Question 1)

A

This travel data set can be used to estimate probabilities with three sampling methods: direct sampling, rejection sampling, and Gibbs sampling. All of these have advantages and disadvantages, which depend on the nature of the model and the interrelation between the variables involved.

The direct sampling method is the easiest of the three methods proposed here and generates samples directly from a known distribution. This method works well if the of interest are simple and easy to sample from. Its strengths are simplicity and accuracy, as it provides unbiased samples directly from the desired distribution without needing complex kinds of modification. However, for complicated or multivariate distributions that are not easily decomposed or sampled directly, this has limitations. For the purposes of this dataset, direct sampling would be applicable if the joint distributions (such as of air travel and business travel) are straightforwardly easy to compute. Nevertheless, direct sampling's feasibility decreases with increased complexity.

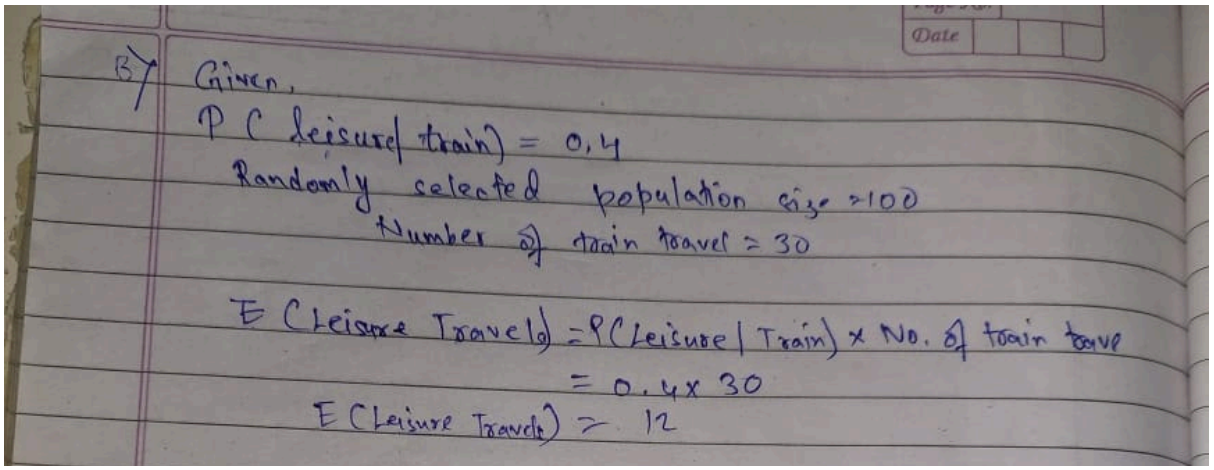
Rejection Sampling is another more flexible method, wherein samples are drawn from a proposal distribution and accepted only if they meet some condition directly related to the target distribution. It is useful when direct sampling is not possible. With complex distributions for example, this method is very useful because it is highly flexible in choosing a proposal distribution, yet typically horribly inefficient when its acceptance rate is low and there are lots of discarded samples. Rejection sampling can be useful for establishing probabilities connected with rare events, such as bus travel under low stress levels, for the travel dataset. However, its inefficiency makes it less suitable when large amounts of samples are required.

Gibbs Sampling is a method within the family of Markov Chain Monte Carlo methods designed to handle complex, multivariate distributions. By iteratively taking samples from the conditional distribution of every one of the variables conditional on the others, it can be very advantageous when dealing with highly dimensional spaces where direct sampling or rejection sampling would prove inordinately expensive. The biggest advantage of Gibbs sampling is its efficiency for large models, especially when the joint distribution is hard to sample directly. It does not even demand the normalization constant of the joint distribution. On the other hand, convergence could sometimes be slow, especially if strong correlations between variables are present, and the method may involve a burn-in period, discarding the early samples to allow the chain to reach an equilibrium distribution. For example, in the travel data set, there exist interdependencies, like travel preferences (air, train, bus) and reasons for travel (business, leisure). Gibbs sampling is very apt in this type of task given the complex conditional probabilities existing in this dataset, such as the probability of someone traveling for leisure given his mode of transport.

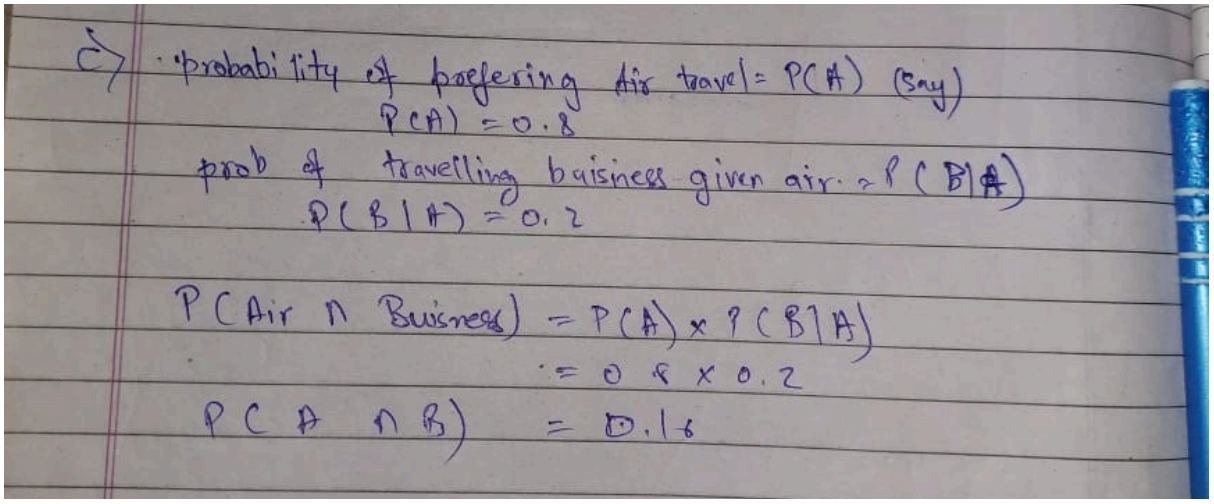
In summary, Gibbs sampling emerges here as the best algorithm to handle this dataset. It can indeed manage the complex interrelated distributions without having an overwhelming number of dependents. Direct sampling might be applicable for some simpler cases but

becomes highly infeasible when the dependencies become really complicated. Rejection sampling is flexible but may be quite inefficient for the computation of probabilities for rare events. Therefore, when it comes to the travel dataset provided, Gibbs sampling is the only choice because it makes a reasonable trade-off between flexibility in handling the complex interactions between travel preferences, stress level, and reasons for traveling and efficiency in processing them.

B)



C)



D)

Increasing the sample size in **direct sampling** significantly impacts both the **accuracy** and **precision** of the estimates obtained, and this has important implications for analyzing the given travel dataset.

Accuracy refers to how closely an estimate matches the true population value, and **precision** relates to the consistency or repeatability of the estimates across different samples. As the sample size increases, the **accuracy** of the estimates improves because,

according to the **Law of Large Numbers**, larger samples provide more information, reducing random sampling error and bringing the sample mean closer to the true population parameter. In the context of the travel dataset, increasing the sample size would improve the accuracy of estimates for probabilities such as **air travel for business** or **train travel for leisure**, making them more reflective of the true population distributions.

In terms of **precision**, a larger sample size reduces the **standard error**, which means there is less variability between repeated sample estimates. As a result, the estimates become more stable and reliable. For example, rare events like **low-stress bus travel**, with a probability of just 0.015, can be better estimated with a larger sample, as more observations allow for a more precise calculation of such rare occurrences. Larger sample sizes also help clarify the relationships between various factors in the dataset, such as how **stress levels** are associated with different travel preferences or how **business** and **leisure** travel affect mode of transport choices.

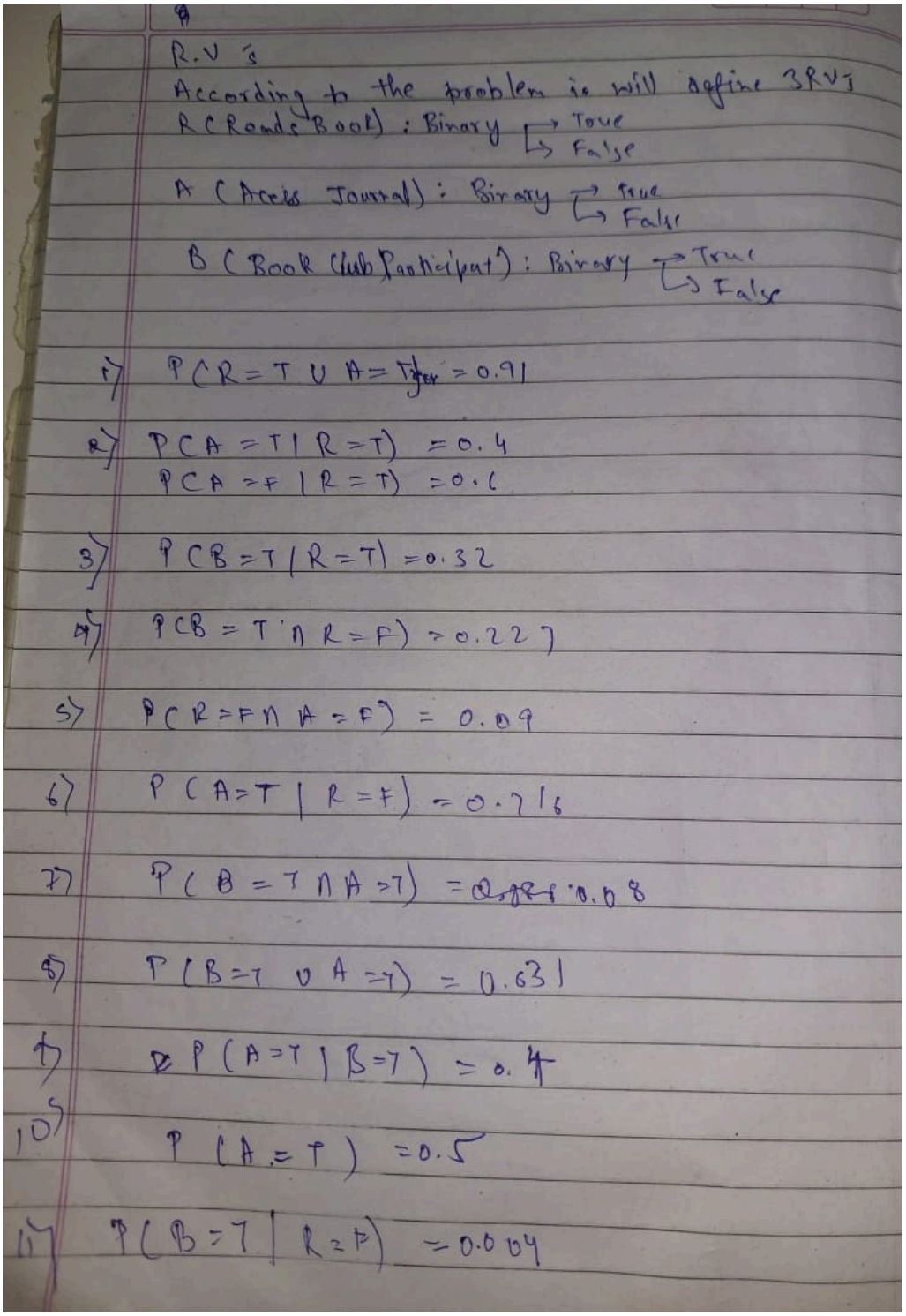
For the given travel dataset, which includes complex interdependencies like the relationship between **travel preferences** (air, train, bus), **stress levels**, and **reasons for travel** (business or leisure), a larger sample size would provide more reliable probability estimates. With more data, conditional probabilities, such as the likelihood of someone traveling for **leisure** given they prefer **train travel**, would become more accurate. Similarly, rare events, such as **low-stress bus travel**, would be better represented in the sample, improving the reliability of their probability estimates.

However, there are practical considerations when increasing sample size. **Data collection** for larger samples requires more resources, time, and effort, which can increase the **cost** of the analysis. Additionally, while larger sample sizes generally improve accuracy and precision, there's a risk of **overfitting**, especially if the model becomes too complex or if the sample size is excessively large relative to the underlying population structure. This can lead to estimates that are too specific and not generalizable.

In conclusion, increasing the sample size in **direct sampling** improves both **accuracy** and **precision**, which is especially beneficial for complex datasets like the one provided. It allows for more reliable and consistent estimates of conditional probabilities and helps address rare events. However, practical challenges, such as the **cost** of data collection and the potential for **overfitting**, should be carefully managed. Ultimately, a larger sample size leads to more accurate and precise probability estimates, enhancing the quality of insights drawn from the travel dataset.

Question 2>

A>



R.V's

According to the problem is will define 3 R.V's

R (Reads Book) : Binary → True / False

A (Access Journal) : Binary → True / False

B (Book Club Participant) : Binary → True / False

1) $P(R = T \cup A = T) = 0.91$

2) $P(A = T \mid R = T) = 0.4$
$P(A = F \mid R = T) = 0.6$

3) $P(B = T \mid R = T) = 0.32$

4) $P(B = T \cap R = F) = 0.227$

5) $P(R = F \cap A = F) = 0.09$

6) $P(A = T \mid R = F) = 0.716$

7) $P(B = T \cap A = T) = 0.08$

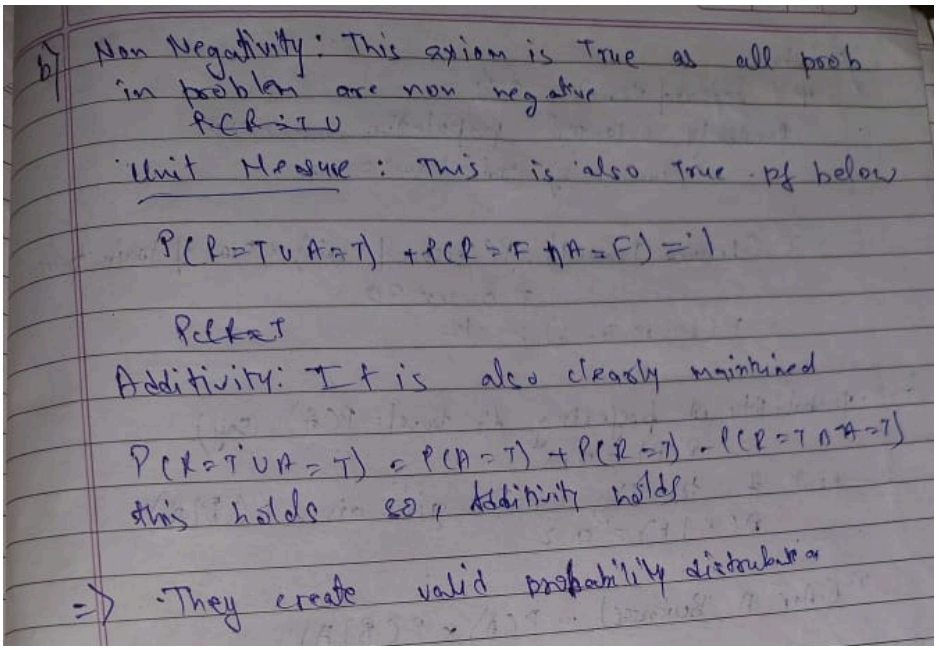
8) $P(B = T \cup A = T) = 0.631$

9) $P(A = T \mid B = T) = 0.4$

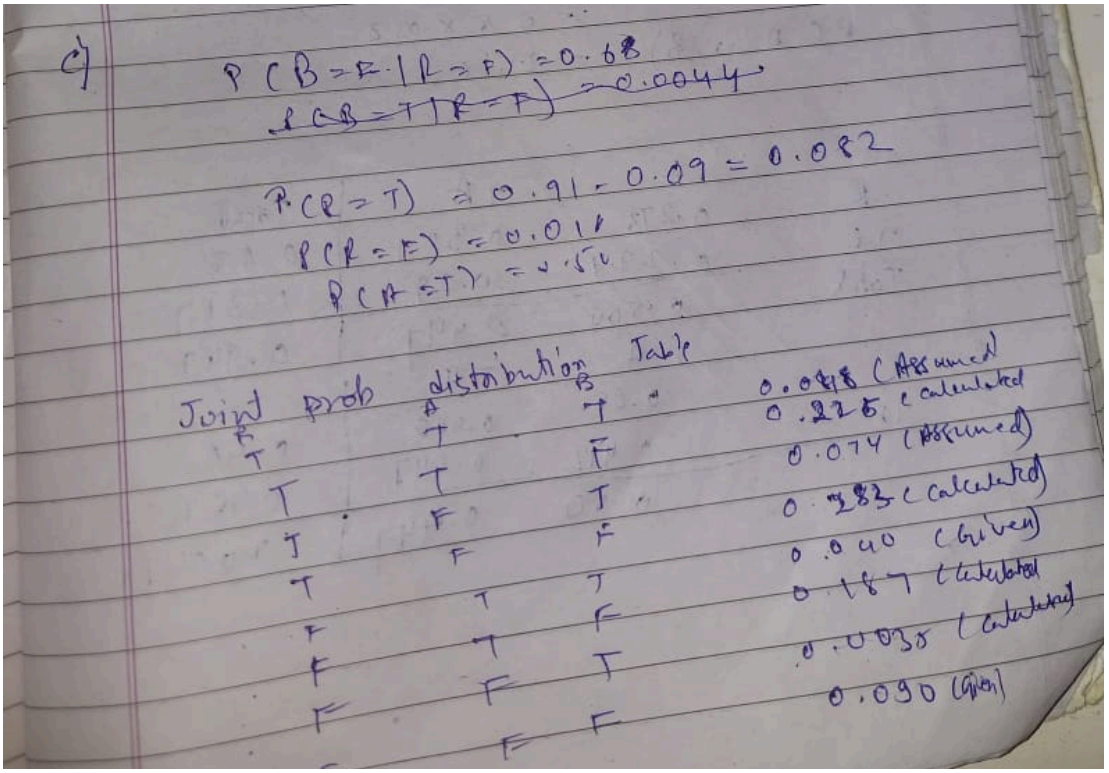
10) $P(A = T) = 0.5$

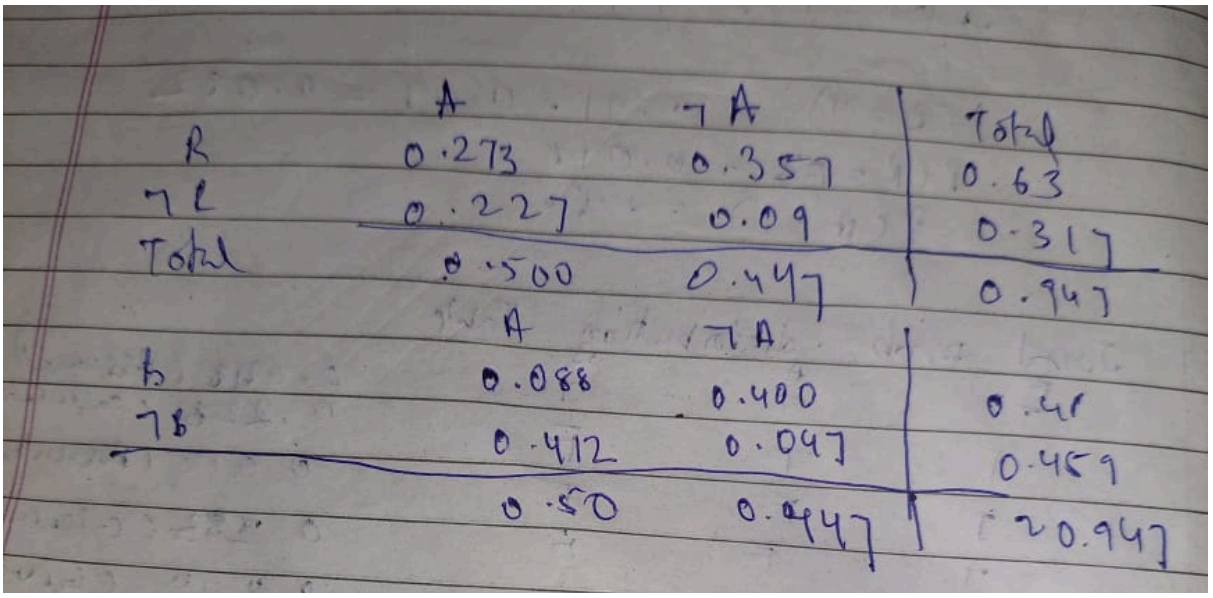
11) $P(B = T \mid R = F) = 0.004$

B>


b) Non Negativity: This axiom is True as all prob in problem are non negative
Unit Measure : This is also True pf below
P(R=T ∪ A=T) + P(R=F ∩ A=F) = 1
Additivity: It is also clearly maintained
P(R=T ∪ A=T) = P(A=T) + P(R=T) - P(R=T ∩ A=T)
this holds so, Additivity holds
=> They create valid probability distribution

C>

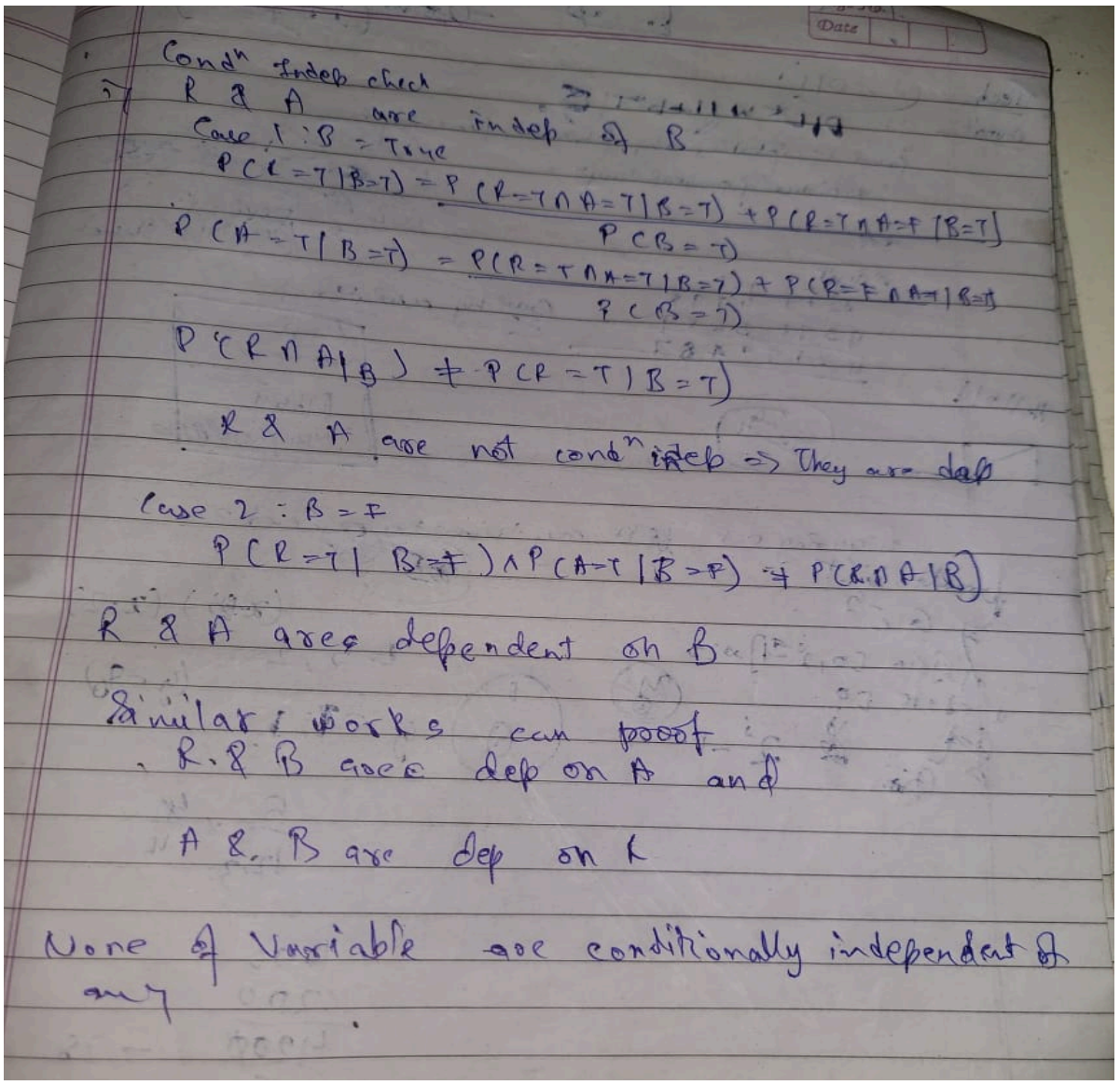

c)
P(B=F | R=F) = 0.68
P(R=T) = 0.91 - 0.09 = 0.082
P(R=F) = 0.018
P(A=T) = 0.15
Joint prob distribution Table
R A B
T T T 0.048 (Assumed)
T T F 0.225 (calculated)
T F T 0.074 (Assumed)
T F F 0.283 (calculated)
F T T 0.040 (Given)
F T F 0.187 (calculated)
F F T 0.0035 (calculated)
F F F 0.090 (Given)



	A	¬A	Total
B	0.273	0.357	0.63
¬B	0.227	0.09	0.317
Total	0.500	0.447	0.947

	A	¬A	
b	0.088	0.400	0.48
¬b	0.412	0.047	0.459
	0.50	0.447	20.947

D>



Also as Either A∩B and C∩B C∩A not equal to 0 so conditional depednce can be proved from here also

Q3>

A)

To formulate the problem using Bayesian inference, we first define the key events involved. Let:

- **A** represent the event where the misclassification was caused by adversarial perturbations.
- **B** represent the event where the misclassification was caused by a backdoor attack.
- **M** represent the event of a misclassification alarm being raised.

Initially, adversarial perturbations and backdoor attacks are considered independent events, meaning that the occurrence of one does not affect the likelihood of the other. However, we are presented with new information that backdoor attacks, specifically the presence of backdoor triggers, have become more prevalent in recent datasets. This updated information impacts our prior beliefs about the likelihood of adversarial perturbations causing the misclassification alarm, prompting us to adjust our prior probabilities.

Bayes' Theorem provides a systematic way to update our beliefs based on this new evidence. The formula for Bayesian updating is:

$$P(A \mid M) = P(M \mid A) \cdot P(A) / P(M)$$

Where:

- **P(A | M)** is the updated probability that the misclassification was caused by adversarial perturbations, given that a misclassification alarm was raised.
- **P(M | A)** is the probability of raising a misclassification alarm given that adversarial perturbations occurred.
- **P(A)** is the prior probability of adversarial perturbations causing a misclassification.
- **P(M)** is the total probability of a misclassification alarm, which accounts for all possible causes of the misclassification. Using the law of total probability, this can be written as:

$$P(M) = P(M \mid A) \cdot P(A) + P(M \mid B) \cdot P(B)$$

The key piece of new information here is that backdoor triggers are now more prevalent, meaning the prior probability **P(B)** (the probability of a backdoor attack causing a misclassification) has increased. As a result, the prior probability of adversarial perturbations, **P(A)**, must decrease to maintain the total probability balance. This adjustment impacts the total probability **P(M)**, as well as the updated belief **P(A | M)**.

In other words, as the prevalence of backdoor attacks increases, the likelihood of **B** (backdoor attack) contributing to the misclassification alarm rises, which makes **A** (adversarial perturbations) less likely as the cause of the misclassification. Hence, after incorporating the new information, we may find that the probability of adversarial perturbations causing the misclassification alarm has decreased. This reflects how Bayesian inference helps us dynamically update our beliefs in response to new, relevant data.

Therefore, the increase in **P(B)** due to the rising prevalence of backdoor triggers shifts our belief about the cause of misclassifications. Bayesian inference allows us to quantitatively assess the impact of this new information and adjust our probabilities accordingly.

B>

Priors:

- **P(A)** represents the prior probability that a misclassification alarm is due to adversarial perturbations. This is our initial belief about how likely adversarial perturbations are to cause a misclassification before considering any new evidence.
- **P(B)** represents the prior probability that a misclassification alarm is due to a backdoor attack. Initially, we assume that adversarial perturbations and backdoor attacks are independent, so **P(A)** and **P(B)** are considered separate prior probabilities.

Likelihoods:

- **P(M | A)** is the likelihood of observing a misclassification alarm given that the cause is adversarial perturbations. This represents how likely it is for a misclassification alarm to be triggered when adversarial perturbations are at play.
- **P(M | B)** is the likelihood of observing a misclassification alarm given that the cause is a backdoor attack. This reflects how likely a misclassification alarm is to occur when a backdoor trigger is responsible.
- **P(M | A, B)** is the likelihood of a misclassification alarm being raised if both adversarial perturbations and a backdoor attack are present simultaneously. This accounts for the scenario where both types of attacks could occur together.

Posterior:

- **P(A | M)** represents the updated belief about adversarial perturbations causing the misclassification, given that a misclassification alarm has been observed. This is the posterior probability that we seek to compute, which is updated based on the evidence (the observed misclassification alarm).
- **P(B | M)** represents the updated belief about a backdoor attack causing the misclassification, given the same misclassification alarm.

To compute the total probability of a misclassification alarm, we use the law of total probability:

$$P(M)=P(M|A)\cdot P(A)+P(M|B)\cdot P(B)-P(M|A,B)\cdot P(A\cap B)$$

Finally, applying Bayes' theorem, we update our belief about the likelihood of adversarial perturbations causing the misclassification:

$$P(A \mid M) = P(M \mid A) \cdot P(A) / P(M)$$

This framework allows us to update our understanding based on new evidence. Specifically, the increase in the prevalence of backdoor attacks (i.e., the rise in **P(B)**) will influence the posterior probability **P(A | M)**. As backdoor triggers become more common, **P(A)** (the prior probability of adversarial perturbations) may decrease, which will affect our belief about the likelihood of adversarial perturbations causing the observed misclassification alarm. By using Bayesian inference, we can systematically adjust our probabilities to account for this new information.

C>

Conditioning on the detection of backdoor attacks, especially given recent reports indicating their increased prevalence, significantly alters our belief about the role of adversarial perturbations in causing a misclassification. Initially, adversarial perturbations and backdoor attacks are treated as independent events, each capable of causing a misclassification alarm. In this scenario, we consider the prior probabilities of adversarial perturbations (**P(A)**) and backdoor attacks (**P(B)**) based on our initial understanding of the threat landscape.

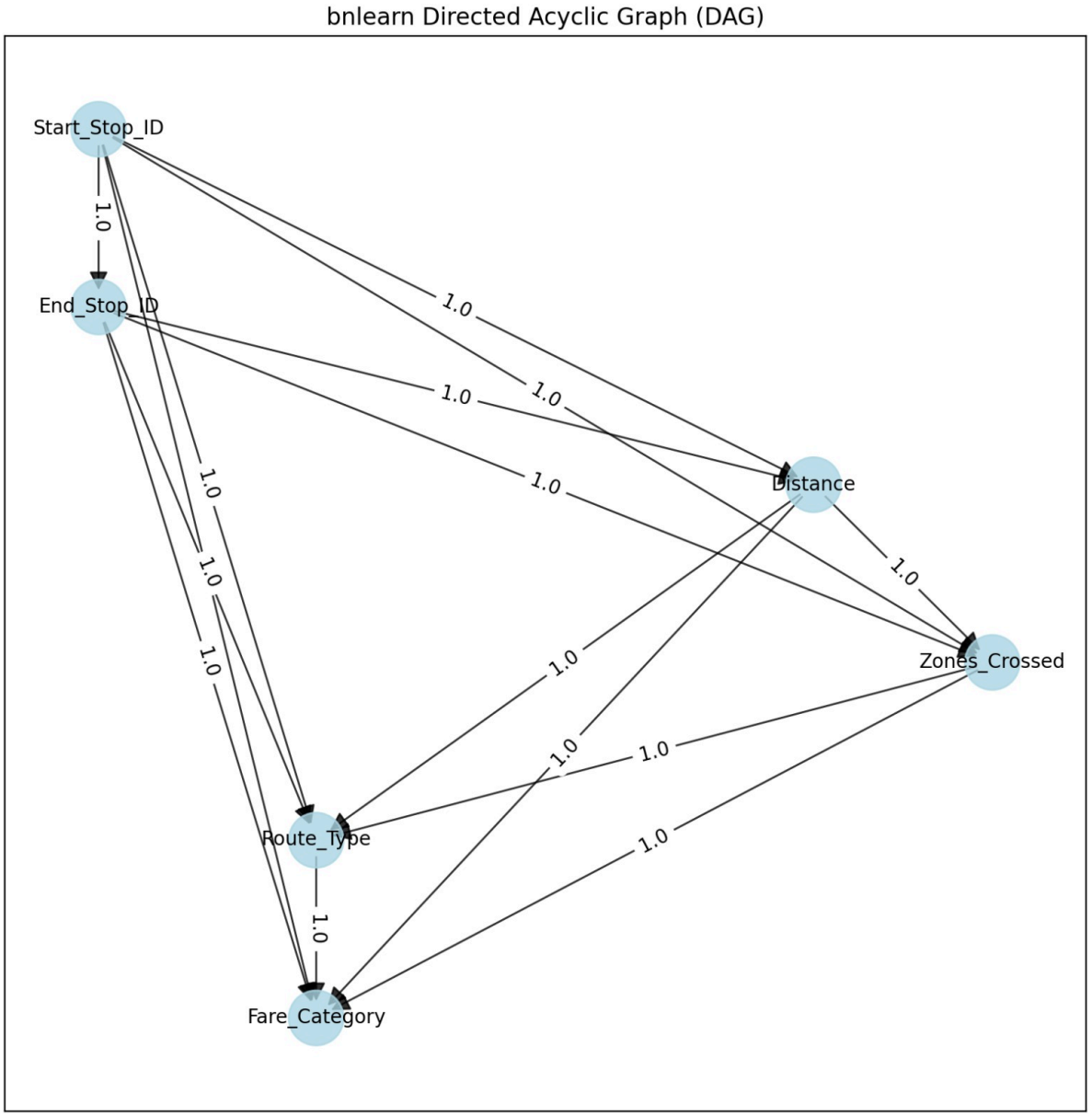
However, the new information that backdoor triggers have become more common leads to an increase in **P(B)**, the prior probability of a backdoor attack. This change affects the total probability of a misclassification alarm (**P(M)**), making it more likely that the misclassification is due to a backdoor attack. As a result, the probability of adversarial perturbations causing the misclassification alarm (**P(A | M)**) decreases. This phenomenon is known as "explaining away"—as we gain more evidence supporting the likelihood of backdoor attacks, the role of adversarial perturbations in causing the alarm becomes less significant, even though adversarial perturbations still contribute to **P(M)**.

In Bayesian terms, as **P(B)** increases, **P(M)** also increases because more misclassification alarms can now be attributed to backdoor triggers. The total probability of a misclassification alarm is updated using the law of total probability, accounting for both adversarial perturbations and backdoor attacks. Consequently, the posterior probability **P(A | M)**, which represents the likelihood that adversarial perturbations caused the misclassification, is reduced, as more of the misclassification alarms are attributed to backdoor attacks.

In essence, the prevalence of backdoor attacks shifts the distribution of probabilities, making it more likely that a misclassification alarm is due to backdoor attacks rather than adversarial perturbations. This Bayesian update reflects how new evidence about the increased prevalence of backdoor attacks leads to a decrease in our belief that adversarial perturbations are responsible for the misclassification, ultimately altering our understanding of the model's security vulnerabilities.

A

The `make_network` function defines and fits a Bayesian Network to the input DataFrame (`df`). It first extracts the feature names and creates pairwise edges between them to form a Directed Acyclic Graph (DAG). The function then uses the `bn.parameter_learning.fit` method to learn the model parameters based on the provided data, and it visualizes the network using `bn.plot`. Finally, the fitted Bayesian Network model is returned.




```
Total Test Cases: 350
Total Correct Predictions: 350 out of 350
Model accuracy on filtered test cases: 100.00%

Network Construction: 912.71478510 seconds
```

Pkl file size:194.5 MB

B

Pruning Method Explanation

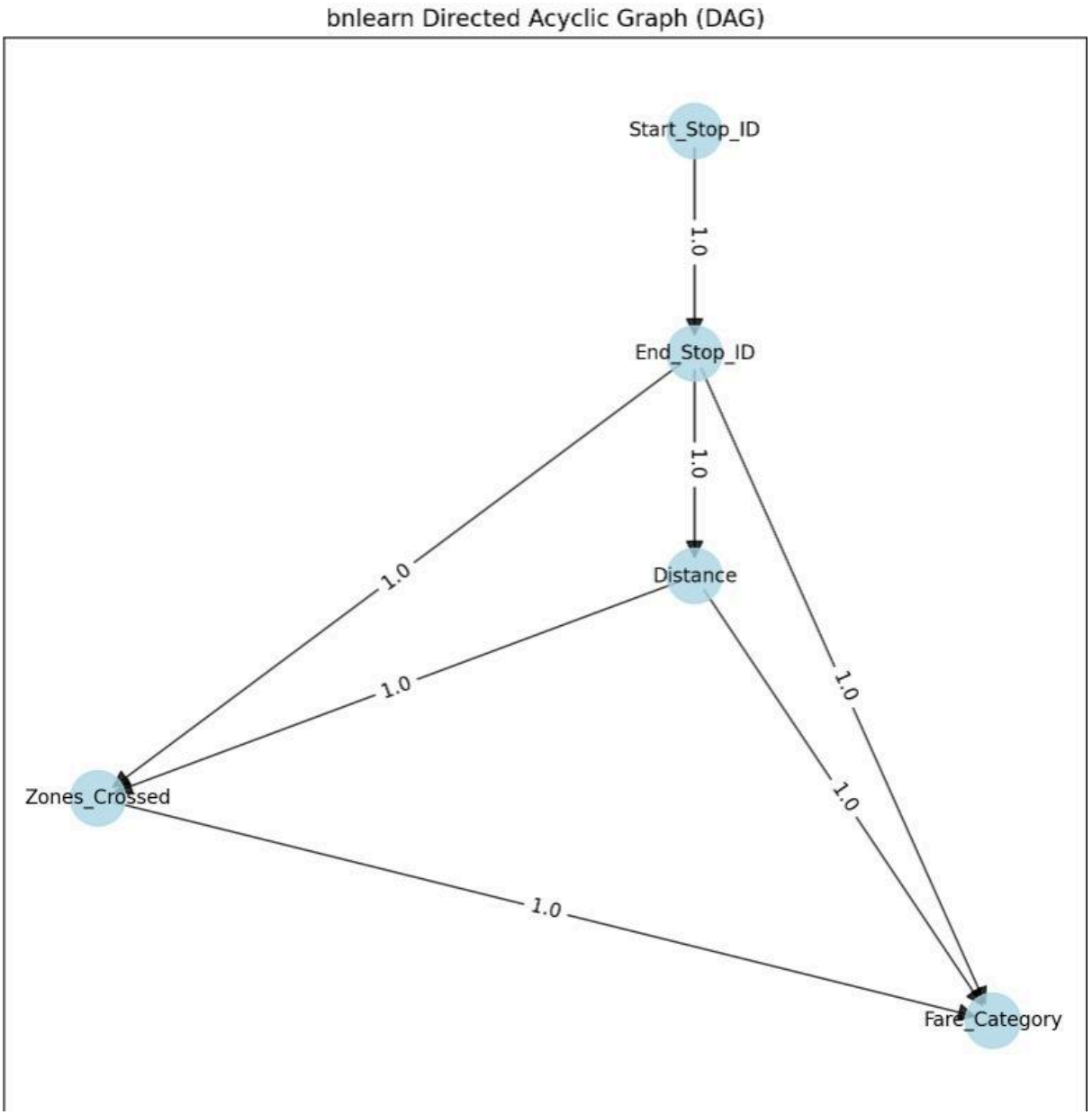
The pruning method in the `make_pruned_network` function aims to improve the **efficiency** and **prediction accuracy** of the Bayesian Network by eliminating weak edges based on the correlation between features in the input DataFrame (`df`). The function first calculates the **correlation matrix**, which measures the linear relationship between each pair of features. For each edge in the network (a pair of features), the absolute correlation value is checked. If the correlation is **below 0.2**, the edge is considered weak and is **pruned** (removed) from the network. Only edges with a correlation of **0.2 or higher** are retained, indicating a strong relationship. The function then constructs the **pruned Directed Acyclic Graph (DAG)**, fits the model using **parameter learning**, and visualizes the final network.

Benefits of Pruning

Pruning offers two key benefits: **efficiency** and **accuracy**.

1. **Improved Efficiency**: By removing weak edges, the model becomes **simpler** and **less complex**, reducing the number of parameters to be learned. This makes the model **faster to fit**, saving **computation time** and resources, particularly in cases with large datasets or high-dimensional data.
2. **Improved Accuracy**: Pruning enhances **predictive accuracy** by focusing on the most **relevant relationships** between features. Weak or irrelevant edges are removed, which helps reduce **noise** in the model. The remaining stronger edges are more likely to reflect true dependencies, resulting in a **more reliable and robust model** that can make better predictions.

In summary, the pruning method helps create a **faster** and **more accurate** Bayesian Network model by focusing on strong correlations and reducing unnecessary complexity.



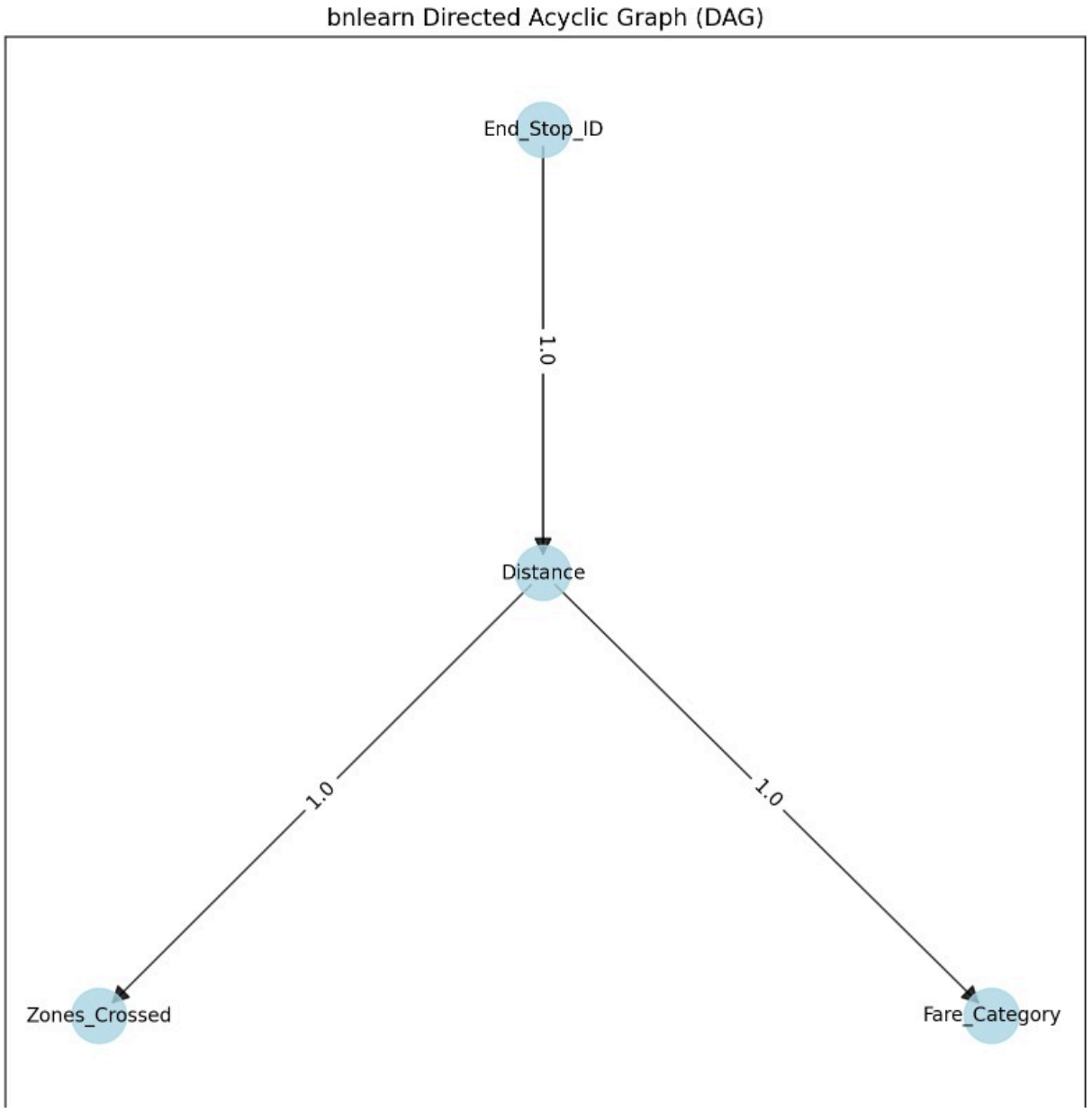

```
Total Test Cases: 350
Total Correct Predictions: 350 out of 350
Model accuracy on filtered test cases: 100.00%

Pruned Construction: 12.95511651 seconds
```

Pkl file size : 1.6 MB
C>

The `make_optimized_network` function improves the **accuracy** and **efficiency** of the Bayesian Network through **structure optimization**. Initially, the function creates a set of edges between all pairs of features in the dataset, forming an unoptimized Directed Acyclic Graph (DAG). It then applies **structure learning** using the **Hill Climbing (hc)** method, which iteratively searches for the best network structure based on the data. This process refines

the network by considering only the predefined edges in the initial DAG and selecting the most suitable structure that best represents the relationships between features. Once the structure is optimized, the model's parameters are learned, and the network is visualized.

The optimization improves **accuracy** by ensuring the network captures true dependencies between variables. The Hill Climbing method helps avoid overfitting and underfitting by selecting the best-fitting structure for the data, leading to a model that is more accurate in its predictions. Additionally, the optimization enhances **efficiency** by limiting the search space to a more manageable set of network structures, which reduces computational complexity. The refined structure allows the model to learn more effectively, requiring fewer iterations and less computational power, thereby speeding up the training process. Overall, the optimization leads to a faster, more accurate Bayesian Network that is better suited for real-world applications.



```
Total Test Cases: 350
Total Correct Predictions: 350 out of 350
Model accuracy on filtered test cases: 100.00%
```

```
Optimized Construction: 0.59016681 seconds
```

Pkl file size: 21.5 Kb

Explanation of State Transitions, Emissions, and the Viterbi Algorithm

1. Emission Probability

The emission probability using a Gaussian distribution can be written as:

P(observation | state) = (1 / sqrt(2 * pi * sigma^2)) * exp(- ((obs_x - true_x)^2 + (obs_y - true_y)^2) / (2 * sigma^2))

Where:

- sigma is the standard deviation of the Gaussian distribution.

For numerical stability, the log of the emission probability is used:

log P(observation | state) = - ((obs_x - true_x)^2 + (obs_y - true_y)^2) / (2 * sigma^2) - log(sqrt(2 * pi * sigma^2))

2. Transition Probability

The transition probability represents the likelihood of moving from one state to another.

- **For the "straight_until_obstacle" policy**, the transition is in a straight line unless an obstacle is encountered, causing a random direction change.
- **For the "random_walk" policy**, the Roomba can transition to any of the four possible headings with equal probability.

The transition probability from state s(t-1) to s(t) is given by:

P(transition | s(t-1), s(t)) = log(1 / number_of_possible_states_from_s(t-1))

If a transition is not possible (e.g., moving into an obstacle), the probability is zero, or in log form, it is -∞.

3. Viterbi Algorithm

Initialization

At time step 1, the probability for each possible state is initialized as:

V_1(state) = log(P(emission | state)) + log(P(transition from start state to state))

Recursion

For each subsequent time step t, the most likely state s_t is calculated by considering all possible previous states s(t-1):

V_t(s_t) = max_s(t-1) [V_(t-1)(s_(t-1)) + log(P(transition | s(t-1), s_t)) + log(P(emission | s_t))]

Where:

- V_(t-1)(s_(t-1)) is the probability of the previous state.
- log(P(transition | s(t-1), s_t)) is the log of the transition probability.
- log(P(emission | s_t)) is the log of the emission probability.

Backtracking

Once the final time step is processed, the most likely sequence of states is found by backtracking from the state with the highest probability:

Most likely state sequence = argmax_s_t [V_t(s_t)]

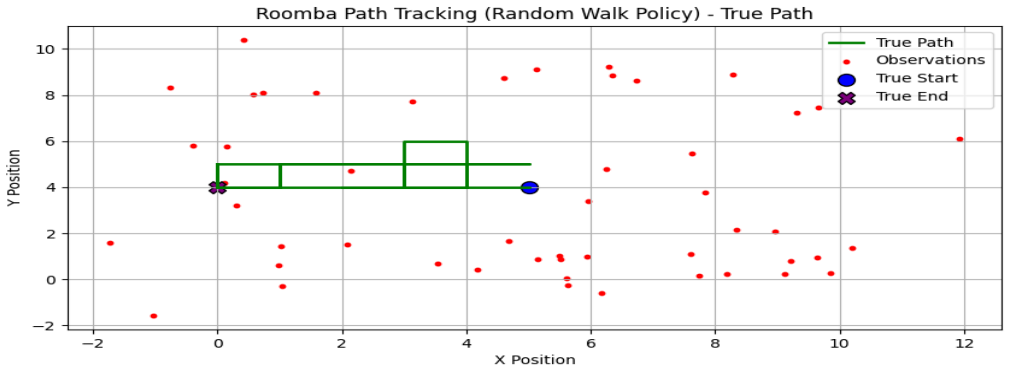
This process is performed for each time step to reconstruct the most probable sequence of states.

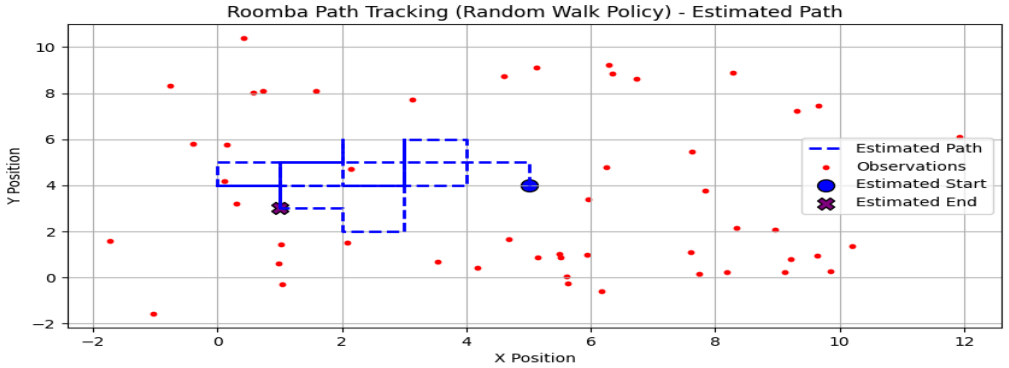
Q) Analyze which policy is more accurate and why.

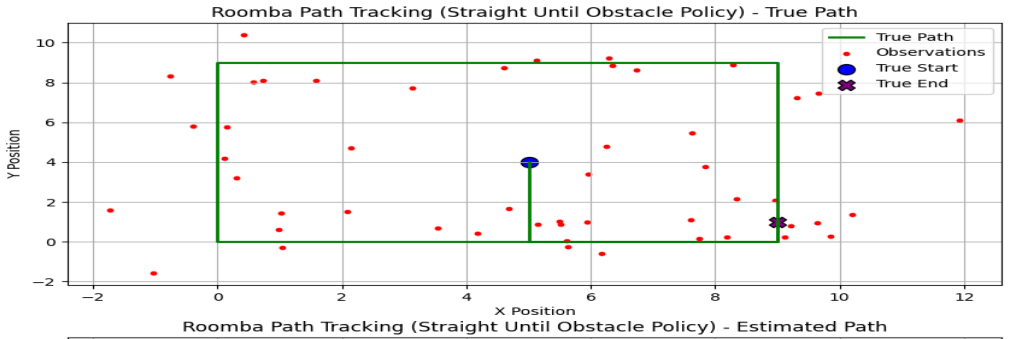
The **"straight_until_obstacle"** policy is more accurate because it offers predictable and deterministic movement. The Roomba moves in a straight line until it encounters an obstacle, which makes it easier to track its position using the Viterbi algorithm. Errors in position accumulate slowly and predictably, allowing for more accurate localization. In contrast, the **"random_walk"** policy introduces more uncertainty due to its random direction changes. This randomness increases the number of possible states, making it harder to estimate the Roomba's position accurately. The Viterbi algorithm struggles with the increased unpredictability and noisy observations in the random walk policy, resulting in less accurate localization.

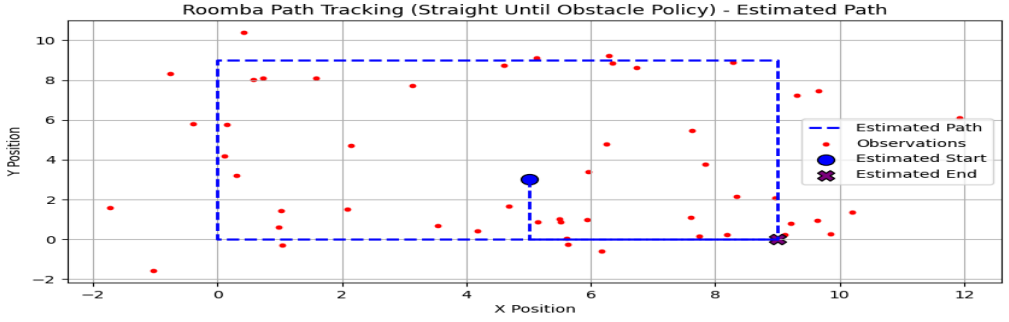
Error Handling:

The code implements exception handling in several key areas to ensure smooth operation. In the `transition_probability` function, it raises a `ValueError` if an invalid movement policy is passed. The `viterbi` function handles log-zero errors gracefully by returning `-inf` for impossible transitions. Additionally, the `is_obstacle` function checks for out-of-bounds positions, preventing invalid state transitions. Though there are no explicit exception handling mechanisms for array index bounds, the code carefully ensures valid indices are used, preventing out-of-bounds errors. Overall, these measures enhance the robustness and stability of the program.

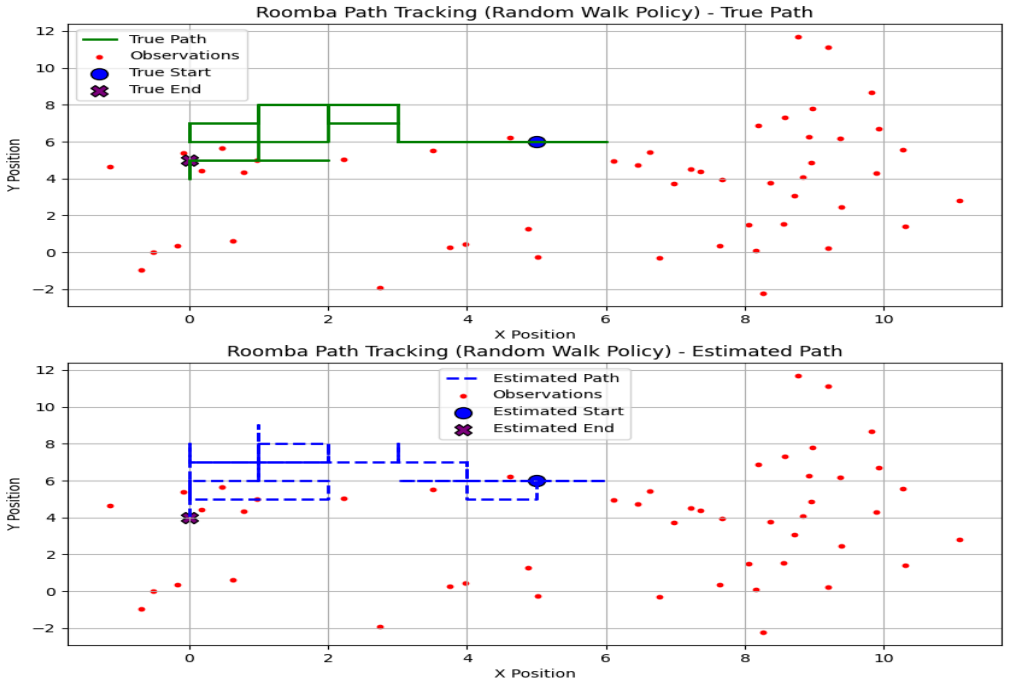


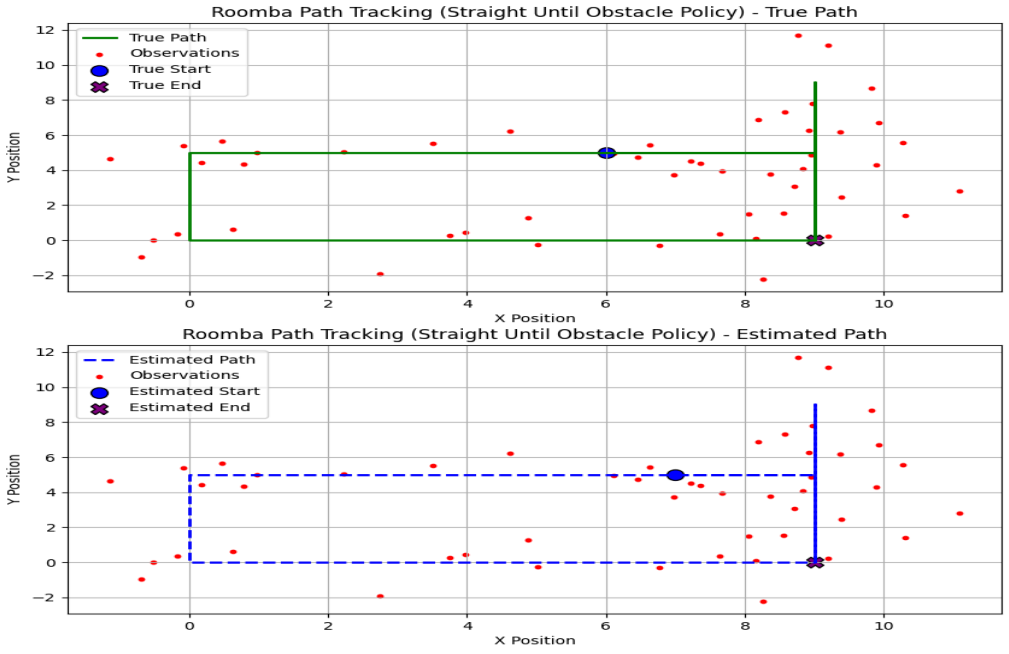




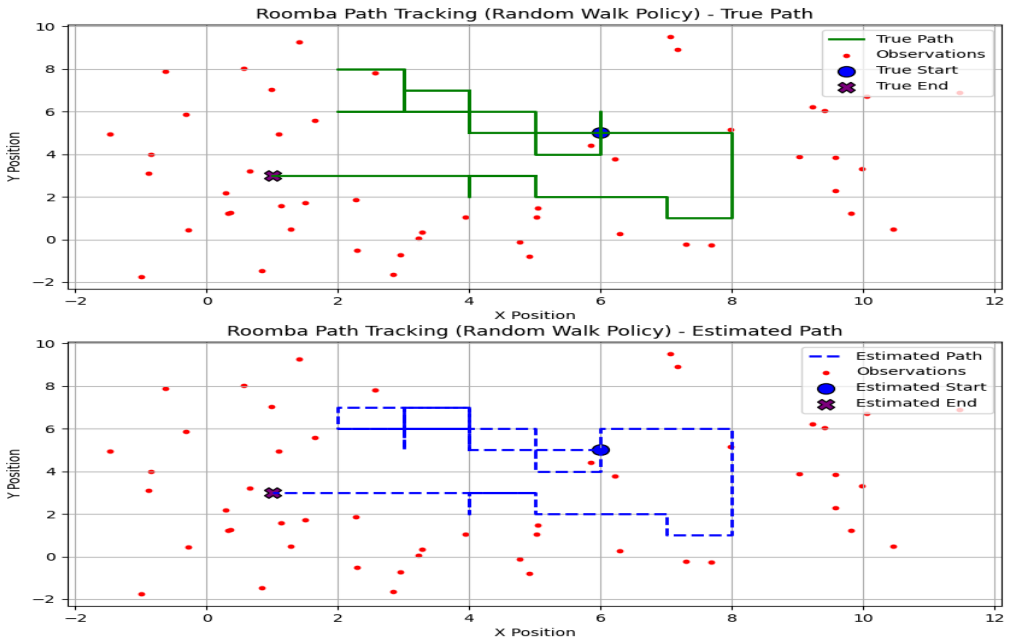


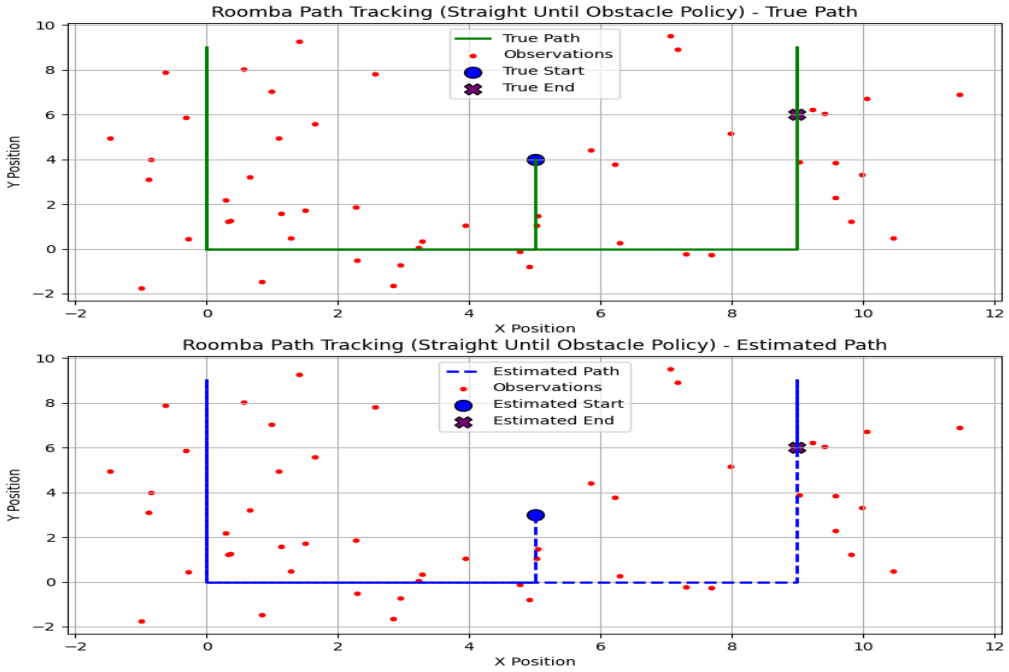
Seed Value -320





Seed Value -111





SEED Value -220

For seed value =111

Environment setup complete with a grid of size 10x10.

Simulating Roomba movement for policy: random_walk

Simulating Movement: 100%|| 50/50 [00:00<00:00, 84392.43it/s]

Simulating Roomba movement for policy: straight_until_obstacle

Simulating Movement: 100%|| 50/50 [00:00<00:00, 44131.99it/s]

Processing policy: random_walk

Tracking accuracy for random walk policy: 42.00%

Processing policy: straight_until_obstacle

Tracking accuracy for straight until obstacle policy: 94.00%

seed= 0
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement: 100%|| 50/50 [00:00<00:00, 92061.11it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement: 100%|| 50/50 [00:00<00:00, 100873.11it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 64.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 74.00%
seed= 20
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|| 50/50 [00:00<00:00, 101311.69it/s]

Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|| 50/50 [00:00<00:00, 103614.23it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 56.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 94.00%
seed= 40
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|| 50/50 [00:00<00:00, 100246.27it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|| 50/50 [00:00<00:00, 109340.56it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 52.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 94.00%
seed= 60
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|| 50/50 [00:00<00:00, 95022.75it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|| 50/50 [00:00<00:00, 105015.12it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 70.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 82.00%
seed= 80
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|| 50/50 [00:00<00:00, 97406.04it/s]

Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|| 50/50 [00:00<00:00, 99438.22it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 58.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 64.00%
seed= 100
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|| 50/50 [00:00<00:00, 102100.88it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|| 50/50 [00:00<00:00, 109397.60it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 54.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 74.00%
seed= 120
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|| 50/50 [00:00<00:00, 92712.29it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|| 50/50 [00:00<00:00, 90628.87it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 64.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 68.00%
seed= 140
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|| 50/50 [00:00<00:00, 97000.56it/s]

Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|| 50/50 [00:00<00:00, 106454.42it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 52.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 94.00%
seed= 160
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%| 50/50 [00:00<00:00, 92958.87it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%| 50/50 [00:00<00:00, 89468.94it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 38.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 72.00%
seed= 180
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%| 50/50 [00:00<00:00, 97632.77it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%| 50/50 [00:00<00:00, 93539.34it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 44.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 92.00%
seed= 200
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%| 50/50 [00:00<00:00, 96776.74it/s]

Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|██| 50/50
[00:00<00:00, 102700.88it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 56.00%
/home/krishna/Downloads/HMM_Question/HMM_Question/HMM_Question.py:326:
RuntimeWarning: More than 20 figures have been opened. Figures created through the
pyplot interface (matplotlib.pyplot.figure) are retained until explicitly closed and may
consume too much memory. (To control this warning, see the rcParam
figure.max_open_warning). Consider using matplotlib.pyplot.close().
  plt.figure(figsize=(10, 10))

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 94.00%
seed= 220
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|██| 50/50
[00:00<00:00, 93958.42it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|██| 50/50
[00:00<00:00, 100102.72it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 80.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 92.00%
seed= 240
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|██| 50/50
[00:00<00:00, 86338.08it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|██| 50/50
[00:00<00:00, 87417.76it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 54.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 94.00%

seed= 260
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|| 50/50 [00:00<00:00, 94636.82it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|| 50/50 [00:00<00:00, 97451.30it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 76.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 92.00%
seed= 280
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|| 50/50 [00:00<00:00, 97451.30it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|| 50/50 [00:00<00:00, 93664.67it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 46.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 92.00%
seed= 300
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|| 50/50 [00:00<00:00, 96553.96it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|| 50/50 [00:00<00:00, 100150.53it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 42.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 76.00%

seed= 320
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|██| 50/50
[00:00<00:00, 102350.02it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|██| 50/50
[00:00<00:00, 101311.69it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 52.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 90.00%
seed= 340
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|██| 50/50
[00:00<00:00, 98968.95it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|██| 50/50
[00:00<00:00, 101165.07it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 36.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 58.00%
seed= 360
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|██| 50/50
[00:00<00:00, 100631.09it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|██| 50/50
[00:00<00:00, 108435.99it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 58.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 74.00%

```
seed= 380
Environment setup complete with a grid of size 10x10.
Simulating Roomba movement for policy: random_walk
Simulating Movement:
100%|████████████████████████████████████████████████| 50/50
[00:00<00:00, 97496.61it/s]
Simulating Roomba movement for policy: straight_until_obstacle
Simulating Movement:
100%|████████████████████████████████████████████████| 50/50
[00:00<00:00, 94084.88it/s]

Processing policy: random_walk
Tracking accuracy for random walk policy: 18.00%

Processing policy: straight_until_obstacle
Tracking accuracy for straight until obstacle policy: 94.00%
```

References and Credits

1. Viterbi Algorithm - AIMA Python
2. Vacuum World - AIMA Python
3. Harsh (2022201) for contributing to the logic formulation.
4. Grammarly for grammar correction and proofreading.