
- (2, 4, 4)
- (2, 5, 3)
- (2, 6, 2)
- (3, 3, 4)

total number of outcomes for sum 10 = 3! + 3! + 3 + 3! + 3 + 3 = 27

to get sum = 13:

- (1, 6, 6)
- (2, 5, 6)
- (3, 4, 6)
- (3, 5, 5)
- (4, 4, 5)

total number of outcomes for sum 13 = 3 + 3! + 3! + 3 + 3 = 21

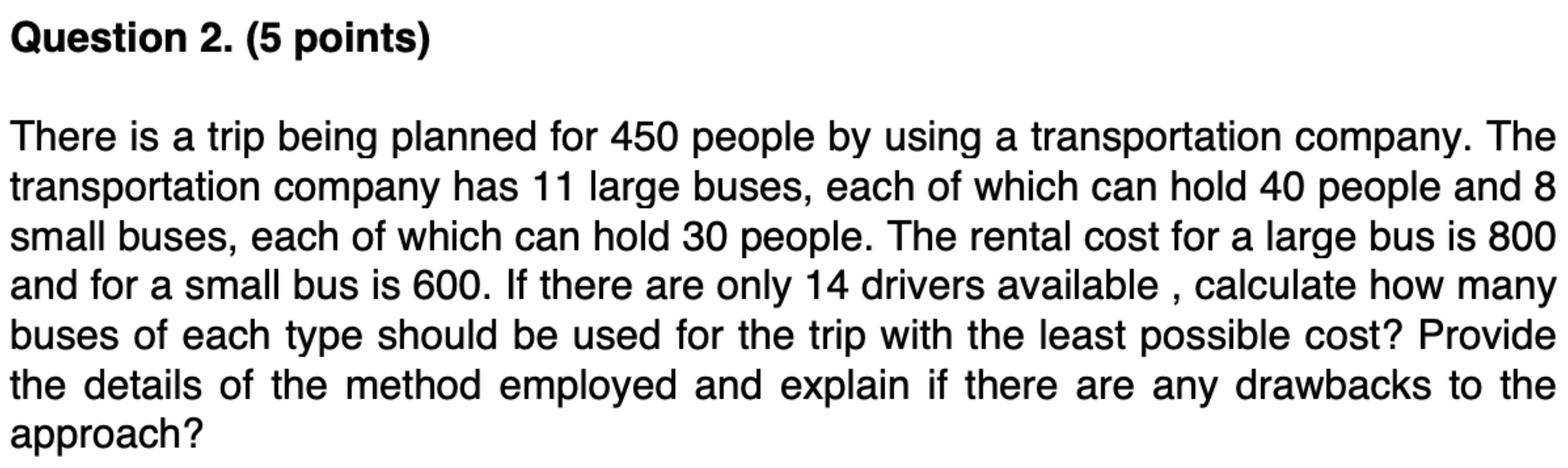
total probability to win $2 = 0.64815 x (6 + 25 + 27 + 21)/216 = 0.23705

total probability to lose $2 = 0.64815 x (1 - (6 + 25 + 27 + 21)/216) = 0.41110

expected return = 0.23148 x 1 + 0.12037 x (-1) + 0.23705 x 2 + 0.41110 x (-1) = 0.17411

since the expected return is positive, i will play this game

winning probability = 0.23148 + 0.23705 = 0.46853



Question 2. (5 points)

There is a trip being planned for 450 people by using a transportation company. The transportation company has 11 large buses, each of which can hold 40 people and 8 small buses, each of which can hold 30 people. The rental cost for a large bus is 800 and for a small bus is 600. If there are only 14 drivers available , calculate how many buses of each type should be used for the trip with the least possible cost? Provide the details of the method employed and explain if there are any drawbacks to the approach?

linear programming can be used to calculate the number of buses of each type needed

- variables: number of small and large buses, S and L
- aim: to minimise rental cost = 600S + 800L
- constraints:
  - S + L ≤ 14
  - L ≤ 11
  - S ≤ 8
  - 30S + 40L ≤ 450

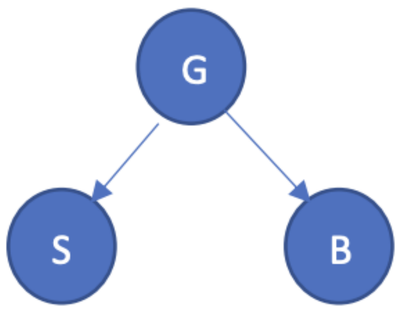
Question 3. (5 pts)

In this problem, you will be tasked with learning the local probability tables of a belief network. There are three variables Genre, G , rating of Sonia, S and rating of Bob, B.

G ∈ {drama - d, comedy - c, thriller - t}

S ∈ {1, 2, 3, 4, 5}

B ∈ {1, 2, 3, 4, 5}



Please find the local probability tables (showing only non-zero values) of the above Bayesian Network given the following 10 data points:

1. d, 5, 4 (indicating drama movie, where rating of Sonia and Bob were 5 and 4 respectively)
2. c, 3, 5

3. t, 5, 3
4. d, 4, 4
5. t, 5, 4
6. d, 5, 5
7. c, 3, 4
8. c, 4, 5
9. t, 5, 2
10. t, 4, 3

#(G = d) = 3

#(G = c) = 3

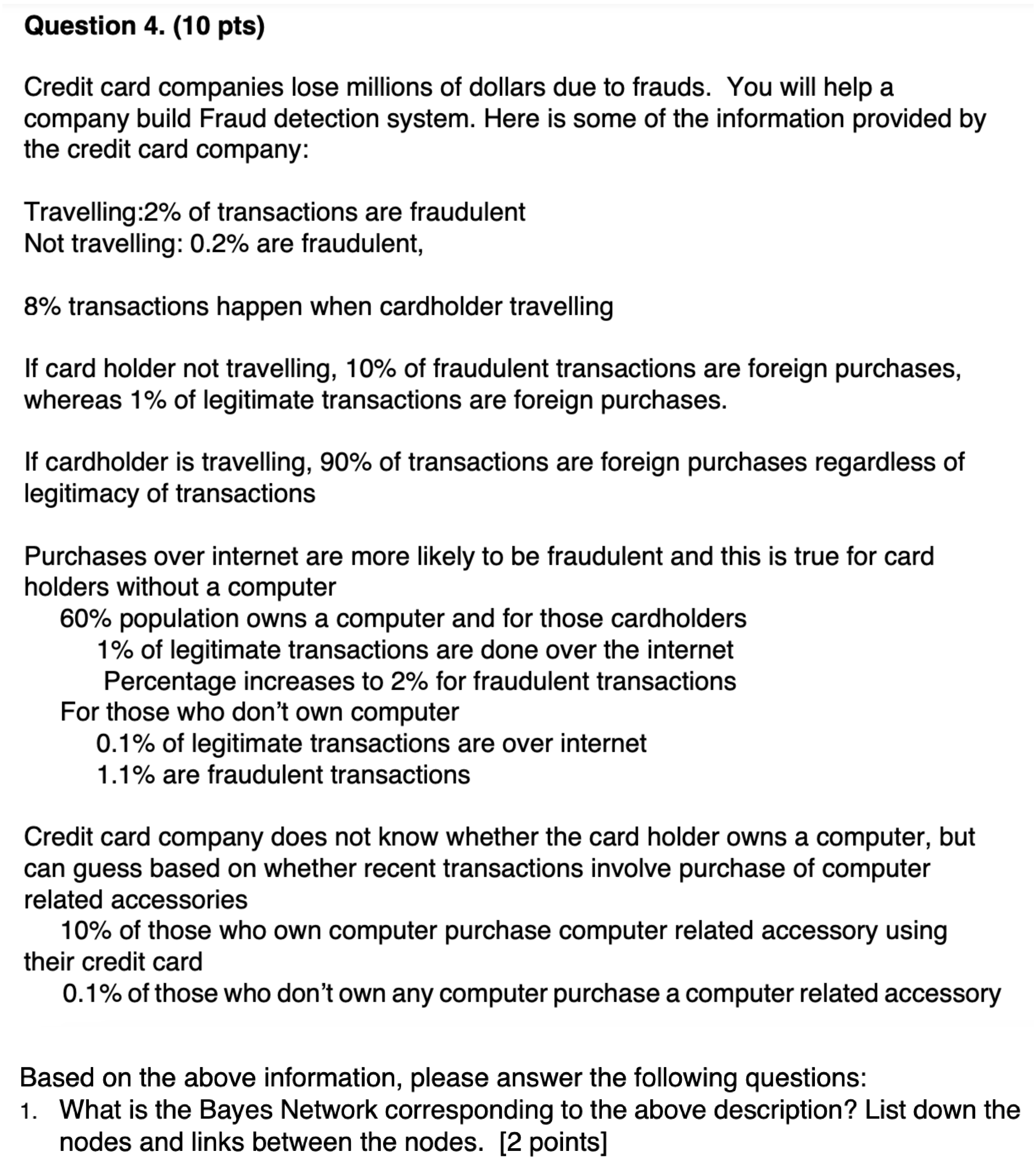
#(G = t) = 4

P(G = d)	P(G = c)	P(G = t)
0.3	0.3	0.4

$$P(S=1|G=d) = \frac{\#(S=1, G=d)}{\#(G=d)} = \frac{0}{3} = 0$$
$$P(S=2|G=d) = \frac{\#(S=2, G=d)}{\#(G=d)} = \frac{0}{3} = 0$$
$$P(S=3|G=d) = \frac{\#(S=3, G=d)}{\#(G=d)} = \frac{0}{3} = 0$$
$$P(S=4|G=d) = \frac{\#(S=4, G=d)}{\#(G=d)} = \frac{1}{3}$$
$$P(S=5|G=d) = \frac{\#(S=5, G=d)}{\#(G=d)} = \frac{2}{3}$$

G	P(S = 1)	P(S = 2)	P(S = 3)	P(S = 4)	P(S = 5)
d	0	0	0	1/3	2/3
c	0	0	2/3	1/3	0
t	0	0	0	1/4	3/4

G	P(B = 1)	P(B = 2)	P(B = 3)	P(B = 4)	P(B = 5)
d	0	0	0	2/3	1/3
c	0	0	0	1/3	2/3
t	0	1/4	1/2	1/4	0



Question 4. (10 pts)

Credit card companies lose millions of dollars due to frauds. You will help a company build Fraud detection system. Here is some of the information provided by the credit card company:

Travelling:2% of transactions are fraudulent
Not travelling: 0.2% are fraudulent,

8% transactions happen when cardholder travelling

If card holder not travelling, 10% of fraudulent transactions are foreign purchases, whereas 1% of legitimate transactions are foreign purchases.

If cardholder is travelling, 90% of transactions are foreign purchases regardless of legitimacy of transactions

Purchases over internet are more likely to be fraudulent and this is true for card holders without a computer
  60% population owns a computer and for those cardholders
    1% of legitimate transactions are done over the internet
    Percentage increases to 2% for fraudulent transactions
  For those who don't own computer
    0.1% of legitimate transactions are over internet
    1.1% are fraudulent transactions

Credit card company does not know whether the card holder owns a computer, but can guess based on whether recent transactions involve purchase of computer related accessories
  10% of those who own computer purchase computer related accessory using their credit card
  0.1% of those who don't own any computer purchase a computer related accessory

Based on the above information, please answer the following questions:

1. What is the Bayes Network corresponding to the above description? List down the nodes and links between the nodes. [2 points]

```
graph TD;
  Travelling-->Fraudulent;
  Travelling-->ForeignPurchase;
  Fraudulent-->ForeignPurchase;
  OwnsComputer-->InternetTransaction;
  Fraudulent-->InternetTransaction;
  OwnsComputer-->PurchaseAccessory;
```

Let T be the event where the cardholder is travelling

Let F be the event that the transaction is fraudulent

Let R be the event that the transaction is a foreign purchase

Let C be the event that the cardholder owns a computer

Let I be the event where the transaction is made over the internet

Let A be the event where the transaction involves purchasing computer related accessories

P(T) = 0.08

P(C) = 0.6

T	P(F)
0	0.002

1	0.02

T	F	P(R)
0	0	0.01
0	1	0.1
1	0	0.9
1	1	0.9

C	F	P(I)
0	0	0.001
0	1	0.011
1	0	0.01
1	1	0.02

C	P(A)
0	0.01
1	0.1

2. What is the prior probability (i.e., before we search for previous computer related purchases and before we verify whether it is a foreign and/or an Internet purchase) that the current transaction is a fraud? What is the probability that the current transaction is a fraud once we have verified that it is not a foreign transaction, but an Internet purchase and that the cardholder did not purchase computer related accessories in the past week? Use likelihood weighting approach with 100,000 samples. Please provide 10 samples generated. How different is the answer compared to if variable elimination is used?[4 points]

```
!pip install pgmpy
```

```
from pgmpy.factors.discrete import State
from pgmpy.models import BayesianNetwork
from pgmpy.factors.discrete import TabularCPD
from pgmpy.sampling import BayesianModelSampling
```

defining the bayesian network

```
# Define the Bayesian Network
bn = BayesianNetwork([('T', 'F'), ('F', 'R'), ('T', 'R'), ('F', 'I'), ('C', 'I'), ('C', 'A')])

# Define the CPDs
cpd_T = TabularCPD('T', 2, [[0.92], [0.08]])
cpd_C = TabularCPD('C', 2, [[0.4], [0.6]])

cpd_F = TabularCPD('F', 2, [[0.998, 0.98], [0.002, 0.02]], ['T'], [2])
cpd_A = TabularCPD('A', 2, [[0.99, 0.9], [0.01, 0.1]], ['C'], [2])

cpd_R = TabularCPD('R', 2, [[0.99, 0.9, 0.1, 0.1], [0.01, 0.1, 0.9, 0.9]], ['T', 'F'], [2, 2])
cpd_I = TabularCPD('I', 2, [[0.999, 0.989, 0.99, 0.98], [0.001, 0.011, 0.01, 0.02]], ['C', 'F'], [2, 2])

# Add CPDs to the model
bn.add_cpds(cpd_T, cpd_C, cpd_F, cpd_A, cpd_R, cpd_I)
```

```
def print_full(cpd):
    backup = TabularCPD._truncate_strtable
    TabularCPD._truncate_strtable = lambda self, x: x
    print(cpd)
    TabularCPD._truncate_strtable = backup

print_full(cpd_T)
print_full(cpd_C)
print_full(cpd_F)
print_full(cpd_A)
print_full(cpd_R)
print_full(cpd_I)
```

```
------+------+
| T(0) | 0.92 |
+------+------+
| T(1) | 0.08 |
+------+------+
+------+-----+
| C(0) | 0.4 |
+------+-----+
| C(1) | 0.6 |
+------+-----+
+------+-------+------+
| T    | T(0)  | T(1) |
+------+-------+------+
| F(0) | 0.998 | 0.98 |
+------+-------+------+
| F(1) | 0.002 | 0.02 |
+------+-------+------+
+------+------+------+
| C    | C(0) | C(1) |
+------+------+------+
| A(0) | 0.99 | 0.9  |
+------+------+------+
| A(1) | 0.01 | 0.1  |
+------+------+------+
+------+------+------+------+------+
| T    | T(0) | T(0) | T(1) | T(1) |
+------+------+------+------+------+
| F    | F(0) | F(1) | F(0) | F(1) |
+------+------+------+------+------+
| R(0) | 0.99 | 0.9  | 0.1  | 0.1  |
+------+------+------+------+------+
| R(1) | 0.01 | 0.1  | 0.9  | 0.9  |
+------+------+------+------+------+
+------+-------+-------+------+------+
| C    | C(0)  | C(0)  | C(1) | C(1) |
+------+-------+-------+------+------+
| F    | F(0)  | F(1)  | F(0) | F(1) |
+------+-------+-------+------+------+
| I(0) | 0.999 | 0.989 | 0.99 | 0.98 |
+------+-------+-------+------+------+
| I(1) | 0.001 | 0.011 | 0.01 | 0.02 |
+------+-------+-------+------+------+
```

using likelihood weighting

```
# Perform inference
inference = BayesianModelSampling(bn)

# Perform likelihood weighted sampling
samples = inference.likelihood_weighted_sample(evidence=[], size=100000, seed=1234)
```

```
print(samples[0:10])
```

```
   T  F  R  I  C  A  _weight
0  0  0  0  0  0  0      1.0
1  0  0  0  0  1  0      1.0
2  0  0  0  0  1  0      1.0
3  0  0  0  0  0  0      1.0
4  0  0  0  1  1  0      1.0
5  0  0  0  0  0  0      1.0
6  0  0  0  0  1  0      1.0
7  0  0  0  0  1  0      1.0
8  1  0  1  0  0  0      1.0
9  0  0  0  0  0  0      1.0
```

```
# Calculate the prior probability of a transaction being fraudulent
total_weight = samples['_weight'].sum()
prob_Fraudulent_1 = samples[samples['F'] == 1]['_weight'].sum() / total_weight
prob_Fraudulent_0 = samples[samples['F'] == 0]['_weight'].sum() / total_weight

print(f"Probability of F being 0: {prob_Fraudulent_0}")
print(f"Probability of F being 1: {prob_Fraudulent_1}")
```

```
Probability of F being 0: 0.9968
Probability of F being 1: 0.0032
```

```
# Calculate the probability that the current transaction is a fraud given some factors
total_weight = samples[(samples['R'] == 0) & (samples['I'] == 1) & (samples['A'] == 0)]['_weight'].sum()
prob_Fraudulent_1 = samples[(samples['F'] == 1) & (samples['R'] == 0) & (samples['I'] == 1) & (samples['A'] == 0)]['_weight'].sum() / total_weight
prob_Fraudulent_0 = samples[(samples['F'] == 0) & (samples['R'] == 0) & (samples['I'] == 1) & (samples['A'] == 0)]['_weight'].sum() / total_weight

print(f"Probability of F being 1: {prob_Fraudulent_1}")
print(f"Probability of F being 0: {prob_Fraudulent_0}")
```

```
Probability of F being 0: 0.9961013645224172
Probability of F being 1: 0.003898635477582846
```

using variable elimination

```
from pgmpy.inference import VariableElimination
```

```
# Calculate the probability of a transaction being fraudulent
inference = VariableElimination(bn)
prob_Fraudulent = inference.query(variables=['F'])

print(prob_Fraudulent)
```

```
+------+----------+
| F    |   phi(F) |
+======+==========+
| F(0) |   0.9966 |
+------+----------+
| F(1) |   0.0034 |
+------+----------+
```

```
# Calculate the probability of a transaction being fraudulent
inference = VariableElimination(bn)
prob_Fraudulent = inference.query(variables=['F'], evidence={'R': 0, 'I': 1, 'A': 0})

print(prod_Fraudulent)
```

```
+------+----------+
| F    |   phi(F) |
+======+==========+
| F(0) |   0.9948 |
+------+----------+
| F(1) |   0.0052 |
+------+----------+
```

conclusion

The answers for likelihood weighting and variable elimination do not differ by a large amount, since a large sample size of 100,000 was used. However, if the sample size were to be smaller, the probability calculated using likelihood weighting is likely to be less accurate.

3. After computing those probabilities, the fraud detection system raises a flag and recommends that the cardholder be called to confirm the transaction. An agent calls at the domicile of the cardholder but she is not home. Her spouse confirms that she is currently out of town on a business trip. How does the probability of a fraud change based on this new piece of information? [4 points]

using likelihood weighting

```
# Calculate the probability that the current transaction is a fraud given some factors
total_weight = samples[(samples['R'] == 0) & (samples['I'] == 1) & (samples['A'] == 0) & (samples['T'] == 1)]['_weight'].sum()
prob_Fraudulent_1 = samples[(samples['F'] == 1) & (samples['R'] == 0) & (samples['I'] == 1) & (samples['A'] == 0) & (samples['T'] == 1)]['_weight'].sum() / total_weight
prob_Fraudulent_0 = samples[(samples['F'] == 0) & (samples['R'] == 0) & (samples['I'] == 1) & (samples['A'] == 0) & (samples['T'] == 1)]['_weight'].sum() / total_weight

print(f"Probability of F being 0: {prob_Fraudulent_0}")
print(f"Probability of F being 1: {prob_Fraudulent_1}")
```

```
Probability of F being 0: 0.8333333333333334
Probability of F being 1: 0.16666666666666666
```

```
# Calculate the probability of a transaction being fraudulent
inference = VariableElimination(bn)
prob_Fraudulent = inference.query(variables=['F'], evidence={'R': 0, 'I': 1, 'A': 0, 'T': 1})

print(prob_Fraudulent)
```

```
+------+----------+
| F    |   phi(F) |
+======+==========+
| F(0) |   0.9493 |
+------+----------+
| F(1) |   0.0507 |
+------+----------+
```

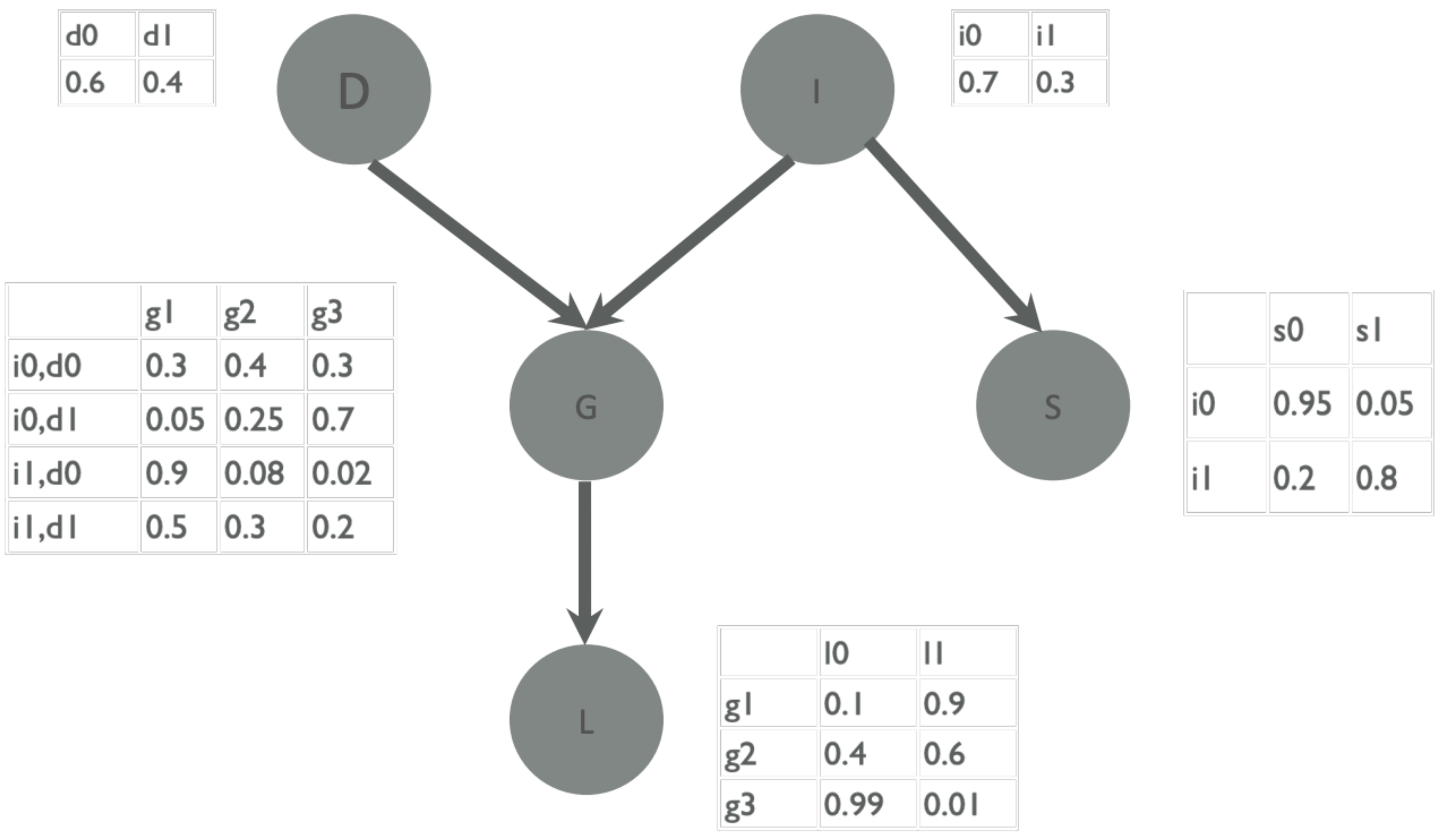
conclusion

The probability of a fraud increases based on the new piece of information that the cardholder is traveling.

Question 5. (10 pts)

You are given the following Bayes network with the conditional probabilities.

Please show how to compute the most probable explanation for S = s0 and G = g0 using variable elimination (forward and backward pass)? Provide the final answer.



d0	d1
0.6	0.4

i0	i1
0.7	0.3

	g1	g2	g3
i0,d0	0.3	0.4	0.3
i0,d1	0.05	0.25	0.7
i1,d0	0.9	0.08	0.02
i1,d1	0.5	0.3	0.2

	s0	s1
i0	0.95	0.05
i1	0.2	0.8

	l0	l1
g1	0.1	0.9
g2	0.4	0.6
g3	0.99	0.01

$$\max_{D,I,L} P(D,I,L|S=s_0,G=g_0)$$
$$= \max_{D,I,L} P(D)P(I)P(L|G=g_0)P(G=g_0|D,I)P(S=s_0|I)$$
$$= \max_{D,I} P(D)P(I)P(G=g_0|D,I)P(S=s_0|I)\max_{L} P(L|G=g_0)$$
$$= \max_{D,I} P(D)P(I)P(G=g_0|D,I)P(S=s_0|I)(0.9)$$
$$= \max_{I} P(I)P(S=s_0|I)\max_{D} P(D)P(G=g_0|D,I)(0.9)$$
$$= \max_{I} P(I)P(S=s_0|I)\max_{D} P(D)P(G=g_0|D,I)(0.9)$$
$$= \max_{I} P(I)P(S=s_0|I)\max_{D} \begin{bmatrix}0.6 & 0.4\end{bmatrix}\begin{bmatrix}0.3 & 0.05\\ 0.9 & 0.5\end{bmatrix}(0.9)$$
$$= \max_{I} P(I)P(S=s_0|I)[\max(0.6*0.3*0.9, 0.4*0.05*0.9), \max(0.6*0.9*0.9, 0.4*0.5*0.9)]$$
$$= \max_{I} P(I)P(S=s_0|I)\begin{bmatrix}0.162 & 0.486\end{bmatrix}$$
$$= \begin{bmatrix}0.7 & 0.3\end{bmatrix}\begin{bmatrix}0.95\\ 2\end{bmatrix}\begin{bmatrix}0.162 & 0.486\end{bmatrix}$$
$$= \max(0.7*0.95*0.162, 0.3*0.2*0.486)$$
$$= \max(0.10773, 0.02916)$$
$$\therefore I=i_0, D=d_0, L=l_1$$

Question 6. (10 pts)

Taxi drivers in Singapore can pick up customers from any location that is not on highways. However, they require guidance on where to pick up customers when they do not have customers on board and there are no bookings. In this question, we address this guidance problem.

There are four locations: L1, L2, L3 and L4 from where taxi drivers can pick up and drop off customers. At any decision epoch, the chances of the taxi driver picking up a customer from different locations are provided in Table 1 below:

Location	**Chance of finding customer**
L1	0.2
L2	0.6
L3	0.4
L4	0.7

Table 1

Once the taxi driver picks up a customer, the customer determines the destination. Observed probabilities (from past data) of a customer starting from a source location and going to a destination location:

Source→ Destination	**Probability**
L1 → L2	0.4
L1 → L3	0.2
L1 → L4	0.4
L2 → L1	0.6
L2 → L3	0.4
L3 → L1	0.6
L3 → L4	0.4
L4 → L1	0.4
L4 → L2	0.6

Table 2

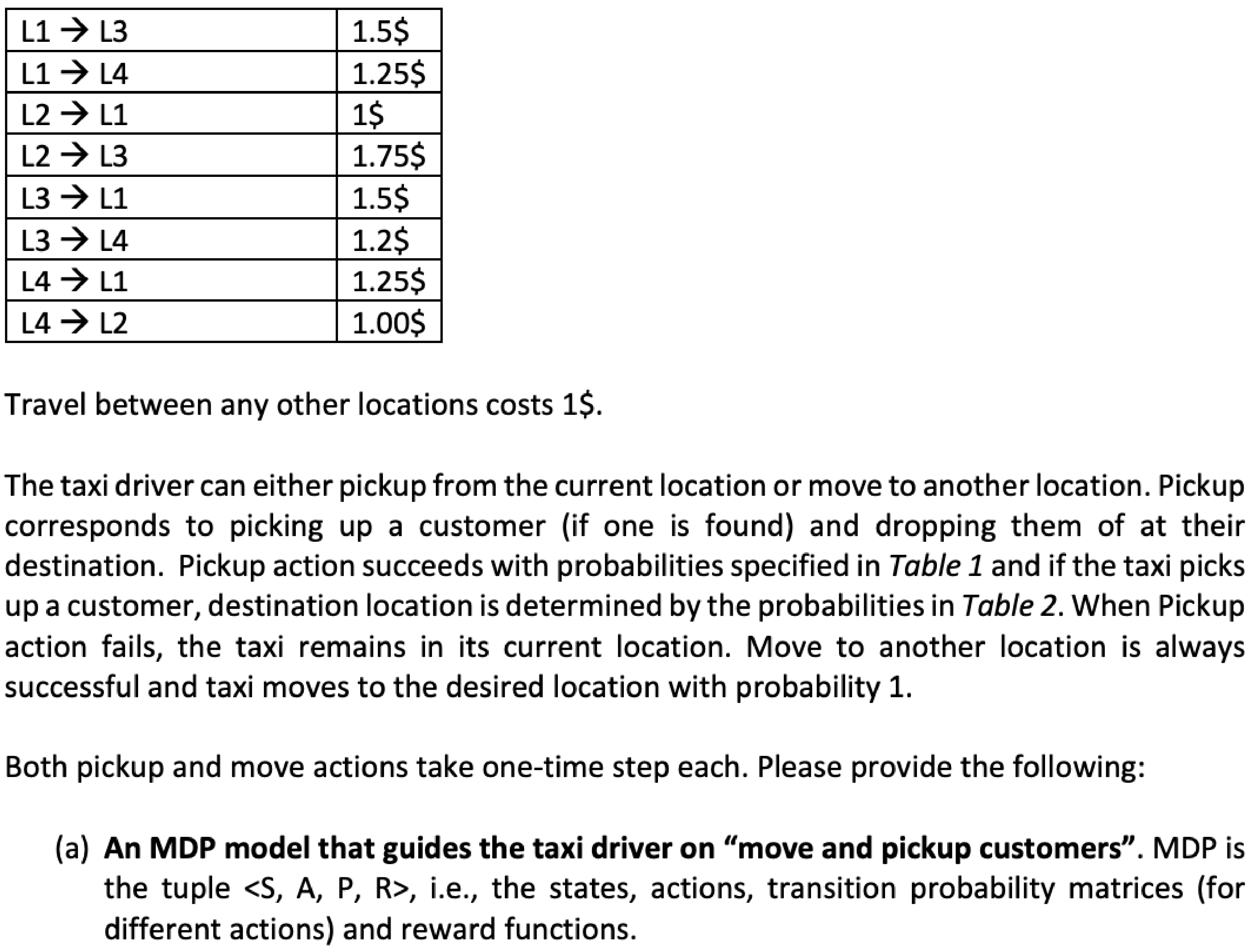
Just to avoid any confusion, first row of Table 2 below the heading row indicates that a customer picked up from L1 gets dropped off at L2 with 0.4 probability.

Travel fare between different locations are as follows:

Source→ Destination	**Fare**
L1 → L2	25$
L1 → L3	12$
L1 → L4	6$
L2 → L1	10$
L2 → L3	8$
L3 → L1	15$
L3 → L4	9$
L4 → L1	11$
L4 → L2	5$

Cost of travelling between locations is as follows:

Source→ Destination	**Fare**
L1 → L2	1$



L1 → L3	1.5$
L1 → L4	1.25$
L2 → L1	1$
L2 → L3	1.75$
L3 → L1	1.5$
L3 → L4	1.2$
L4 → L1	1.25$
L4 → L2	1.00$

Travel between any other locations costs 1$.

The taxi driver can either pickup from the current location or move to another location. Pickup corresponds to picking up a customer (if one is found) and dropping them of at their destination. Pickup action succeeds with probabilities specified in *Table 1* and if the taxi picks up a customer, destination location is determined by the probabilities in *Table 2*. When Pickup action fails, the taxi remains in its current location. Move to another location is always successful and taxi moves to the desired location with probability 1.

Both pickup and move actions take one-time step each. Please provide the following:

(a) **An MDP model that guides the taxi driver on "move and pickup customers"**. MDP is the tuple <S, A, P, R>, i.e., the states, actions, transition probability matrices (for different actions) and reward functions.

The state would be the current location of the taxi.

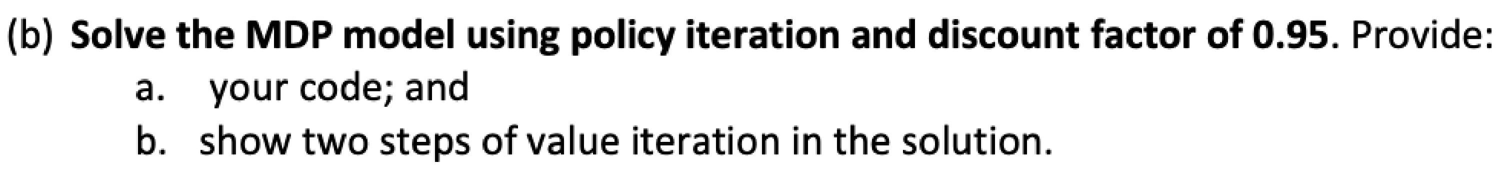
The actions would be the taxi moving to another location or staying in the current location to pick up a customer (if successful). This is represented as the location number ranging from 1 to 4. If the current location of the taxi is L1, pickup will be action 1 and traveling to L2, L3 or L4 will be actions 2, 3, 4 respectively.

The transitional probability matrix for different actions will be 1.

rewards

For the action of pickup, the reward will be the expected travel fare collected from the customer - the cost of travelling to the destination set by the customer. This is calculated using the probability of picking up a customer and the probability that the customer is going to the different destinations.

For the action of the taxi moving to another location, since the taxi does not gain any travel fare from the customer, the reward will be the cost of travelling to the location, which will be negative.



(b) **Solve the MDP model using policy iteration and discount factor of 0.95**. Provide:

a. your code; and

b. show two steps of value iteration in the solution.

expected value of a state-action pair, Q(s, a)

For the action of pickup, Q(s, a) is calculated using (1)(reward + discount_factor * V(current_location)

For the action of the taxi moving to another location, Q(s, a) is calculated using (1)(reward + discount_factor * V(next_location). This is because the value of the next location should be taken into account, in combination with the cost of moving to the next location.

expected value of a state, V(s)

V(s) is held in the dictionary v, whilst the "new" V(s) are kept in the dictionary updating_v, so that the values from V(s) can still be used for the current round of calculations, even when new values are calculated.

best policy, π(s)

The best policy for each state is held in a dictionary p, with the starting policy as {1:4, 2:3, 3:2, 4:1}.

```
import copy

finding_customer = {1:0.2, 2: 0.6, 3: 0.4, 4:0.7}
destination={1:[2, 3, 4], 2:[1, 3], 3:[1, 4], 4:[1, 2]}
customer_destination={1:{2:0.4, 3: 0.2, 4: 0.4}, 2:{1:0.6, 3: 0.4}, 3:{1:0.6, 4:0.4}, 4:{1:0.4, 2:0.6}}
fare_cost = {1:{2:25, 3:12, 4:6}, 2:{1:10, 3:8}, 3:{1:15, 4:9}, 4:{1:11, 2:5}}
```

```
travel_cost = {1:{2:1, 3:1.5, 4:1.25}, 2:{1:1, 3:1.75, 4:1}, 3:{1:1.5, 2:1, 4:1.2}, 4:{1:1.25, 2:1, 3: 1}}
discount_factor = 0.95

v = {1:0, 2:0, 3:0, 4:0}
updating_v = {1:0, 2:0, 3:0, 4:0}

# best policy for each state
p = {1:4, 2:3, 3:2, 4:1}

# calculate reward
rewards = []

for l in range(1, 5):
  for a in range(1, 5):
    reward = 0
    if l == a:
      for d in destination[l]:
        reward += finding_customer[a] * customer_destination[a][d] * ((fare_cost[a][d] - travel_cost[a][d]))
    else:
      reward = -travel_cost[l][a]
    rewards.append(reward)

rewards = np.array(rewards)
rewards = rewards.reshape(4, 4)
print(rewards)

for i in range(10):
  for l in range(1, 5):
    # determine value of policy
    curr_q = rewards[l-1][p[l]-1] + discount_factor * v[p[l]]

    max_q = curr_q
    best_policy = p[l]

    # check if a better action exists
    for a in range(1, 5):
      if (p[l] != a):
        q = rewards[l-1][a-1] + discount_factor * v[a]

        # update policy
        if q > max_q:
          max_q = q
          best_policy = a

    updating_v[l] = max_q
    p[l] = best_policy

  v = copy.deepcopy(updating_v)

  print(f"v: {v}")
  print(f"p: {p}")
```

```
[[ 2.72  -1.    -1.5   -1.25 ]
 [-1.     4.74  -1.75  -1.   ]
 [-1.5   -1.     4.488 -1.2  ]
 [-1.25  -1.    -1.     4.41 ]]
v: {1: 2.72, 2: 4.74, 3: 4.4879999999999995, 4: 4.409999999999999}
p: {1: 1, 2: 2, 3: 3, 4: 4}
v: {1: 5.304, 2: 9.243, 3: 8.7516, 4: 8.599499999999999}
p: {1: 1, 2: 2, 3: 3, 4: 4}
v: {1: 7.780849999999999, 2: 13.52085, 3: 12.802019999999999, 4: 12.579524999999997}
p: {1: 2, 2: 2, 3: 3, 4: 4}
v: {1: 11.844807499999998, 2: 17.584807499999997, 3: 16.649918999999997, 4: 16.360548749999996}
p: {1: 2, 2: 2, 3: 3, 4: 4}
v: {1: 15.705567124999995, 2: 21.445567124999997, 3: 20.305423049999995, 4: 19.952521312499993}
p: {1: 2, 2: 2, 3: 3, 4: 4}
v: {1: 19.373288768749998, 2: 25.113288768749996, 3: 23.778151897499992, 4: 23.364895246874994}
p: {1: 2, 2: 2, 3: 3, 4: 4}
v: {1: 22.857624330312493, 2: 28.59762433031249, 3: 27.07724430262499, 4: 26.606650484531244}
p: {1: 2, 2: 2, 3: 3, 4: 4}
v: {1: 26.167743113796867, 2: 31.90774311379687, 3: 30.21138208749374, 4: 29.68631796030468}
p: {1: 2, 2: 2, 3: 3, 4: 4}
v: {1: 29.312355958107023, 2: 35.05235595810702, 3: 33.18881298311905, 4: 32.61200206228944}
p: {1: 2, 2: 2, 3: 3, 4: 4}
v: {1: 32.29973816020167, 2: 38.03973816020167, 3: 36.017372333963095, 4: 35.391401959174964}
p: {1: 2, 2: 2, 3: 3, 4: 4}
```

Hence, the best policies for each state are: {1: 2, 2: 2, 3: 3, 4: 4}.

(c) Solve the MDP model using linear programming formulation and discount factor of 0.95. Provide:

a. Your LP formulation
b. Your code

objective: minimise V(s)

constraint: $V(s) \geq \sum_{s'} Pr(s'|s,a)[R(s',a,s) + \gamma V(s')]$

```
import numpy as np
import pulp

finding_customer = {1:0.2, 2: 0.6, 3: 0.4, 4:0.7}
destination={1:[2, 3, 4], 2:[1, 3], 3:[1, 4], 4:[1, 2]}
customer_destination={1:{2:0.4, 3: 0.2, 4: 0.4}, 2:{1:0.6, 3: 0.4}, 3:{1:0.6, 4:0.4}, 4:{1:0.4, 2:0.6}}
fare_cost = {1:{2:25, 3:12, 4:6}, 2:{1:10, 3:8}, 3:{1:15, 4:9}, 4:{1:11, 2:5}}
travel_cost = {1:{2:1, 3:1.5, 4:1.25}, 2:{1:1, 3:1.75, 4:1}, 3:{1:1.5, 2:1, 4:1.2}, 4:{1:1.25, 2:1, 3: 1}}
discount_factor = 0.95
```

```
v = {1:0, 2:0, 3:0, 4:0}

rewards = []

# calculate reward
for l in range(1, 5):
  for a in range(1, 5):
    reward = 0
    if l == a:
      for d in destination[l]:
        reward += finding_customer[a] * customer_destination[a][d] * ((fare_cost[a][d] - travel_cost[a][d]))
    else:
      reward = -travel_cost[l][a]
    rewards.append(reward)

rewards = np.array(rewards)
rewards = rewards.reshape(4, 4)
print(rewards)

v = pulp.LpVariable.dicts("s", range(4)) # number of states
prob = pulp.LpProblem("taxi", pulp.LpMinimize)
prob += sum([v[i] for i in range(4)]) # defines the objective function
for l in range(4):
  prob += v[l] >= sum(rewards[l][a] + discount_factor * v[a] for a in range(4))
  prob += v[l] >= -25
  prob += v[l] <= 25

prob.solve()

print("Status:", pulp.LpStatus[prob.status])

V = np.zeros(4) # value function
for i in range(4):
    V[i] = v[i].varValue

# extract the optimal policy
pi_star = np.zeros((4), dtype=np.int64)
vduales = np.zeros((4, 4))
s = 0
a = 0
for name, c in list(prob.constraints.items()):
    vduales [s, a] = c.pi
    if a < 4 - 1:
        a = a + 1
    else:
        a = 0
        if s < 4 - 1:
            s = s + 1
        else:
            s = 0

print(vduales)
for s in range(4):
    pi_star[s] = np.argmax(vduales [s, :]) + 1

print(pi_star)
```

```
[[ 2.72  -1.    -1.5   -1.25 ]
 [-1.     4.74  -1.75  -1.   ]
 [-1.5   -1.     4.488 -1.2  ]
 [-1.25  -1.    -1.     4.41 ]]
Status: Optimal
[[0. 1. 0. 0.]
 [1. 0. 0. 1.]
 [0. 0. 1. 0.]
 [0. 0. 0. 0.]]
[2 1 3 1]
```

Hence, the best policies for each state are: {1: 2, 2: 1, 3: 3, 4: 1}