Intro to AI - Assignment 2

Yousef Attia, Ameel Jani, Fauzan Amjad

April 9, 2022

Question 1

a)

As stated in the problem, this can be done through a simple JPD lookup simplified based on the structure of the given Bayesian Network. So, P(A=T, B=T, C=T, D=T, E=T)= P(A=T) * P(B=T) * P(C=T) * P(D=T | $A = T, B = T$)$*P(E = T \mid B = T, C = T)$

This equals 0.2*0.5*0.8*0.1*0.3, which is 0.0024

b)

As stated in the problem, this can be done through a simple JPD lookup simplified based on the structure of the given Bayesian Network. So, P(A=F, B=F, C=F, D=F, E=F)= P(A=F) * P(B=F) * P(C=F) * P(D=F | $A = F, B = F$)$*P(E = F \mid B = F, C = F)$

I calculated some of these probabilities not found in the given tables by doing 1 minus the given opposite probability in the tables.

This equals 0.8*0.5*0.2*0.1*0.8, which is 0.0064

c)

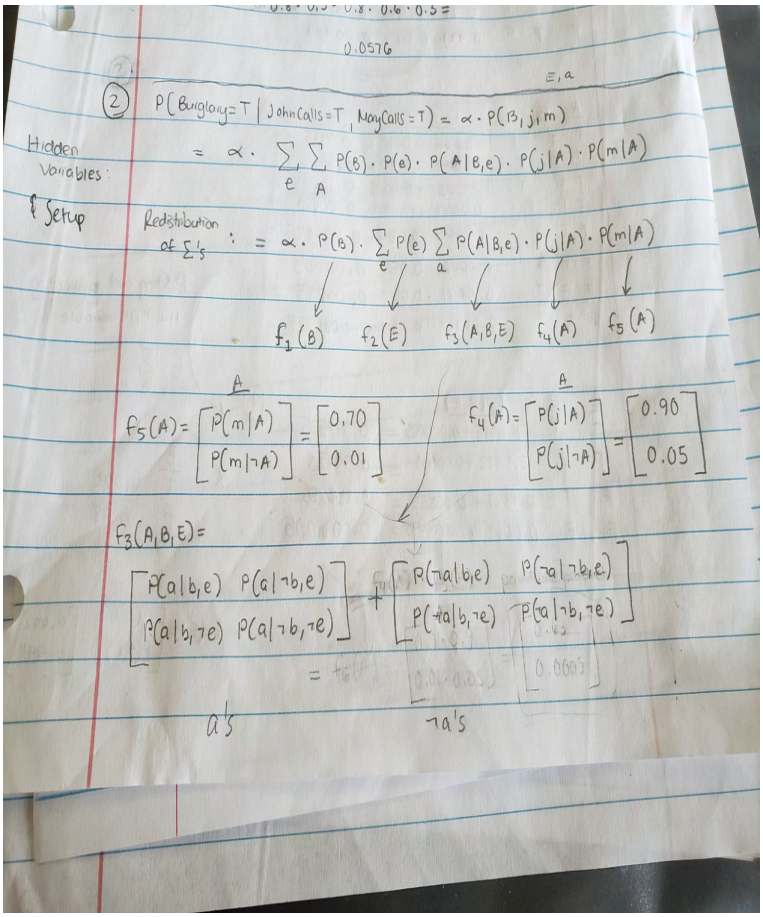
P(A=F | $B, C, D, E) = P(A = F) * P(B = T) * P(C = T) * P(D = T \mid A = F, B = T) * P(E = T \mid B = T, C = T)$

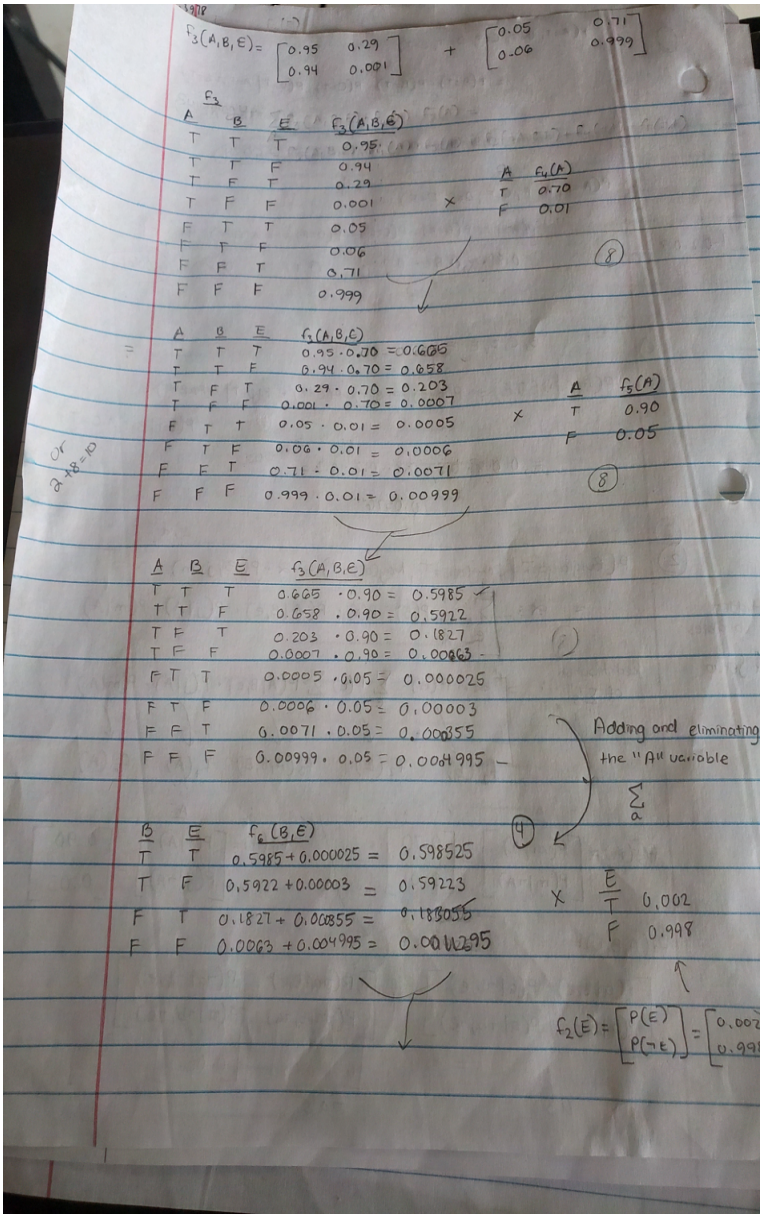
This equals 0.8*0.5*0.8*0.6*0.3, which is 0.0576

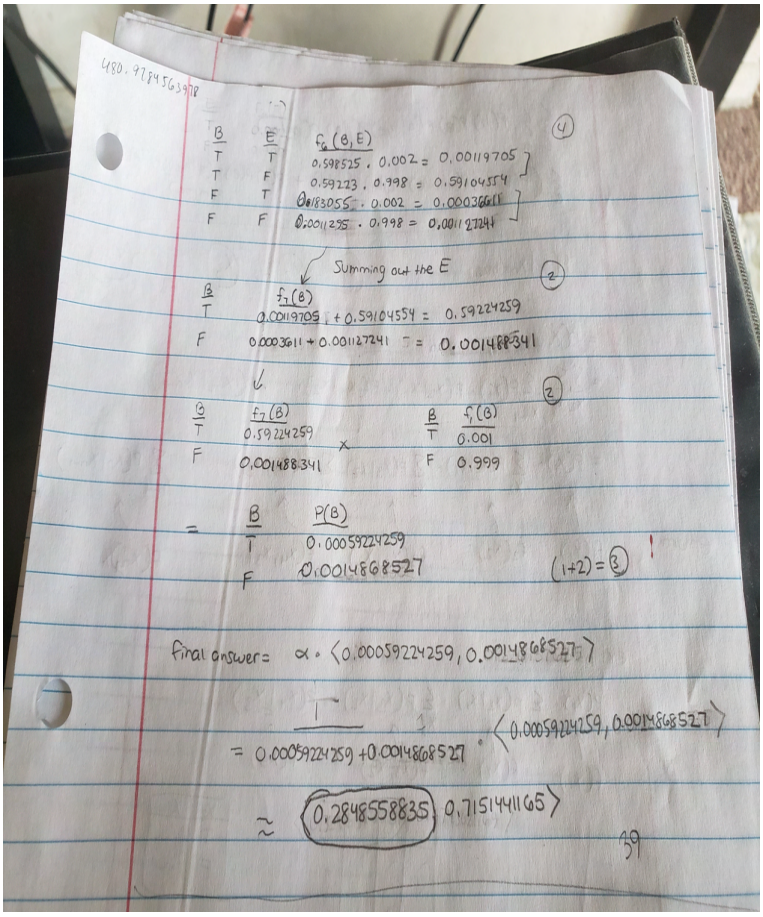
Question 2

a)

My final answer for this is 0.2848558835. Work is shown below (Numbers in circles are a count of calculations):




$f_3(A,B,E) = \begin{bmatrix} 0.95 & 0.29 \\ 0.94 & 0.001 \end{bmatrix} + \begin{bmatrix} 0.05 & 0.71 \\ 0.06 & 0.999 \end{bmatrix}$
f_3
A B E $f_3(A,B,E)$
T T T 0.95
T T F 0.94
T F T 0.29
T F F 0.001
F T T 0.05
F T F 0.06
F F T 0.71
F F F 0.999
A $f_4(A)$
T 0.70
F 0.01
(8)
A B E $f_3(A,B,E)$
T T T 0.95 · 0.70 = 0.665
T T F 0.94 · 0.70 = 0.658
T F T 0.29 · 0.70 = 0.203
T F F 0.001 · 0.70 = 0.0007
F T T 0.05 · 0.01 = 0.0005
F T F 0.06 · 0.01 = 0.0006
F F T 0.71 · 0.01 = 0.0071
F F F 0.999 · 0.01 = 0.00999
A $f_5(A)$
T 0.90
F 0.05
(8)
A B E $f_3(A,B,E)$
T T T 0.665 · 0.90 = 0.5985
T T F 0.658 · 0.90 = 0.5922
T F T 0.203 · 0.90 = 0.1827
T F F 0.0007 · 0.90 = 0.00063
F T T 0.0005 · 0.05 = 0.000025
F T F 0.0006 · 0.05 = 0.00003
F F T 0.0071 · 0.05 = 0.000355
F F F 0.00999 · 0.05 = 0.000495
Adding and eliminating the "A" variable
$\sum_a$
(4)
B E $f_6(B,E)$
T T 0.5985 + 0.000025 = 0.598525
T F 0.5922 + 0.00003 = 0.59223
F T 0.1827 + 0.000355 = 0.183055
F F 0.00063 + 0.000495 = 0.001125
E
T 0.002
F 0.998
$f_2(E) = \begin{bmatrix} P(E) \\ P(\neg E) \end{bmatrix} = \begin{bmatrix} 0.002 \\ 0.998 \end{bmatrix}$



b)

For the calculation of this value by enumeration, I separated out the sigma summation symbol to make calculation more efficient.

Specifically:

$$P(B \mid J, M) = a * P(B) * \sum_{e} * P(e) * \sum_{A} P(A \mid B, e) * P(j \mid A) * P(M \mid A) \qquad (1)$$

When this is expanded, it takes 15 individual multiplication and addition operations to simplify the parts of this expression without the alpha. It takes another 15 to calculate the denominator of the alpha and another 1 to divide 1 by the alpha's denominator to get the actual alpha. It also takes 1 operation to multiply the alpha value with the calculated

expression to obtain the final answer. Furthermore, since the probabilities

$$P(\neg e), P(\neg A \mid B, e), and P(\neg A \mid B, \neg e) \tag{2}$$

cannot be looked up in the given table and must each be calculated through a subtraction from involving data in the given tables, 3 more operations must be added.

In total, enumeration takes up 15+15+1+1+3, which is 35 individual operations.

For the calculation of the 2a value by variable elimination, as shown above, I multiplying (through pointwise product) the factors with each other and summing out variables in the factors whenever a sigma appears in the formula. As you can see, the process of computing the pointwise product of f3 and f4 takes 8 multiplications. Similarly, the process of computing the pointwise product of f3 and f5 also takes 8 multiplications. Then, summing out A from f3 takes 4 additions and produces an additional factor f6(B, E). Multiplying this by f2 requires 4 multiplications. Next, summing out the E takes 2 multiplications and creates a new factor f7(B). Subsequently, multiplying f7 by f1 takes 2 multiplications. Finally, calculating the denominator of the alpha takes 1 addition and 1 division while the process of multiplying this to the first element in the remaining vector requires 1 more multiplication. Furthermore, since the probabilities

$$P(\neg B), P(\neg e), P(\neg A \mid B, e), P(\neg A \mid \neg B, e), P(\neg A \mid \neg B, \neg e) P(\neg A \mid B, \neg e) \tag{3}$$

cannot be looked up in the given table and must each be calculated through a subtraction from involving data in the given tables, 6 more operations must be added. This leads to a total of 8+8+4+4+2+2+1+1+1+6= 41. Though, in the beginning, if I had multiplied f4 and f5 first before multiplying that product to f3 (which is more efficient), the operations would be 2+8+4+4+2+2+1+1+1+6= 35.

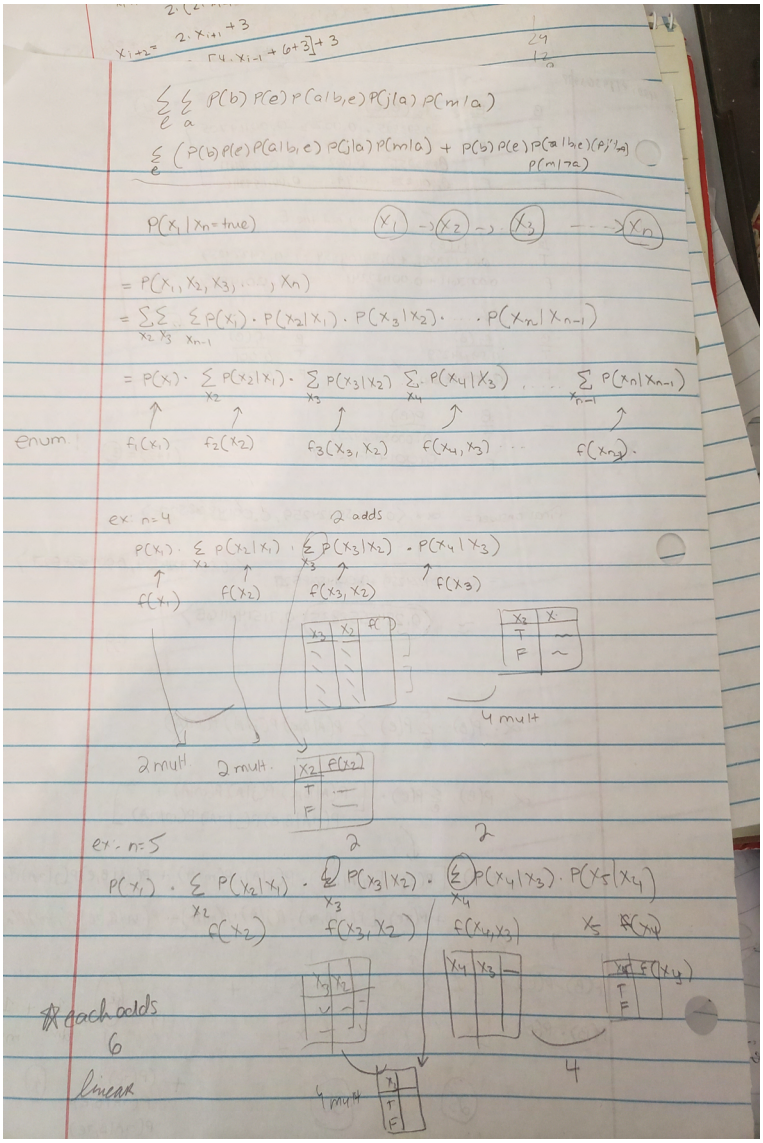
c)

In this case, I believe that the time complexity of the calculation of this value via variable enumeration would be linear. For each

$$X_{3:n} \tag{4}$$

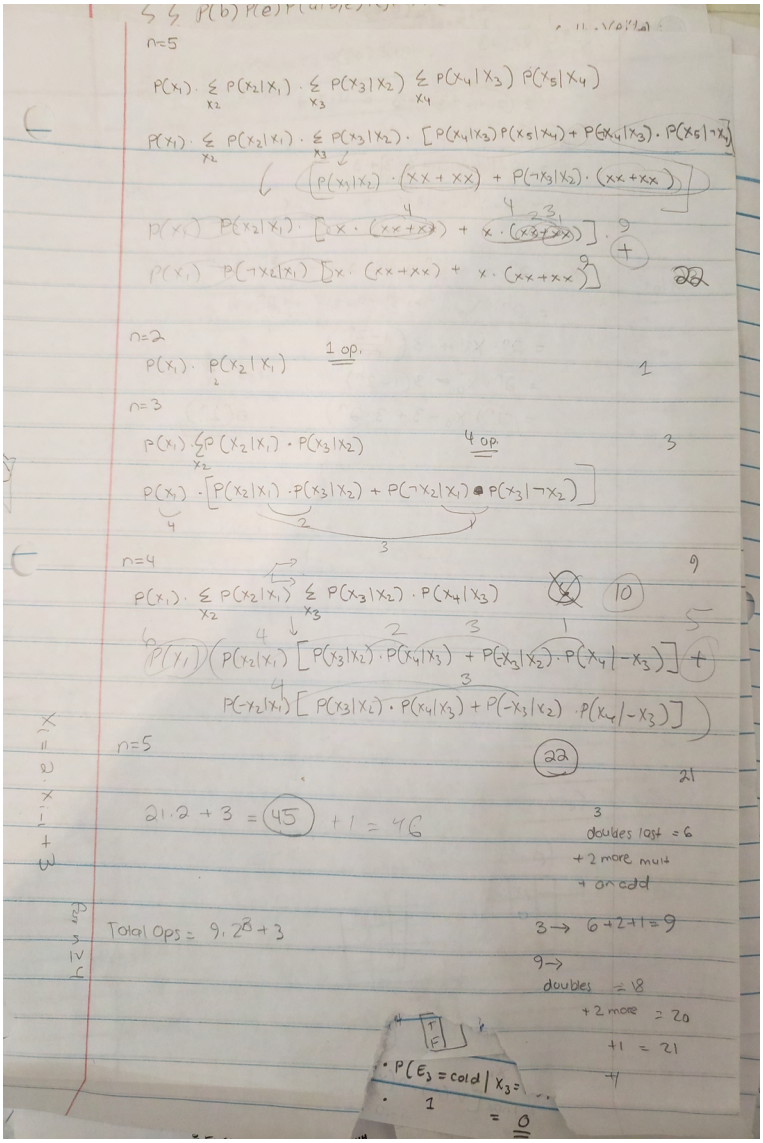
, inclusive, an additional increase in the n value would result in 6 (for additional multiplications and addition operations) + 2 (for looking up the negative values) operations added. So, the total number of operations for any n would be approximately 4+8(n-2). The time complexity of this would therefore be $O(N)$.

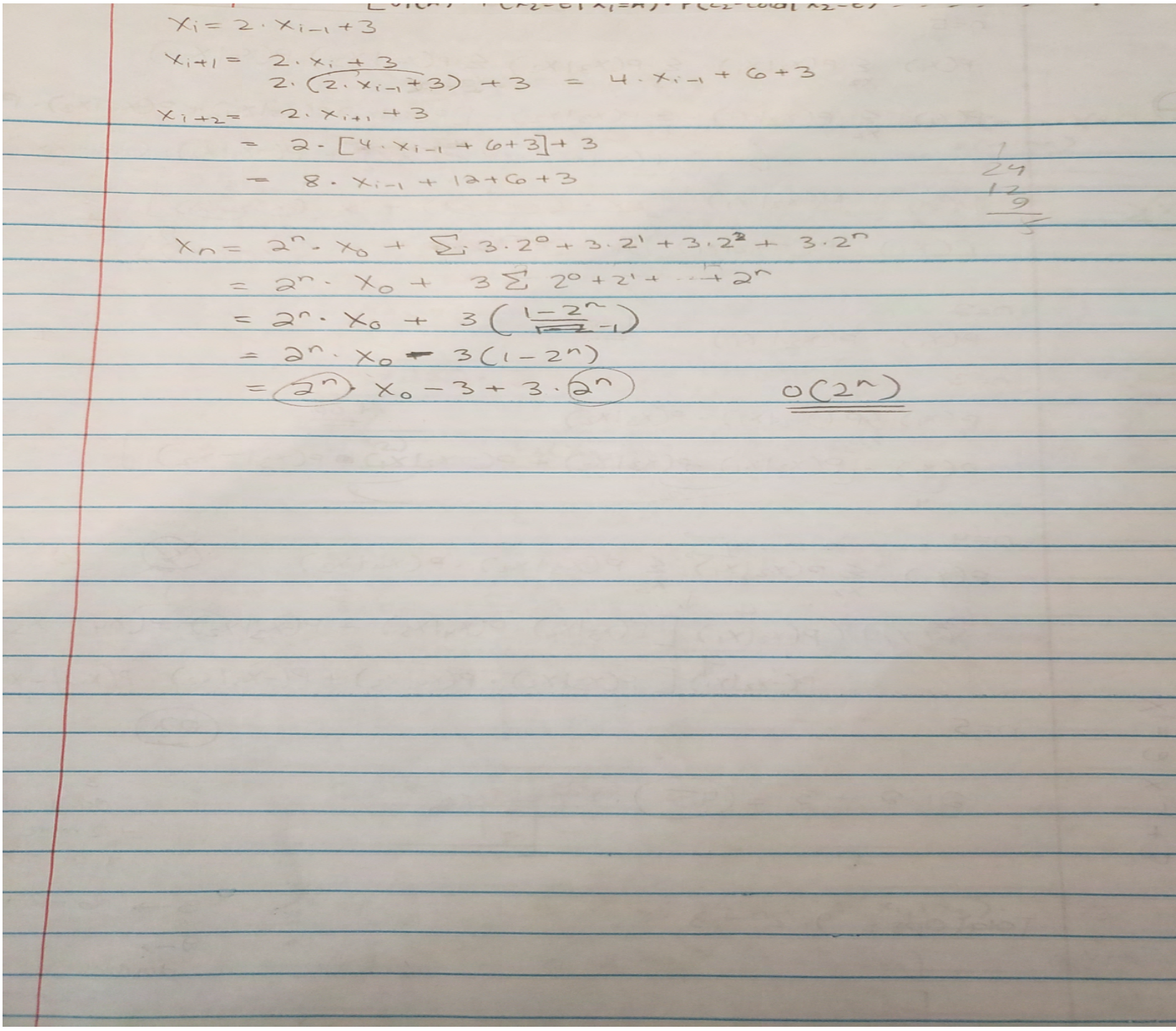
Picture attached to show this in more detail:



Additionally, to solve this problem by enumeration, I believe that the time complexity is exponential. I think this is the case because for n=4 and upward values of n, each additional n adds approximately 2*2*(the number of operations for n-1)+3 operations. As you can see in the image below, when this is put into equation form and simplified using the methods we learned in 344, the time complexity ends up being $O(2^n)$.

Picture attached to show this in more detail:


n=5
n=2
1 op.
n=3
4 op.
n=4
n=5
Total Ops = 9.2^3 + 3
3 doubles last = 6
+2 more mult
+ an add
3 → 6+2+1=9
9 → doubles = 18
+2 more = 20
+1 = 21
T
F
P(E3 = cold | X3 =
1
= 0


$x_i = 2 \cdot x_{i-1} + 3$
$x_{i+1} = 2 \cdot x_i + 3$
$2 \cdot (2 \cdot x_{i-1} + 3) + 3 = 4 \cdot x_{i-1} + 6 + 3$
$x_{i+2} = 2 \cdot x_{i+1} + 3$
$= 2 \cdot [4 \cdot x_{i-1} + 6 + 3] + 3$
$= 8 \cdot x_{i-1} + 12 + 6 + 3$
$x_n = 2^n \cdot x_0 + \sum 3 \cdot 2^0 + 3 \cdot 2^1 + 3 \cdot 2^2 + \cdots + 3 \cdot 2^n$
$= 2^n \cdot x_0 + 3 \sum 2^0 + 2^1 + \cdots + 2^n$
$= 2^n \cdot x_0 + 3\left(\frac{1-2^n}{1-2}\right)$
$= 2^n \cdot x_0 + 3(1-2^n)$
$= 2^n \cdot x_0 - 3 + 3 \cdot 2^n$
$O(2^n)$

Question 3

a)

Using Enumeration to find $P(d \mid c)$

	b	b	$\neg b$	$\neg b$
	d	$\neg d$	d	$\neg d$
c	0.30375	0.10125	0.0225	0.0225
$\neg c$	0.0495	0.4455	0.011	0.044

Using the Bayesian Network, we were able to construct the table shown directly above. If you add up all the elements in the table, it adds up to 1, which means all the possible events regarding parameters b, c, and d are represented within this table. Let's find $P(d \mid c)$. If we dissect this given probability into written terms, it would be represented as the probability of d given that c occurs, so $\neg c$ does not occur based on our current knowledge base. From our "Using Enumeration to find $P(b \mid c)$" section, we know that in order for c to occur, b must occur. You can read the proof or the reasoning behind how we derived this logic in the section 3 sections after this section. But it essence, if we're given that c happens, this directly means that b happens. Now we can look at the table that we derived from the Bayesian Network to find $P(d \mid c)$. We know that c occurs, so we'll only consider the third row of the probability table. We also know that if c occurs, then b must occur; therefore, we will only consider the first two columns of row 3. We're interested in seeing $P(b \mid c)$ so we can calculate that as follows: $P(d \mid c) = (0.30375)/(0.30375 + 0.10125) = 0.75$.

Rejection Sampling to find $P(d \mid c)$

Let's employ rejection sampling to find $P(d \mid c)$. Please refer to Part1RS.java to get the rejection sampling algorithm to find $P(d \mid c)$. The algorithm rejects the sample when the evidence, in this case c, is negated. The algorithm was able to construct an approximation for $P(d \mid c)$, which was 0.75638051044082353. The approximation is very close to the actual value. Here's a snippet of the terminal:

```
- + - E reject
- + - E reject
- - - E reject
- - - E reject
- + - E reject
- + + + accept
- + - E reject
- + + + accept
- + - E reject
- + - E reject
- + - E reject
- + - E reject
- - - E reject
P(d | c) = 0.7563805104408353

 ----jGRASP: operation complete.
```

Likelihood Weighting to find $P(d \mid c)$

Let's now employ likelihood weighting to approximate the value of $P(d \mid c)$. Rather than

rejecting when a parameter does not match our evidence, we force the evidence in and keep track of the weights. In this case, the evidence is c. At the end of the algorithm that constructed the 1000 samples, it calculated $P(d \mid c)$ via the sample weights. Please refer to Part1LW.java to get the likelihood weighting algorithm. The approximation for $P(d \mid c)$ via likelihood weighting came out to 0.7474972191323693. The approximation is very close to the actual value. Here's a snippet of the terminal:

Using Enumeration to find $P(b \mid c)$

	a	a	$\neg a$	$\neg a$
	c	$\neg c$	c	$\neg c$
b	0.2	0.8	0.5	0.5
$\neg b$	0.6	0.4	0	1.0

We were able to construct the table shown directly above via the given Bayesian network in the problem. Now let's find $P(b \mid c)$. $P(b \mid c)$ can first be rewritten using Bayes' theorem:

$$P(b|c) = P(b)\frac{P(c|b)}{P(c)} \tag{5}$$

Now that we've rewritten $P(b \mid c)$ into probabilities that we can derive and plug into the equation, let's find the the probability. $P(b)$ is already given to us by the Bayesian network and it does not have any dependencies, so we'll leave $P(b) = 0.9$. We're left with $P(c|b)$ and $P(c)$. The $P(c \mid b)$ can be calculated via the table shown directly above. Since the probability of A is equal to 0, this means that we can ignore the probabilities under whether a actually happens because a never happens, not a will always happen - $P(\neg A) = 1.0$. So the probability that c occurs given that b occurs will equal 0.5, which corresponds to the second (right hand) half of the table, specifically the third row. Now we need to find $P(c)$. $P(c) = P(B)*P(C|B) + P(\neg B)*P(C|\neg B)$. We can add the probabilities respectively as $P(c) = (0.9)(0.5) + (0.1)(0.0)$. Now that we have the probabilities to plug into the Bayes' theorem form of $P(b \mid c)$, let's actually plug it in.

$$(0.9)\frac{0.5}{0.45} = 1.0 \tag{6}$$

For $P(b \mid c)$, we get 1.0. To put into different terms, if we know that c occurs, that means b has had to occur. c can only happen if b is true, so if we know that c is true, then we know that b is true.

Rejection Sampling to find $P(b \mid c)$

Let's employ rejection sampling to find $P(b \mid c)$. Please refer to Part2RS.java to get the rejection sampling algorithm to find $P(b \mid c)$. The algorithm rejects the sample when the evidence, which in this case is c, is negated. Node C is dependent on Node B; therefore, this algorithm was not efficient. However, it did get $P(b \mid c)$ by outputting 1.0 at the end of program. This is exactly the calculated value. Here's a snippet of the terminal:

Likelihood Weighting to find $P(b \mid c)$

Let's now employ likelihood weighting to approximate the value of $P(b \mid c)$. Rather than rejecting when a parameter does not match our evidence, we force the evidence in and keep track of the weights. In this case, the evidence is c, just like with the previous example. At the end of the algorithm that constructed the 1000 samples, it calculated $P(b \mid c)$ via the sample weights. Please refer to Part2LW.java to get the likelihood weighting algorithm. The approximation for $P(b \mid c)$ via likelihood weighting came out to 1.0. This is exactly the calculated value. Here's a snippet of the terminal:

```
- + + 0.5
- + + 0.5
- + + 0.5
- + + 0.5
- + + 0.5
- + + 0.5
- + + 0.5
- + + 0.5
- - + 0.0
P(b | c) = 1.0

 ----jGRASP: operation complete.
```

Using Enumeration to find $P(d \mid \neg a, b)$

	b	b	$\neg b$	$\neg b$
	d	$\neg d$	d	$\neg d$
c	0.30375	0.10125	0.0225	0.0225
$\neg c$	0.0495	0.4455	0.011	0.044

To compute the value of $P(d \mid \neg a, b)$, we will need to look at the table above again to derive the probability. $P(\neg a) = 1.0$ because $P(a) = 0.0$ and $P(a) + P(\neg a) = 1.0$. This means that the only way c and d occur is if $P(\neg a)$ happens because it's the only thing that can happen. Since we know that b happens, we can ignore the final two columns in the above table of probabilities. We can thus calculate $P(d \mid \neg a, b)$ as $P(d \mid \neg a, b) = (0.30375 + 0.0495)/(0.30375 + 0.0495 + 0.10125 + 0.4455) = 0.3925$.

Rejection Sampling to find $P(d \mid \neg a, b)$

Let's employ rejection sampling to find $P(d \mid \neg a, b)$. Please refer to Part3RS.java to get the rejection sampling algorithm to find $P(d \mid \neg a, b)$. The algorithm rejects the sample when the evidence, in this case $\neg a$ and b don't appear in the sample (when their counterpart is in the sample). The algorithm was able to construct an approximation for $P(d \mid \neg a, b)$ at 0.3980154355016538. The approximation is close to the actual value, but repeated iterations of the program were not so great. This might point to a limitation of rejection sampling that is discussed in Part C of this problem. Here's a snippet of the terminal:

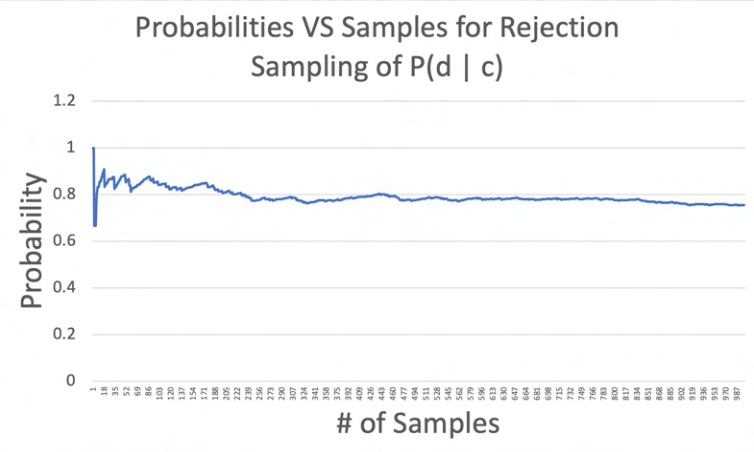
```
- + + + accept
- + + + accept
- + + - accept
- + + + accept
- - E E reject
- + + + accept
- + + + accept
- + + + accept
- + - - accept
- + + + accept
- + - - accept
- + - - accept
- + - - accept
- + + - accept
P(d | not a, b) = 0.3980154355016538

 ----jGRASP: operation complete.
```

Likelihood Weighting to find $P(d \mid \neg a, b)$

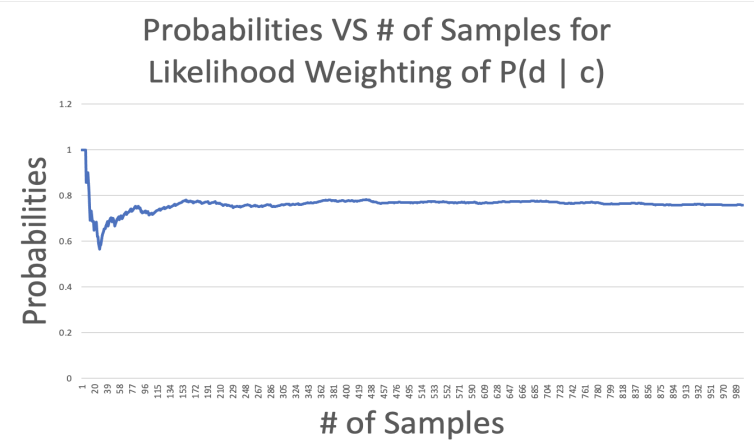
Let's no employ likelihood weighting to approximate the value of $P(d \mid \neg a, b)$. Rather than rejecting when the parameter does not match our evidence, we force the evidence in and keep track of the weights. In this case, the evidence is $\neg a$ and b, so we'll be forcing these in and accounting for it in via the weights At the end of the algorithm that constructed 1000 samples, it calculated $P(d \mid \neg a, b)$. Please refer to Part3LW.java to getthe likelihood weighting algorithm for this part of the problem. The approximation for $P(d \mid \neg a, b)$ came out to 0.3910000000000052. The approximation is very close to the actual value. Here's a snippet of the terminal:

b)

Below is the probability of $P(d \mid c)$ as a function of the number of samples used via rejection sampling. This was created by slightly altering the Part1RS.java program so that it would store the probability after every sample was made as opposed to at the end when 1000 samples have been created. These probabilities were stored in an ArrayList, outputted into a txt file, and then transferred to Excel where the graph below was created. The program responsible for this is named PartBRS.java. The output txt file is Probabilities_RS.txt.



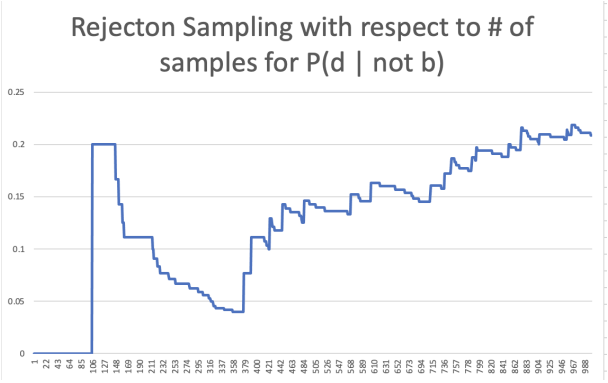
Below is the probability of $P(d \mid c)$ as a function of the number of samples used via likelihood weighting. This was created by slightly altering the Part1LW.java file so that a probability can be stored into an ArrayList after every sample was made. This program then outputted the probabilities into a txt file. I then transferred these probabilities to Excel where the graph below was created. The program responsible for this is named PartBLW.java. The output txt file is Probabilities_LW.txt.



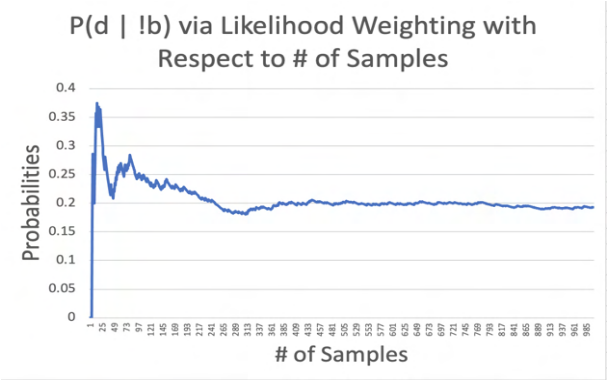
There are noticeable divergences in the convergence rates among the methods, particularly when the number of samples constructed is small. With regards to the rejection sampling probability plot with respect to the number of samples, it seems like there are many divergences in beginning, but it tries to level out (when there is an extreme on one end, another extreme balances it out on the other end). The likelihood weighting probability plot with respect to the number of samples has a different type of divergence where it seems to fall way below the true probability for the first 10s of samples, but over the course of the next couple hundred samples, it rises and then levels out to its true probability.

c)

In order for the convergence and effectiveness of rejection sampling to become noticeably worse than that of likelihood weighting, the probability of what we want to calculate should be small. This not makes it harder to rejection sampling to find the true probability because the true probability is low and, therefore, hard to calculate, but most samples will be rejected anyway, making rejection sampling hopelessly expensive. Likelihood weighting solves this expensive issue by ensuring that none of the samples get rejected because of the lack of evidence within the sample and forcing the evidence into every sample, adjusting for it via the sample weights. $P(d \mid \neg b)$ is a query that rejection sampling noticeably is bad at because the probability of the evidence is low. When the probability of the evidence is low, it's rejecting a lot. PartC_RS.java is the rejection sampling. Probabilities_3_RS.txt is where the probabilities are stored. Here is the rejection sampling plot. It came out 0.2087912087912088. Here is the plot.



As you can see, not a really good plot. Below is likelihood weighting.



As you can see, the likelihood weighting output is looking a lot nicer. The likelihood weighting program for this problem is called Probabilities_C_LW.java. The reason why this converges faster than rejection sampling is because all the samples in likelihood weighting

are relevant in finding the appropriate probability. In rejection sampling, we do not use all the samples because some of them are being rejected. In a query such as $P(d \mid \neg b)$, where the probability of the evidence is low, the rejecting sampling algorithm is rejecting a lot and cannot converge as fast as likelihood weighting to the true probability.

Thus, likelihood sampling is more efficient and attains the correct probability because it doesn't spend time rejecting samples we don't need and directly creates the samples we need.

Question 4

A:

On day 3 (assuming that the rover's position on day 1 is known to be A), the probability distribution for the rover's position is <0,0.2, 0.8, 0, 0, 0>, basically saying that the probability the rover is at B is 0.2, the probability that it is at A is 0.8, and the probability of it being anywhere else is 0.

I solved this problem using the filtering algorithm we learned in class:

Day 1:

Prior belief: <1,0,0,0,0,0>

Since the sensor reading of hot at this step is always correct, the above is also the same probability distribution on Day 1 after considering evidence.

Day 2:

Using only transition model/prediction=

$$P(X_2) = \sum_{X_1} * P(X_2 \mid X_1) * P(X_1) \tag{7}$$

In numerical terms, this is: <0.2, 0.8, 0, 0, 0, 0>*1 + <0.2, 0.8, 0, 0, 0, 0>*0, which is just <0.2, 0.8, 0, 0, 0, 0>

Then, bringing in the observation model:

$$P(X_2 \mid Cold_2) = a * P(Cold_2 \mid X_2) * P(X_2) \tag{8}$$

In numerical terms, this is: a*(<0,1,1,0,1,1> X <0.2, 0.8, 0, 0, 0, 0>) , which is just a* <0, 0.8, 0, 0, 0, 0>

This equals <0, 1, 0, 0, 0, 0>

Day 3:

Using only transition model/prediction:

$$P(X_3 \mid Cold_2) = \sum_{X_2} * P(X_3 \mid X_2) * P(X_2 \mid Cold_2) \tag{9}$$

This equals:

< 0, 0.2, 0.8, 0, 0, 0 > *0 + < 0, 0.2, 0.8, 0, 0, 0 > *1+< 0, 0.2, 0.8, 0, 0, 0 > *0 + < 0, 0.2, 0.8, 0, 0, 0 > *0+ < 0, 0.2, 0.8, 0, 0, 0 > *0 + < 0, 0.2, 0.8, 0, 0, 0 > *0

, which is just < 0, 0.2, 0.8, 0, 0, 0 >

Bringing in Evidence at Day 3:

$$P(X_3 \mid Cold_2, Cold_3) = a * P(Cold_3 \mid X_3) * P(X_3 \mid Cold_2) \tag{10}$$

a * < 0, 1 , 1 , 0, 1, 1 > X < 0, 0.2, 0.8, 0, 0, 0 >

This equals a * < 0, 0.2, 0.8, 0, 0, 0 >

, which is just < 0, 0.2, 0.8, 0, 0, 0 >, which is our final answer/probability distribution for all 6 squares

B:

On day 2 (assuming that the rover's position on day 1 is known to be A), the probability distribution for the rover's position is ¡0,1, 0, 0, 0, 0¿, basically saying that the probability the rover is at B is 1 and the probability of it being anywhere else is 0.

I solved this problem using the smoothing algorithm we learned in class:

The question is essentially asking:

$$P(X_2 \mid Hot_1, Cold_2, Cold_3) = a * P(X_2 \mid Hot_1, Cold_2) * P(Cold_3 \mid X_2) \tag{11}$$

The first part of the right-hand side of this equation is something that was calculated in step A, and it is < 0, 1, 0, 0, 0, 0 >.

The second part of the right-hand side, can, through the definition of smoothing, be turned into:

$$P(Cold_3 \mid X_2) = \sum_{X_3} P(Cold_3 \mid X_3) * P(E_{4:3} \mid X_3) * P(X_3 \mid X_2) \tag{12}$$

Ignoring the 2 states where the observation model described here returns 0, this can be simplified to:

1*1*<0, 0.2, 0.8, 0, 0, 0 > + 1*1*<0, 0.2, 0.8, 0, 0, 0 > + 1*1* <0, 0.2, 0.8, 0, 0, 0 > + 1*1* <0, 0.2, 0.8, 0, 0, 0 >,

which is <0, 0.8, 0.32 , 0, 0, 0 >

So, going back to the original equation,

$$P(X_2 \mid Hot_1, Cold_2, Cold_3) = \tag{13}$$

a*< 0, 1, 0, 0, 0, 0 > X <0, 0.8, 0.32 , 0, 0, 0 >

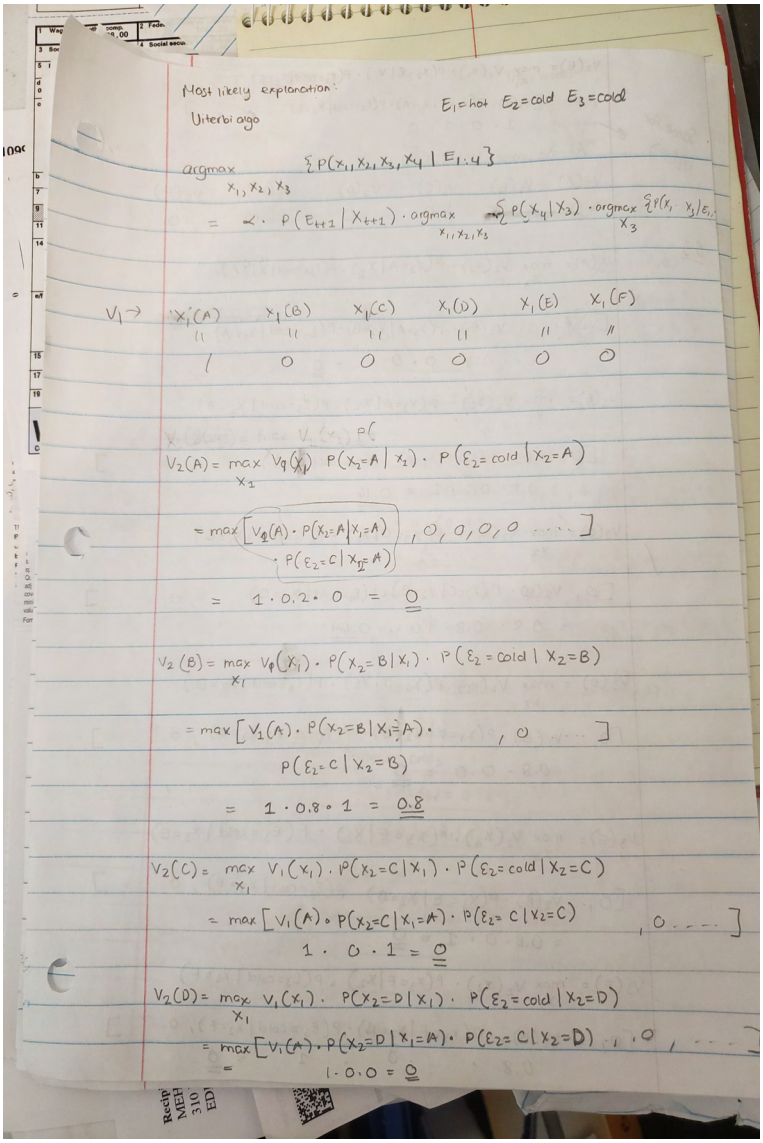
, which is a*< 0, 0.8, 0, 0, 0, 0 >

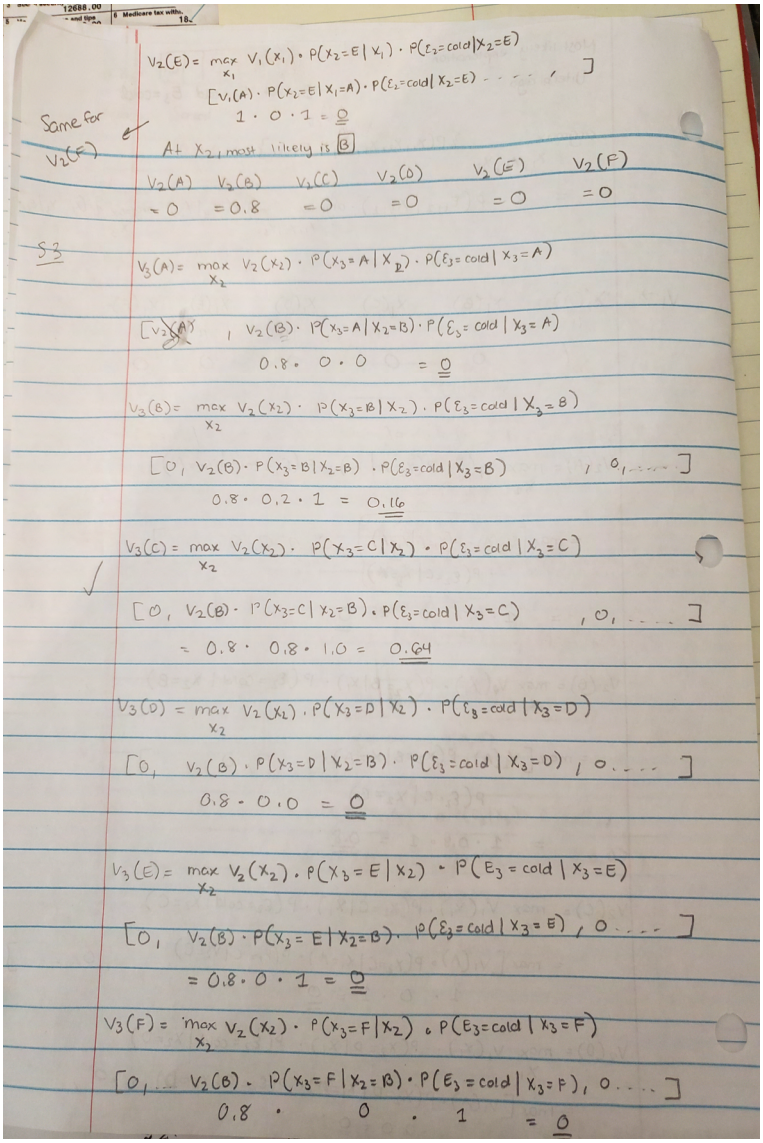
, which is < 0, 1, 0, 0, 0, 0 >

C:

The most likely sequence is A on Day 1, B on Day 2, and C on Day 3.

I used the Viterbi algorithm (as shown in the recitation example with taxis) to solve this:


Most likely explanation:
Viterbi algo
E_1 = hot E_2 = cold E_3 = cold
$\underset{X_1, X_2, X_3}{\text{argmax}} \{P(X_1, X_2, X_3, X_4 \mid E_{1:4})\}$
$= \alpha \cdot P(E_{t+1} \mid X_{t+1}) \cdot \underset{X_1, X_2, X_3}{\text{argmax}} \sum P(X_4 \mid X_3) \cdot \underset{X_3}{\text{argmax}} \{P(X_1, X_2, X_3 \mid E_{1:3})\}$
$V_1 \rightarrow$ $X_1(A)$ $X_1(B)$ $X_1(C)$ $X_1(D)$ $X_1(E)$ $X_1(F)$
1 0 0 0 0 0
$V_2(A) = \max_{X_1} V_1(X_1) \cdot P(X_2 = A \mid X_1) \cdot P(E_2 = \text{cold} \mid X_2 = A)$
$= \max [V_1(A) \cdot P(X_2 = A \mid X_1 = A) \cdot P(E_2 = C \mid X_2 = A), 0, 0, 0, 0 \ldots]$
$= 1 \cdot 0.2 \cdot 0 = 0$
$V_2(B) = \max_{X_1} V_1(X_1) \cdot P(X_2 = B \mid X_1) \cdot P(E_2 = \text{cold} \mid X_2 = B)$
$= \max [V_1(A) \cdot P(X_2 = B \mid X_1 = A) \cdot P(E_2 = C \mid X_2 = B), 0 \ldots]$
$= 1 \cdot 0.8 \cdot 1 = 0.8$
$V_2(C) = \max_{X_1} V_1(X_1) \cdot P(X_2 = C \mid X_1) \cdot P(E_2 = \text{cold} \mid X_2 = C)$
$= \max [V_1(A) \cdot P(X_2 = C \mid X_1 = A) \cdot P(E_2 = C \mid X_2 = C), 0 \ldots]$
$1 \cdot 0 \cdot 1 = 0$
$V_2(D) = \max_{X_1} V_1(X_1) \cdot P(X_2 = D \mid X_1) \cdot P(E_2 = \text{cold} \mid X_2 = D)$
$= \max [V_1(A) \cdot P(X_2 = D \mid X_1 = A) \cdot P(E_2 = C \mid X_2 = D), 0, \ldots]$
$= 1 \cdot 0 \cdot 0 = 0$



D:

$P(hot_4, hot_5, cold_6 | hot_1, cold_2, cold_3) = P(cold_6 | hot_4, hot_5, hot_1, cold_2, cold_3)$
We will start by calculating $P(hot_1)$, since the question states the rover fell in cell A, $P(hot_1) = 1$
now we can do:
$P(cold_2 | hot_1) = P(X_2 = B | X_1 = A) * P(hot_1)$
$= 1 * 1 = 1$
$P(cold_3 | cold_2, hot_1) = P(X_3 = C | X_2 = B) * P(cold_2 | hot_1)$

$= 0.8 * 1 = 0.8$
$P(hot_4|cold_3, cold_2, hot_1) = P(X_4 = D|X_3 = C) * P(cold_3|cold_2)$
$= 0.8 * 0.8 = 0.64$
since 5 is hot, then we are predicting $X_5 = D$
$P(hot_5|hot_4, cold_3, cold_2, hot_1) = P(X_5 = D|X_4 = D) * P(hot_4|cold_3, cold_2, hot_1)$
$= 0.2 * 0.64 = 0.128$
$P(cold_6|hot_4, hot_5, hot_1, cold_2, cold_3) = P(X_6 = E|X_5 = D) * P(hot_5|hot_4, cold_3, cold_2, hot_1)$
$= 0.8 * 0.128 = 0.1024$

E:

$P(X_4|hot_1, cold_2, cold_3) =$
$< P(X_4 = D|X_3 = C) * P(X_3|cold_2, hot_1), P(X_4 = C|X_3 = C) * P(X_3|cold_2, hot_1) >$
$=< 0.8 * 0.8, 0.2 * 0.8 >=$
$< 0.64, 0.16 >$

$P(X_5|hot_1, cold_2, cold_3)$
$=< [P(X_5 = D|X_4 = D) + P(X_5 = D|X_4 = C)] * P(X_3|cold_2, hot_1), [P(X_5 = E|X_4 = D) + P(X_5 = E|X_4 = C)] * P(X_3|cold_2, hot_1) >$
$=< [0.2 + 0.8] * 0.8, [0.8 + 0] * 0.8 >$
$=< 0.8, 0.64 >$

Question 5

A:

```
step 1:
[
[0.0020639834881320956, 0.010319917440660476, 0.019607843137254905],
[0.037151702786377715,  0.18575851393188855,   0.35294117647058826],
[0.3715170278637771,    0.0,                   0.020639834881320953]
]

step 2:
[
[1.7286084701815044e-05, 0.0001210025929127053, 0.0012100259291270528],
[0.0056006914433880724,  0.03920484010371651,   0.39204840103716504],
[0.5600691443388073,     0.0,                   0.01728608470181504]
]

step 3:
[
[8.8541500138961e-06,  6.197905009727271e-05, 3.4432805609595945e-05],
[8.19008876285389e-05, 0.005593609271278861,  0.00573306213399723],
[0.08040453627619046,  0.0,                   0.9080816254251837]
]

step 4:
[
[1.9288505665440432e-06, 1.3501953965808305e-05, 4.1672697425334267e-07],
[9.778200788730218e-07,  0.0003418619733287296,  3.656779199073081e-05],
[0.004869986021042481,   0.0,                    0.9947347588620525]
]
```

according to the calculations by the program, the rover is most likely at position (3,3) or the far right corner after executing the actions.

B:

see attached zip file.

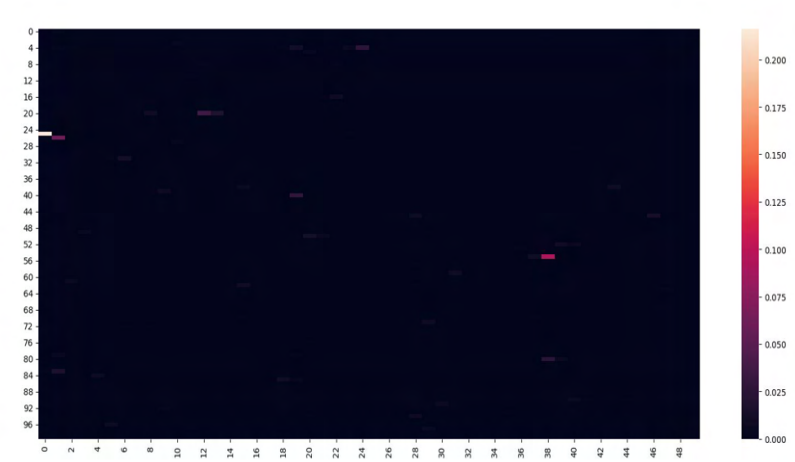


Figure 1: heatmap at iteration 10, ground truth position: (31, 6)

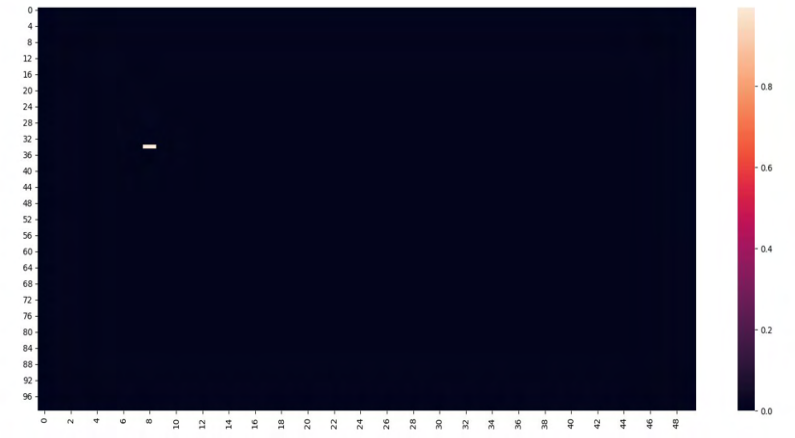


Figure 2: heatmap at iteration 50, ground truth position: (34, 8)

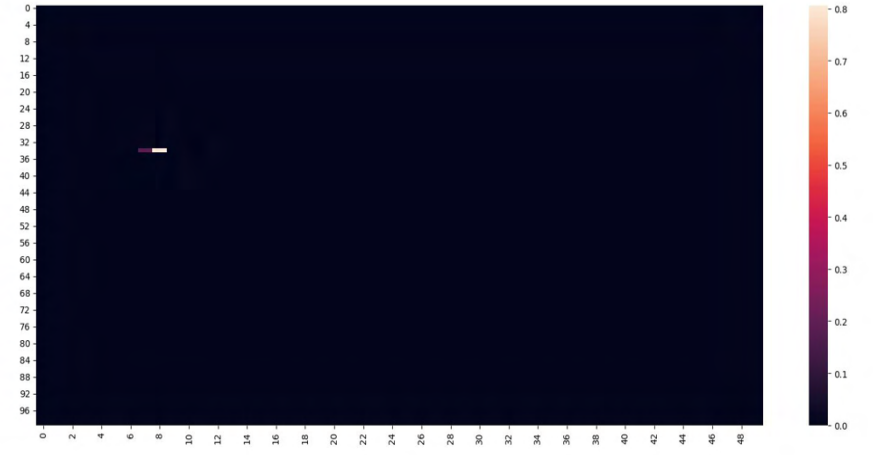
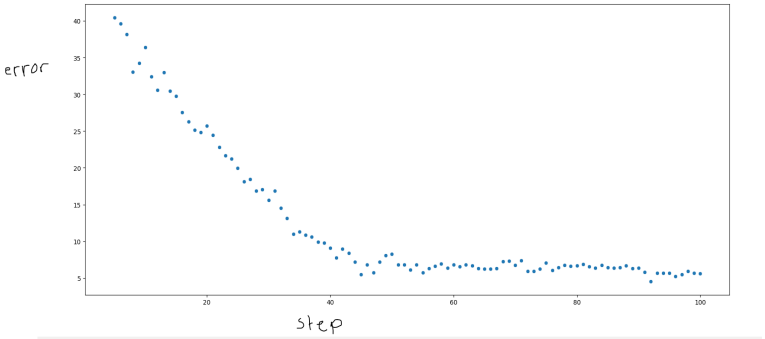


Figure 3: heatmap at iteration 100, ground truth position: (34, 8)

C:



as you can see the average error decreases with each iteration, as the filtering algorithm narrows down the agents position. For this plot I skipped the first five iterations as specified in the question.

Question 6

A:

In this question, there are two possible conditions of the vehicle, good quality or bad quality. I used the Expected Utility formula for choosing to buy the car to solve this.

$$E(U)(buycar \mid e) = \sum_i U(Result_i(buycar)) * P(Result_i \mid buycar, e) \tag{14}$$

This equals:

$$0.7 * (4000 - 3000) + 0.3 * (4000 - 1400 - 3000) \tag{15}$$

, which is 580 (dollars) .

B:

This question can be simply answered by using the Bayes' theorem to calculate the 4 probabilities given in the hint below.

From the 2 given probabilities,

$$P(\neg pass(c1) \mid q^+(c1)) = 1 - P(pass(c1) \mid q^+(c1)) = 0.2$$

and:

$$P(\neg pass(c1) \mid q^-(c1)) = 1 - P(pass(c1) \mid q^-(c1)) = 0.65$$

also,

$$P(pass(c1)) = P(pass(c1) \mid q^+(c1)) * P(q^+(c1)) + P(pass(c1) \mid q^-(c1)) * P(q^-(c1))$$

This equals: 0.8*0.7 + 0.35 * 0.3= 0.665

$$P(\neg pass(c1)) = P(\neg pass(c1) \mid q^+(c1)) * P(q^+(c1)) + P(\neg pass(c1) \mid q^-(c1)) * P(q^-(c1))$$

This equals: 0.2*0.7 + 0.65*0.3= 0.335

Using this information and Bayes' theorem in the way shown below, we can compute the 4 listed probabilities:

$$P(q^+(c1) \mid pass(c1)) = (P(pass(c1) \mid q^+(c1)) * P(q^+(c1)))/P(pass(c1))$$

, which equals (0.8*0.7)/(0.665) = 0.842105

$$P(q^{-}(c1) \mid pass(c1)) = (P(pass(c1) \mid q^{-}(c1)) * P(q^{-}(c1)))/P(pass(c1))$$

, which equals (0.35*0.3)/(0.665) = 0.157895

$$P(q^{+}(c1) \mid \neg pass(c1)) = (P(\neg pass(c1) \mid q^{+}(c1)) * P(q^{+}(c1)))/P(\neg pass(c1))$$

, which equals (0.2*0.7)/(0.335) = 0.417910

$$P(q^{-}(c1) \mid \neg pass(c1)) = (P(\neg pass(c1) \mid q^{-}(c1)) * P(q^{-}(c1)))/P(\neg pass(c1))$$

, which equals (0.65*0.3)/(0.335) = 0.582089

So, the probability that the car will be in good quality given that the test passes is 0.842105 and the probability that the car will be bad quality given that the test passes is 0.157895.

Furthermore, given that the test fails, the probability that the car is in good quality is 0.417910, while the probability that the car is in bad quality given the test fails is 0.582089.

Lastly, the probability that the car will simply pass the test is 0.665 while the probability that it will simply fail the test is 0.335.

C:

For this problem, I will assume that you have taken the car to the mechanic and have performed the test. I used the expected utility formula for new evidence (which the test is) to solve this problem.

The new evidence in this case can take on 2 possible forms: pass or fail.

If the test returns pass:

$$E(U)(A \mid e, E_j = pass) = max_A \sum_i U(Result_i(A)) * P(Result_i(A) \mid A, e, E_j = pass) \quad (16)$$

In this scenario, there are 2 possible actions given the test passes: buy the car, don't buy the car. A=[buy car, don't buy car]

Expected utility for buying the car given the test passes:

Since the car bought can be either good or bad quality, there are 2 possible results that can arise from this action.

$$P(q^+(c1) \mid pass(c1)) * U(q^+(c1)) + P(q^-(c1) \mid pass(c1)) * U(q^-(c1))$$

This equals 0.842105*(4000-3000) + 0.157895*(4000-3000-1400), which is 778.947

Expected utility for not buying the car given the test passes: 0

So, in the case that the test passes, the best action is to buy the car.

Expected utility for buying the car given the test fails:

Since the car bought can be either good or bad quality, there are 2 possible results that can arise from this action.

$$P(q^+(c1) \mid \neg pass(c1)) * U(q^+(c1)) + P(q^-(c1) \mid \neg pass(c1)) * U(q^-(c1))$$

This equals 0.417910*(4000-3000) + 0.582089*(4000-3000-1400), which is 185.0714

Expected utility for not buying the car given the test fails: 0

So, in the case that the test fails, the best action is to buy the car.

In conclusion, if you take the car to the mechanic, you should buy the car no matter what the test results show.

D:

The value of optimal information can be calculated by using a similar formula to the one used above.

We can use the formula:

$$VPI(E_j) = (\sum_k P(E_j = e_{jk} \mid E) * E(U)(a_{E_{jk}} \mid E, E_j = e_{jk})) - EU(a \mid E) \tag{17}$$

We already computed the EU(a $\mid E) in 6a, and it is equal to 580.$

The

$$(\sum_k P(E_j = e_{jk} \mid E) * E(U)(a_{E_{jk}} \mid E, E_j = e_{jk})) \tag{18}$$

part can be rewritten as:

$$P(E_j = pass) * E(U)(a_{E_{jk}} \mid E, E_j = pass) + P(E_j = fail) * E(U)(a_{E_{jk}} \mid E, E_j = fail) \quad (19)$$

, since the new evidence can take on 2 variables: pass or fail

We've already computed the expected utility values in this equation for when the car passes the test and when the car fails the test in question 6c.

$$E(U)(a_{E_{jk}} \mid E, E_j = pass) = 778.947 \quad (20)$$

and

$$E(U)(a_{E_{jk}} \mid E, E_j = fail) = 185.074 \quad (21)$$

We also know, from the calculations done in part B, that the probability of the test returning pass is 0.665 while the probability of it returning a fail 0.335.

Putting this all together,

$$VPI(E_j) = (0.665 * 778.947 + 0.335 * 185.0714) - 580 = -0.001 \approx 0 \quad (22)$$

Since we performed rounding in the calculations above and the value of optimal information cannot be negative according to the textbook, we can say that for this scenario, the value of optimal information is 0 dollars.

Since the value of optimal information in this case is 0, we should should not take c1 to the mechanic. This is because it costs 100 dollars to take the car to the mechanic in the first place. In order for this to be a worthwhile expense, the value of information gained from taking the car to the mechanic must be greater than the cost of obtaining that information. In this case, since the value of information gained is less than the cost of obtaining that information (0¡100), we should not take c1 to the mechanic.

Question 7

The Value Iteration implementation is in the file question_7_cs_440_assignment_2.py. The code is also provided in Appendix A.

In order to create the Markov Decision Process Model in Python along with it's transitional probabilities list, we represented each state and actions as it's own integer and stored it

into an array. We have 4 actions; however, not all those 4 actions are possible to do in every state, so we created an adjacency list that stored each state and it's relevant actions. The transitional probabilities were stored as a dictionary and can be accessed via the transition_model(s,a,dest) function. The discount factor will be 0.9 for the purposes of this program. We will also use a epsilon value of 0.1 as our threshold because this is a very standard but still small enough value. The Value Iteration implementation works by using a while loop that is controlled by a sole true statement. It will break when the the current V values are very close to the previous V values via a threshold we establish. We create a for loop for states and within that, a for loop of actions. Within that for loop for a, we calculate the R(s,a) + $\lambda \sum_{s' \in S} T(s, a, s')V_k(s')$.

After that we update pi and V via np argmax and max respectively. We print the iteration number, the V, and the pi : the iteration we are on, the intermediate optimum utilities, and intermediate policies. We finally check to see whether or not we should break based on our epsilon threshold value. The program concludes by printing out the computation time from when it starting the clock at the beginning of the program.

The optimum utilities were respectively [3.5532675111347767, 4.001178449540639, 4.660801971401323, 4.1413297939752045].
The optimal policy was [2, 2, 3, 1]. This means that State 1 should choose action 2, State 2 should choose action 2, State 3 should choose action 3, and State 4 should should choose action 1.
The computation time came out to 0.012331724166870117 seconds.
The number of iterations it took to converge based on our criteria was 13 iterations. The number of iterations will chance depending the discount factor and the value of epsilon.
Here is the output with the iteration , the intermediate optimum utilities, and intermediate optimal policies:

```
Iteration: 0
V: [0.0, 0.0, 1.0, 0.81]
pi: [1, 1, 1, 1]
Iteration: 1
V: [0.5832000000000002, 0.7200000000000001, 1.729, 1.47339]
pi: [2, 2, 3, 1]
Iteration: 2
V: [1.1658168, 1.3744800000000001, 2.326051, 2.0167064100000003]
pi: [2, 2, 3, 1]
Iteration: 3
V: [1.6618756392000003, 1.9221631200000002, 2.8150357690000005, 2.461682549790001]
pi: [2, 2, 3, 1]
Iteration: 4
V: [2.071549050904801, 2.3728151152800003, 3.2155142948110007, 2.826118008278011]
pi: [2, 2, 3, 1]
Iteration: 5
V: [2.4076837951230323, 2.742277013014321, 3.5435062074502097, 3.124590648779691]
pi: [2, 2, 3, 1]
```

```
Iteration: 6
V: [2.683088350243523, 3.044934331706729, 3.812131583901722, 3.369039741350567]
pi: [2, 2, 3, 1]
Iteration: 7
V: [2.9086645168162426, 3.2928229201164516, 4.032135767215511, 3.569243548166115]
pi: [2, 2, 3, 1]
Iteration: 8
V: [3.0934149677065266, 3.4958458780161292, 4.212319193349503, 3.7332104659480487]
pi: [2, 2, 3, 1]
Iteration: 9
V: [3.2447262296697703, 3.662122077254546, 4.3598894193532445, 3.8674993716114527]
pi: [2, 2, 3, 1]
Iteration: 10
V: [3.368650268900805, 3.7983023558401543, 4.480749434450308, 3.9774819853497805]
pi: [2, 2, 3, 1]
Iteration: 11
V: [3.470144077853987, 3.90983401685545, 4.579733786814803, 4.06755774600147]
pi: [2, 2, 3, 1]
Iteration: 12
V: [3.5532675111347767, 4.001178449540639, 4.660801971401323, 4.1413297939752045]
pi: [2, 2, 3, 1]

Computation Time: 0.012331724166870117
```

Appendix A - Value Iteration Implementation

```python
import random
import numpy as np
import time

start = time.time()


# Create the Markov Decision Process model
states = [1,2,3,4]
rewards = [0,0,1,0]
discountFactor = 0.9
actions = [1,2,3,4]

adj_list = {
    1: [1,2,4],
    2: [2,3,1],
    3: [2,4],
    4: [4,1,3]
}

# Usage: V[state-1]
V = [0,0,0,0]
prev_V = []
# to use: policy[state-1]
policy = []
pi = [None, None, None, None]

def transition_model(s,a,dest):
  # Transitional Probabilities
  # usage: transition_model[(state,action,destination_state)] = probability
```

```
  tm = {
      (1,1,1): 0.2,
      (1,1,2): 0.8,
      (1,2,1): 0.2,
      (1,2,4): 0.8,
      (2,2,2): 0.2,
      (2,2,3): 0.8,
      (2,3,2): 0.2,
      (2,3,1): 0.8,
      (3,4,2): 1,
      (3,3,4): 1,
      (4,1,4): 0.1,
      (4,1,3): 0.9,
      (4,4,4): 0.2,
      (4,4,1): 0.8
  }
  if (s,a,dest) in tm:
    return tm[(s,a,dest)]
  else:
    return 0

def rand_policy():
  policy[0] = random.randint(1,2)
  policy[1] = random.randint(2,3)
  policy[2] = random.randint(4,3)
  policy[3] = random.randint(1,4)

# Value Iteration Implementation
def value_iteration():
  #find optimal utility
  iteration_count = 0;
  while True:
    prev_V = V.copy()
    for s in states:
      action_vals = []
      for a in actions:
        state_sum = 0
        for n in adj_list[s]:
          state_sum += transition_model(s,a,n) * V[n-1]
        action_vals.append(rewards[s-1] + discountFactor*state_sum)
      pi[s-1] = actions[np.argmax(action_vals)]
      V[s-1] = max(action_vals)


    # Print Iteration Number
    print("Iteration: " + str(iteration_count));
    iteration_count = iteration_count + 1;

    # Print V
    print(V)

    #Print pi
    print(pi)


    # Checks to see whether the while loop should break
    if not check(0.1,V,prev_V):
      break
```

```
89
90
91  def check(epsilon,val,val_prev):
92    if len(val_prev) == 0:
93      return True
94    for s in range(0,len(val)):
95      diff = val[s] - val_prev[s]
96      if  diff > epsilon:
97        # print("change detected")
98        return True
99    return False
100
101 # Execute Value Iteration Implementation
102 value_iteration()
103
104 # Calculate Computation Time
105 end = time.time()
106 computation_time = end - start
107
108 # Prints Computation Time
109 print("\nComputation Time: "+ str(computation_time))
```