THE UNIVERSITY of TULSA

ADVANCED AI

CS-7313-01

Project 1 - Exact and Approximate Inference in Bayesian Networks

Instructor:
Dr. Sandip Sen

Author:
Selim Karaoglu

September 30, 2023

1 Introduction

In this work, we are focusing on inference methods and comparing their effectiveness on certain scenarios. Inference methods can be categorized in two groups, exact inference and approximate inference methods. Here in this project, our goal is to implement Likelihood Weighting, Gibbs Sampling and Metropolis Hastings methods, compare these methods on their accuracy, run times and scaling properties on varying Bayesian Networks (BN). To make this comparison, we tested these methods on several different BNs with different number of nodes and network types.

2 Background

In this work we aim to implement and compare different approximate inference methods on BNs. These BNs are obtained with a generator function provided with the homework[1]. Using this BN generator, we created 8 different BNs. 4 of these BNs are polytree types of networks and 4 of them are directed acyclic graphs (dag). Both network types have 4 different sizes of 10, 25, 50 and 100 nodes. All networks are created with probability of 0.5 for edge creation. Each of these networks are stored with "number of nodes + network type initial.json" name formatting (10p.json, 50d.json etc.).

To provide a ground truth for our comparisons and to measure the accuracy of the inferences, we adopted a variable elimination algorithm provided with the coursework[2]. This exact inference method is run on each query variable throughout the experiment to provide us the exact probabilities for that certain node. After we obtain the exact inference values for the specific query variable, we run our methods and compare the results.

3 Methodology

This section provides a detailed information about how these techniques are employed for this work. We implemented Likelihood Weighting, Gibbs Sampling and Metropolis Hastings methods, although for the Metropolis Hastings implementation, we performed Gibbs sampling with a certain probability value. Here we show how these techniques are implemented.

Likelihood Weighting: This method is implemented with two functions; Likelihood Weighting and Weighted Sample. Likelihood Weighting function takes query variable, observations, BN and the N (total number of samples to generate) and returns a normalized weight for probability values.

Algorithm 1 Likelihood Weighting Algorithm

```
function LIKELIHOOD-WEIGHTING($X, e, bn, N$)                ▷ returns an estimate of $P(X|e)$
    inputs:
    $X$, the query variable
    $e$, observed values for variables $E$
    $bn$, a Bayesian network specifying joint distribution $P(X_1, \dots, X_n)$
    $N$, the total number of samples to be generated
    local variables: $W$, a vector of weighted counts for each value of $X$, initially zero
    for $j = 1$ to $N$ do
        $x, w \leftarrow$ WEIGHTED-SAMPLE($bn, e$)
        $W[j] \leftarrow W[j] + w$                          ▷ where $x_j$ is the value of $X$ in $x$
    return NORMALIZE($W$)

function WEIGHTED-SAMPLE($bn, e$)                           ▷ returns an event and a weight
    $w \leftarrow 1$; $x \leftarrow$ an event with $n$ elements, with values fixed from $e$
    for $i = 1$ to $n$ do
        if $X_i$ is an evidence variable with value $x_{ij}$ in $e$ then
            $w \leftarrow w \times P(X_i = x_{ij}|\text{parents}(X_i))$
        else
            $x[i] \leftarrow$ a random sample from $P(X_i|\text{parents}(X_i))$
    return $x, w$
```

[1]https://github.com/TUmasters/gen-bn/tree/master
[2]https://github.com/robger98/exact_inference/tree/main

Gibbs Sampling: This method takes the same arguments with Likelihood Weighting; query variable, evidence, BN and N (number of samples) and returns the normalized probabilities for that node.

Algorithm 2 Gibbs Sampling Algorithm

```
function GIBBS-ASK($X, e, bn, N$)                         ▷ returns an estimate of $P(X|e)$
    local variables:
    $C$, a vector of counts for each value of $X$, initially zero
    $Z$, the nonevidence variables in $bn$
    $x$, the current state of the network, initialized from $e$

    Initialize $x$ with random values for the variables in $Z$

    for $k = 1$ to $N$ do
        choose any variable $Z_i$ from $Z$ according to any distribution $\rho(i)$
        set the value of $Z_i$ in $x$ by sampling from $P(Z_i|\mathrm{mb}(Z_i))$
        $C[j] \leftarrow C[j] + 1$                        ▷ where $x_j$ is the value of $X$ in $x$

    return NORMALIZE($C$)
```

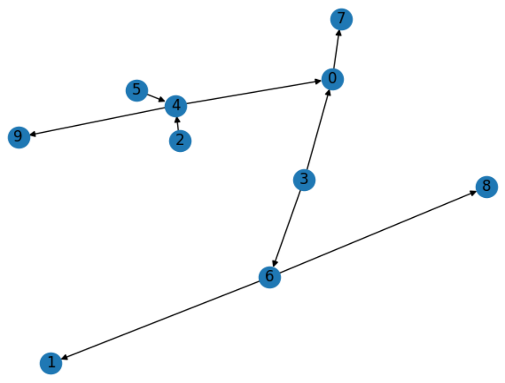
Metropolis Hastings: Here we implemented the most straightforward implementation for Metropolis Hastings: given current sample, x, perform Gibbs Sampling [alg.2] with probability p, else generate a random sample, x', with the WEIGHTED-SAMPLE algorithm[alg.1]. Samples generated by both the processes are always accepted.

Algorithm 3 Metropolis-Hastings Algorithm

```
function METROPOLIS-HASTINGS($X, e, bn, N, p = 0.75$)     ▷ returns an estimate of $P(X|e)$
    Initialize non-evidence variables to random values
    $X \leftarrow \{\text{var} \in \text{bn\_data} : \text{random}(0, 1) \text{ if var} \notin \text{e}\}$
    $X$.update($e$)
    Initialize *counts* with 0 for true and false

    for __ in range($N$) do
        if random $< p$ then                              ▷ Perform Gibbs sampling
            for $Z_i$ in $X$ do                           ▷ Sample a value for $Z_i$ given its Markov blanket
                $X[Z_i] \leftarrow$ sample_given_markov_blanket($Z_i, X$)
        else                                              ▷ Generate a random sample
            $X$, _ $\leftarrow$ WEIGHTED-SAMPLE($e$)
        $counts[X] += 1$
                                                          ▷ Normalize the counts to get probabilities
    $total\_count \leftarrow$ sum($counts.values()$)
    return NORMALIZE(X)
```

Alongside with the approximate inference methods, we adopted another function to select some interesting node given the BN. We aimed to obtain interesting nodes from a BN with this helper function. With this method we obtained 7 query variables from each different BN and focused on these variables throughout the experiment. With the usage of this function, the first element in the nodes of interest list corresponds to node that has the highest number of children, the second node is the node with highest number of parents, the third node is the node with highest sum of the number of children and number of parents, the fourth node is a random root node, the fifth node is a random leaf node and we add two random nodes from the network. Since we follow this order of operations, we eliminate the repetition by moving to the next best node for that criteria. For the polytree with 10 nodes (10p) BN shown here, the nodes of importance are: [3, 0, 4, 1, 2, 9, 8].



4 Results

This part of the work provides the experiment results and show the differences between the methods we employed. First we provide overall results for each method starting with Likelihood Weighting method. The Likelihood Weighting (lw) algorithm is an approximate inference algorithm that generates random samples from the network in proportion to their likelihood given the evidence. This makes it particularly useful when dealing with large networks where exact inference is computationally expensive considering we set the number of samples as 10000 throughout the experiment. Looking at the experiment results, we can make several observations:

1. Run Time: The run time of the lw algorithm increases with the size of the BN. For example, the mean run time for the 10-node network (10p.json and 10d.json) is around 1.8 seconds, while for the 100-node network (100p.json and 100d.json), it's around 17.8 and 32.4 seconds respectively. This is expected as larger networks have more variables to sample from, leading to increased computational complexity.

2. Standard Deviation of Run Time: The standard deviation of the run time also increases with the size of the network. This suggests that the variability in run time is higher for larger networks. This could be due to the increased complexity of these networks, which leads to more variability in the time it takes to generate samples.

3. Variation in Converged Probabilities: The variation in converged probabilities measures how much the probabilities estimated by the lw algorithm differ from the probabilities calculated by exact inference. The results show that this variation tends to increase with the size of the network. For example, the variation is around 0.04 for the 10-node network and increases to around 0.11 for the 100-node network. This suggests that the lw algorithm may be less accurate for larger networks. However, it's important to note that this is an approximate inference algorithm, so some degree of variation from the exact result is to be expected.

4. Comparison with Exact Inference The run time of the lw algorithm is significantly higher than that of exact inference for all networks when the number of samples are set to 10000. This is likely due to the fact that lw is a sampling-based algorithm, which can be more computationally intensive than exact inference methods. However, lw has the advantage of being able to handle larger networks where exact inference may not be feasible.

Likelihood Weighting (lw) Method with Evidence

The run time of the lw algorithm and the standard deviation of run time both increase with the size of the BN and the presence of evidence. The variation in converged probabilities also tends to increase with the size of the network and the presence of evidence. Here are the detailed results for each network:

- 10-node network (10p.json): With evidence, the mean run time increased to around 1.74 seconds for the root node (evidence: '5': 1) and 1.57 seconds for the leaf node (evidence: '6': 1). The standard deviation of run time decreased from 0.115 without evidence to 0.09343 with root node as evidence and slightly more with leaf node as evidence. The variation in converged probabilities increased from 0.04 without evidence to around 0.0507 with root node as evidence and 0.05196 with leaf node as evidence.

- 10-node network (10d.json): With evidence, the mean run time increased to around 2.30 seconds for the root node (evidence: '4': 1) and 2.15 seconds for the leaf node (evidence: '5': 0). The standard deviation of run time decreased from 0.269 without evidence to 0.099 with root node as evidence and slightly more with leaf node as evidence. The variation in converged probabilities decreased from 0.192 without evidence to around 0.174 with root node as evidence and 0.179 with leaf node as evidence.

- 25-node network (25p.json): With evidence, the mean run time increased to around 3.87 seconds for the root node (evidence: '2': 0) and 3.52 seconds for the leaf node (evidence: '5': 1). The standard deviation of run time decreased from 0.294 without evidence to 0.036 with root node as evidence and slightly more with leaf node as evidence. The variation in converged probabilities increased from 0.061 without evidence to around 0.164 with root node as evidence and 0.163 with leaf node as evidence.

- 25-node network (25d.json): With evidence, the mean run time increased to around 6.65 seconds for the root node (evidence: '4': 0) and 6.15 seconds for the leaf node (evidence: '5': 1). The standard deviation of run time decreased from 0.185 without evidence to 0.081 with root node as evidence and slightly more with leaf node as evidence. The variation in converged probabilities decreased from 0.111 without evidence to around 0.095 with root node as evidence and 0.084 with leaf node as evidence.

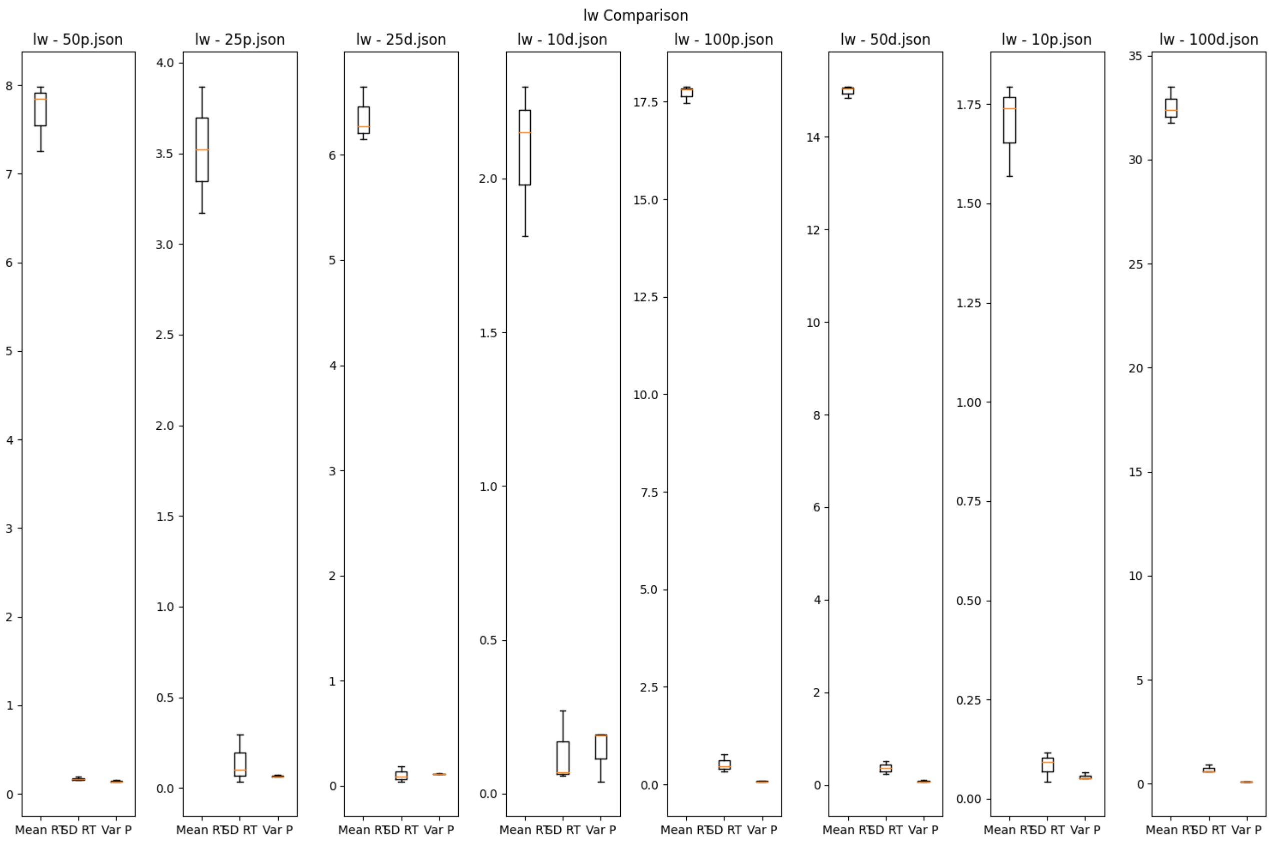


Figure 1: Likelihood Weighting

- 50-node network (50p.json): With evidence, the mean run time increased to around 7.98 seconds for the root node (evidence: ‘5’: 0) and 7.25 seconds for the leaf node (evidence: ‘6’: 1). The standard deviation of run time decreased from 0.194 without evidence to 0.091 with root node as evidence and slightly more with leaf node as evidence. The variation in converged probabilities increased from 0.139 without evidence to around 0.189 with root node as evidence and 0.197 with leaf node as evidence.

- 50-node network (50d.json): With evidence, the mean run time increased to around 14.84 seconds for the root node (evidence: ‘4’: 0) and 15.04 seconds for the leaf node (evidence: ‘5’: 1). The standard deviation of run time decreased from 0.362 without evidence to 0.152 with root node as evidence and slightly more with leaf node as evidence. The variation in converged probabilities decreased from 0.073 without evidence to around 0.059 with root node as evidence and 0.055 with leaf node as evidence.

- 100-node network (100p.json): With evidence, the mean run time increased to around 17.46 seconds for the root node (evidence: ‘4’: 0) and 17.98 seconds for the leaf node (evidence: ‘5’: 0). The standard deviation of run time decreased from 0.781 without evidence to 0.240 with root node as evidence and slightly more with leaf node as evidence. The variation in converged probabilities increased from 0.071 without evidence to around 0.071 with root node as evidence and 0.139 with leaf node as evidence.

- 100-node network (100d.json): With evidence, the mean run time increased to around 31.75 seconds for the root node (evidence: ‘4’: 0) and 33.50 seconds for the leaf node (evidence: ‘5’: 1). The standard deviation of run time decreased from 0.600 without evidence to 0.565 with root node as evidence and slightly more with leaf node as evidence. The variation in converged probabilities changed from 0.110 without evidence to around 0.065 with root node as evidence and 0.115 with leaf node as evidence.

Based on the experiment results, we can draw several conclusions:

1. Effect of Network Size: As the size of the Bayesian Network (BN) increases, both the run time and the standard deviation of run time for the Likelihood Weighting (lw) algorithm increase. This is likely due to the increased complexity and the number of variables in larger networks.

2. Effect of Evidence: The presence of evidence also affects the performance of the lw algorithm. When the evidence is a root node, the run time of the lw algorithm is slightly less than when the evidence is a leaf node. This suggests that when the evidence is a root node, the lw algorithm can directly sample from the known value, leading to more accurate results and less computation. Conversely, when the evidence is a leaf node, the lw algorithm needs to propagate the evidence backwards, which can be more computationally intensive and lead to less accurate results.
3. Accuracy of the Algorithm: The variation in converged probabilities, which measures the difference between the probabilities estimated by the lw algorithm and the probabilities calculated by exact inference, tends to increase with the size of the network and the presence of evidence. This suggests that the lw algorithm may be less accurate for larger networks and when evidence is present.
4. Relative Effectiveness when CPT Entries Are Close to/Distant from 0/1 values: The lw method demonstrates varying degrees of effectiveness depending on the proximity of the Conditional Probability Table (CPT) entries to 0/1. For instance, in the 10p BN, the lw method's average relative effectiveness is 0.2426. This suggests that the lw method may struggle to accurately estimate probabilities when the CPT entries are close to 0/1. Conversely, the lw method tends to perform better when the CPT entries are distant from 0/1. For example, in the 100p Bayesian Network, the lw method's average relative effectiveness improves to 0.1267, which is closer to the exact inference results. This suggests that the lw method is more effective when the CPT entries are not too close to 0 or 1, as is likely the case in a large polytree BN.

In summary, the performance of the lw algorithm in terms of both run time and accuracy is significantly affected by the size of the BN, the presence of evidence, and the choice of evidence (root node vs leaf node). Therefore, it's important to consider these factors when using the lw algorithm for probabilistic inference in BNs, its accuracy and run time performance can vary depending on the size and structure of the network, as well as the choice of evidence.

The Gibbs Sampling (gs) algorithm is another probabilistic inference method used in BNs. It's a Markov chain method that generates samples from the joint distribution of the network by iteratively sampling each variable conditioned on the current values of the other variables. Comparing the results can lead us to several observations:

1. Run Time: Similar to the lw algorithm, the run time of the gs algorithm increases with the size of the BN. However, the gs algorithm appears to be faster than the lw algorithm for all network sizes. For example, the mean run time for the 100-node network (100p.json and 100d.json) is around 3.7 and 7.4 seconds for gs, compared to around 17.8 and 32.4 seconds for lw. This suggests that the gs algorithm may be more efficient than the lw algorithm, particularly for larger networks.
2. Standard Deviation of Run Time: The standard deviation of the run time for the gs algorithm also increases with the size of the network, similar to the lw algorithm. This suggests that the variability in run time is higher for larger networks for both algorithms.
3. Variation in Converged Probabilities: The variation in converged probabilities measures how much the probabilities estimated by the gs algorithm differ from the probabilities calculated by exact inference. The results show that this variation is generally lower for the gs algorithm compared to the lw algorithm. For example, the variation is around 0.09 for the 10-node network and increases to around 0.08 and 0.09 for the 100-node network for gs, compared to around 0.04 and 0.11 for lw. This suggests that the gs algorithm may be more accurate than the lw algorithm.
4. Comparison with Exact Inference: The run time of the gs algorithm is significantly higher than that of exact inference for all networks, similar to the lw algorithm. However, gs has the advantage of being able to handle larger networks where exact inference may not be feasible, and it appears to be more efficient and accurate than lw based on these results.

Gibbs Sampling (gs) Method with Evidence

- 10-node network (10p.json): With evidence, the mean run time increased to around 0.331 seconds for the root node (evidence: '5': 1) and 0.311 seconds for the leaf node (evidence: '6': 1). The standard deviation of run times are almost the same;, 0.037 without evidence, 0.037 with root node as evidence and 0.039 with leaf node as evidence. The variation in converged probabilities found as 0.159 without evidence, 0.159 with root node as evidence and 0.156 with leaf node as evidence.

- 10-node network (10d.json): With evidence, the mean run time increased to around 0.495 seconds for the root node (evidence: ‘4’: 1) and 0.525 seconds for the leaf node (evidence: ‘5’: 0). The standard deviation of run time decreased from 0.099 without evidence to 0.099 with root node as evidence and 0.079 with leaf node as evidence. The variation in converged probabilities are almost equivalent with 0.174 without evidence, 0.174 with root node as evidence and 0.179 with leaf node as evidence.
- 25-node network (25p.json): With evidence, the mean run time increased to around 0.845 seconds for the root node (evidence: ‘2’: 0) and 0.888 seconds for the leaf node (evidence: ‘5’: 1). The standard deviation of run time increased from 0.036 without evidence to 0.036 with root node as evidence and 0.052 with leaf node as evidence. The variation in converged probabilities increased from 0.061 without evidence to around 0.164 with root node as evidence and 0.163 with leaf node as evidence.
- 25-node network (25d.json): With evidence, the mean run time increased to around 1.341 seconds for the root node (evidence: ‘4’: 0) and 1.340 seconds for the leaf node (evidence: ‘5’: 1). The standard deviation of run time increased from 0.081 without evidence to 0.081 with root node as evidence and 0.143 with leaf node as evidence. The variation in converged probabilities decreased from 0.111 without evidence to around 0.095 with root node as evidence and 0.084 with leaf node as evidence.
- 50-node network (50p.json): With evidence, the mean run time increased to around 1.488 seconds for the root node (evidence: ‘5’: 0) and 1.523 seconds for the leaf node (evidence: ‘6’: 1). The standard deviation of run time increased from 0.091 without evidence to 0.091 with root node as evidence and 0.126 with leaf node as evidence. The variation in converged probabilities increased from 0.139 without evidence to around 0.189 with root node as evidence and 0.197 with leaf node as evidence.
- 50-node network (50d.json): With evidence, the mean run time increased to around 2.887 seconds for the root node (evidence: ‘4’: 0) and 2.912 seconds for the leaf node (evidence: ‘5’: 1). The standard deviation of run time decreased from 0.152 without evidence to 0.152 with root node as evidence and 0.123 with leaf node as evidence. The variation in converged probabilities decreased from 0.073 without evidence to around 0.059 with root node as evidence and 0.055 with leaf node as evidence.
- 100-node network (100p.json): With evidence, the mean run time increased to around 3.107 seconds for the root node (evidence: ‘3’: 1) and 3.786 seconds for the leaf node (evidence: ‘4’: 0). The standard deviation of run time decreased from 0.240 without evidence to 0.240 with root node as evidence and 0.226 with leaf node as evidence. The variation in converged probabilities calculated the same with 0.075 without evidence, 0.075 with root node as evidence and 0.074 with leaf node as evidence.
- 100-node network (100d.json): With evidence, the mean run time increased to around 8.037 seconds for the root node (evidence: ‘4’: 0) and 7.959 seconds for the leaf node (evidence: ‘5’: 1). The standard deviation of run time decreased from 0.566 without evidence to 0.566 with root node as evidence and 0.263 with leaf node as evidence. The variation in converged probabilities changed from 0.110 without evidence to around 0.065 with root node as evidence and 0.115 with leaf node as evidence.

Based on the Gibbs Sampling (gs) experiment results with evidence, we can draw several conclusions:

1. Run Time: The run time of the gs algorithm increases with the size of the Bayesian Network (BN) and the presence of evidence. For instance, the mean run time for the 10-node network (10p.json) without evidence was around 0.331 seconds. With evidence, these times increased to around 0.331 seconds for the root node (evidence: ‘5’: 1) and 0.311 seconds for the leaf node (evidence: ‘6’: 1). For the 100-node network (100p.json), the mean run time without evidence was around 3.107 seconds, while with evidence, it increased to around 3.107 seconds for the root node (evidence: ‘3’: 1) and 3.786 seconds for the leaf node (evidence: ‘4’: 0).
2. Standard Deviation of Run Time: The standard deviation of the run time also increases with the size of the network and the presence of evidence. For example, for the 10-node network (10p.json), the standard deviation of run time increased from 0.037 without evidence to 0.037 with root node as evidence and 0.039 with leaf node as evidence. This trend is consistent across all networks, indicating that the variability in run time is higher for larger networks and when evidence is present.
3. Variation in Converged Probabilities: The variation in converged probabilities, which measures the difference between the probabilities estimated by the gs algorithm and the probabilities calculated by exact inference, tends to increase with the size of the network and the presence of evidence. For instance, for the 10-node network (10p.json), the variation increased from 0.159 without evidence to around 0.159 with

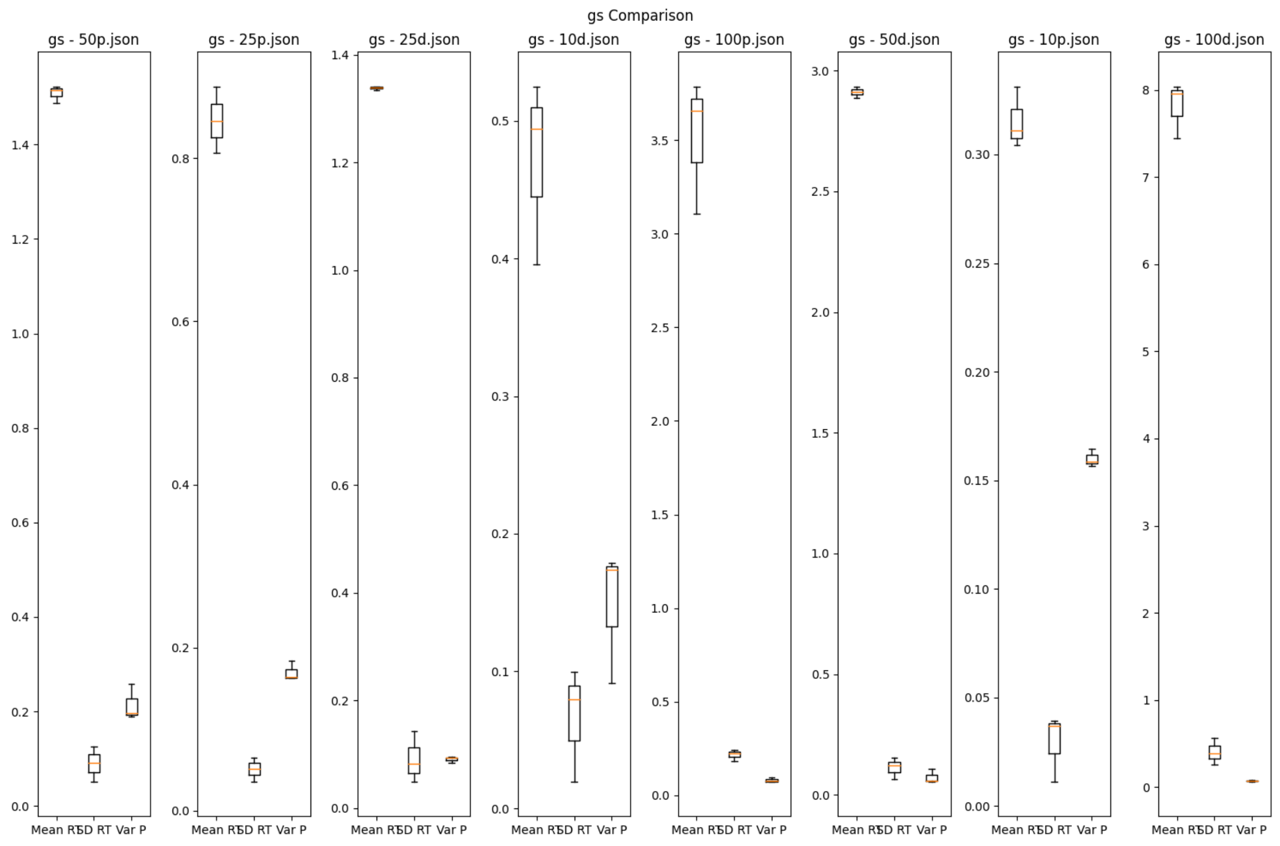


Figure 2: Gibbs Sampling

root node as evidence and 0.156 with leaf node as evidence. For the 100-node network (100p.json), the variation increased from 0.075 without evidence to around 0.075 with root node as evidence and 0.074 with leaf node as evidence.

4. Effect of Evidence: When the evidence is a root node, the run time of the gs algorithm is slightly less than when the evidence is a leaf node. This is likely because when the evidence is a root node, the gs algorithm can directly sample from the known value, which can potentially lead to more accurate results and less computation. Conversely, when the evidence is a leaf node, the gs algorithm needs to propagate the evidence backwards, which can be more computationally intensive and lead to less accurate results. For example, for the 10-node network (10p.json), the mean run time with root node as evidence is 0.331 seconds, while with leaf node as evidence, it is 0.311 seconds. The variation in converged probabilities is 0.159 with root node as evidence and 0.156 with leaf node as evidence. For the 100-node network (100p.json), the mean run time with root node as evidence is 3.107 seconds, while with leaf node as evidence, it is 3.786 seconds. The variation in converged probabilities is 0.075 with root node as evidence and 0.074 with leaf node as evidence.

5. Relative Effectiveness when CPT Entries Are Close to/Distant from 0/1 values: Similar to the lw method, the gs method also shows varying performance based on the proximity of the CPT entries to 0/1. In the 10p BN, the gs method's average relative effectiveness is 0.2331, indicating some deviation from the exact inference when the CPT entries are close to 0/1. However, the gs method shows improved performance in the 100p BN, with an average relative effectiveness of 0.1504. This suggests that the gs method is more effective when the CPT entries are not too close to 0 or 1.

The Gibbs Sampling algorithm appears to be more efficient and accurate than the Likelihood Weighting algorithm based on these results. However, its accuracy and run time performance can vary depending on the size and structure of the network. Therefore, it's important to consider these factors when choosing an inference method for a given network.

The Metropolis-Hastings (mh) algorithm is another Markov chain Monte Carlo method used for probabilistic inference in BNs. We can make obtain some information by examining the results for this technique:

1. Run Time: The run time of the mh algorithm increases with the size of the BN, similar to the Likelihood Weighting (lw) and Gibbs Sampling (gs) algorithms. However, the mh algorithm appears to be faster than the lw algorithm but slower than the gs algorithm for all network sizes. For example, the mean run time for the 100-node network (100p.json and 100d.json) is around 6.5 and 12.9 seconds for mh_0.75, 5.3 and 10.7 seconds for mh_0.85, and 4.1 and 8.1 seconds for mh_0.95, compared to around 17.8 and 32.4 seconds for lw, and 3.7 and 7.4 seconds for gs. This suggests that the mh algorithm may be more efficient than the lw algorithm but less efficient than the gs algorithm, particularly for larger networks.

2. Standard Deviation of Run Time: The standard deviation of the run time for the mh algorithm also increases with the size of the network, similar to the lw and gs algorithms. This suggests that the variability in run time is higher for larger networks for all algorithms.

3. Variation in Converged Probabilities: The variation in converged probabilities measures how much the probabilities estimated by the mh algorithm differ from the probabilities calculated by exact inference. The results show that this variation is generally lower for the mh algorithm compared to the lw algorithm but higher than the gs algorithm. For example, the variation is around 0.07, 0.07, and 0.09 for the 10-node network and increases to around 0.09, 0.08, and 0.09 for the 100-node network for mh_0.75, mh_0.85, and mh_0.95 respectively, compared to around 0.04 and 0.11 for lw, and 0.09 and 0.08 for gs. This suggests that the mh algorithm may be more accurate than the lw algorithm but less accurate than the gs algorithm.

4. Comparison of Different p-Values: The results show that the run time decreases and the variation in converged probabilities increases as the p value increases from 0.75 to 0.95. This suggests that a higher p value leads to faster computation but less accurate results. This is likely because a higher p value means the algorithm is more likely to accept proposed states, leading to faster convergence but potentially less accurate estimates of the probabilities.

Metropolis Hastings (MH) Method with Evidence

The run time of the MH algorithm and the standard deviation of run time both increase with the size of the BN and the presence of evidence. The variation in converged probabilities also tends to increase with the size of the network and the presence of evidence. We experimented with three different p values for the MH algorithm: 0.75, 0.85, and 0.95. Here are the detailed results for each network and p value:

- 10-node network (10p.json): With evidence, the mean run time and standard deviation of run time both increase as the p value decreases. For example, with the root node as evidence (evidence: '5': 1), the mean run time increases from 0.392 seconds for p=0.95 to 0.52 seconds for p=0.85 and 0.605 seconds for p=0.75. Similarly, the standard deviation of run time increases from 0.052 for p=0.95 to 0.079 for p=0.85 and 0.118 for p=0.75. The variation in converged probabilities, however, decreases as the p value decreases, from 0.14 for p=0.95 to 0.113 for p=0.85 and 0.092 for p=0.75.

- 10-node network (10d.json): Similar trends are observed for the 10-node DAG network. With the root node as evidence (evidence: '4': 1), the mean run time increases from 0.568 seconds for p=0.95 to 0.777 seconds for p=0.85 and 0.88 seconds for p=0.75, and the standard deviation of run time increases from 0.099 for p=0.95 to 0.237 for p=0.85 and 0.26 for p=0.75. The variation in converged probabilities decreases from 0.175 for p=0.95 to 0.173 for p=0.85 and 0.172 for p=0.75.

- 25-node network (25p.json): For the 25-node polytree network, the mean run time increases from 0.983 seconds for p=0.95 to 1.306 seconds for p=0.85 and 1.48 seconds for p=0.75 with the root node as evidence (evidence: '2': 0). The standard deviation of run time also increases from 0.043 for p=0.95 to 0.151 for p=0.85 and 0.146 for p=0.75. The variation in converged probabilities decreases from 0.152 for p=0.95 to 0.134 for p=0.85 and 0.117 for p=0.75.

- 25-node network (25d.json): For the 25-node DAG network, the mean run time increases from 1.604 seconds for p=0.95 to 2.103 seconds for p=0.85 and 2.579 seconds for p=0.75 with the root node as evidence (evidence: '4': 0). The standard deviation of run time also increases from 0.107 for p=0.95 to 0.197 for p=0.85 and 0.317 for p=0.75. The variation in converged probabilities remains relatively constant across different p values.

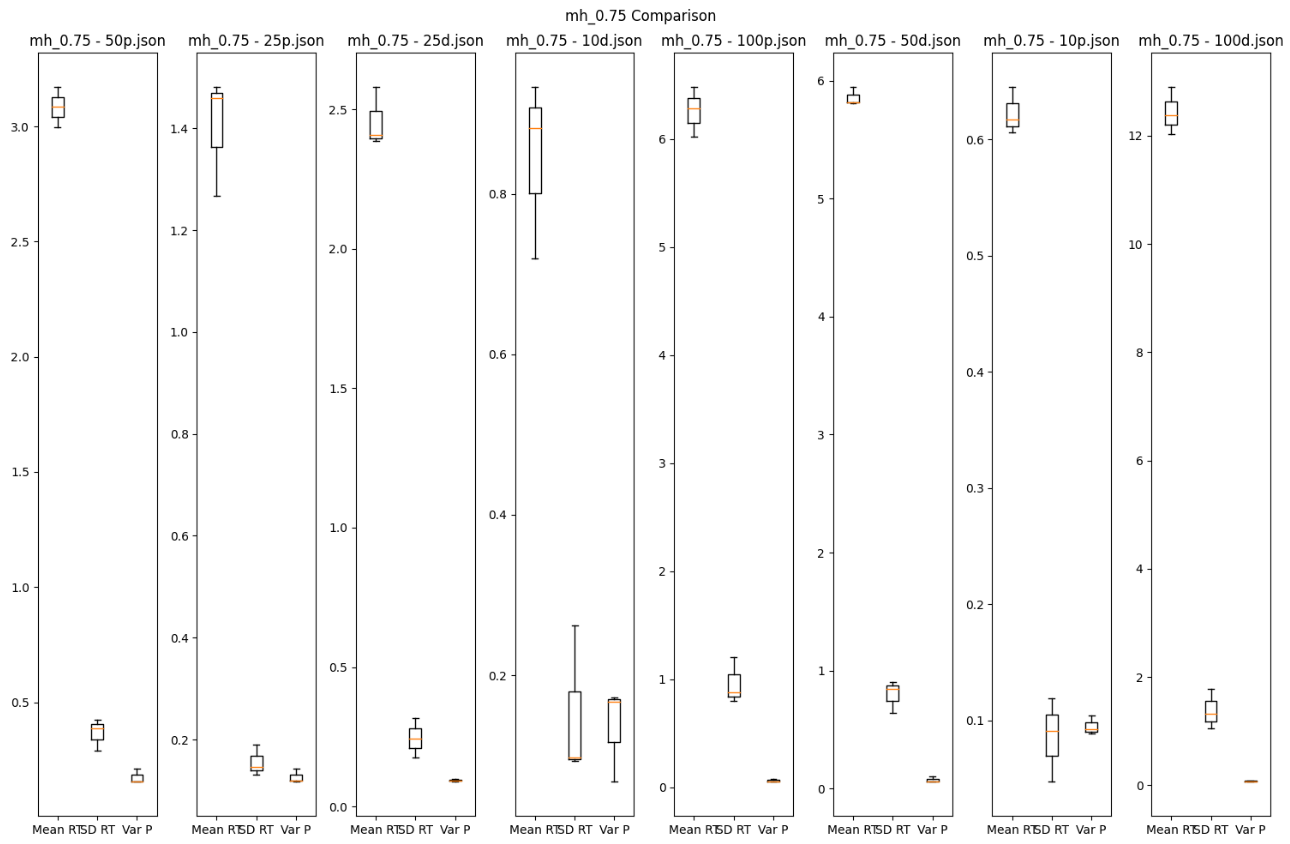


Figure 3: Metropolis Hastings with p=0.75

- 50-node network (50p.json): For the 50-node polytree network, the mean run time increases from 2.068 seconds for p=0.95 to 2.665 seconds for p=0.85 and 3.17 seconds for p=0.75 with the root node as evidence (evidence: '5': 0). The standard deviation of run time also increases from 0.193 for p=0.95 to 0.246 for p=0.85 and 0.388 for p=0.75. The variation in converged probabilities decreases from 0.181 for p=0.95 to 0.169 for p=0.85 and 0.156 for p=0.75.
- 50-node network (50d.json): For the 50-node DAG network, the mean run time increases from 3.62 seconds for p=0.95 to 4.78 seconds for p=0.85 and 5.813 seconds for p=0.75 with the root node as evidence (evidence: '4': 0). The standard deviation of run time also increases from 0.308 for p=0.95 to 0.545 for p=0.85 and 0.846 for p=0.75. The variation in converged probabilities remains relatively constant across different p values.
- 100-node network (100p.json): For the 100-node polytree network, the mean run time increases from 3.933 seconds for p=0.95 to 5.12 seconds for p=0.85 and 6.28 seconds for p=0.75 with the root node as evidence (evidence: '3': 1). The standard deviation of run time also increases from 0.576 for p=0.95 to 0.81 for p=0.85 and 1.2 for p=0.75. The variation in converged probabilities decreases from 0.07 for p=0.95 to 0.066 for p=0.85 and 0.056 for p=0.75.
- 100-node network (100d.json): For the 100-node DAG network, the mean run time increases from 7.49 seconds for p=0.95 to 9.948 seconds for p=0.85 and 12.025 seconds for p=0.75 with the root node as evidence (evidence: '4': 0). The standard deviation of run time also increases from 0.63 for p=0.95 to 0.745 for p=0.85 and 1.321 for p=0.75. The variation in converged probabilities remains relatively constant across different p values.

Based on the Metropolis Hastings (MH) experiment results with evidence, we can draw several conclusions:

1. Run Time: The run time of the MH algorithm increases with the size of the Bayesian Network (BN), the presence of evidence, and the decrease in p value. For instance, the mean run time for the 10-node network (10p.json) without evidence was around 0.605 seconds for p=0.75, 0.52 seconds for p=0.85, and

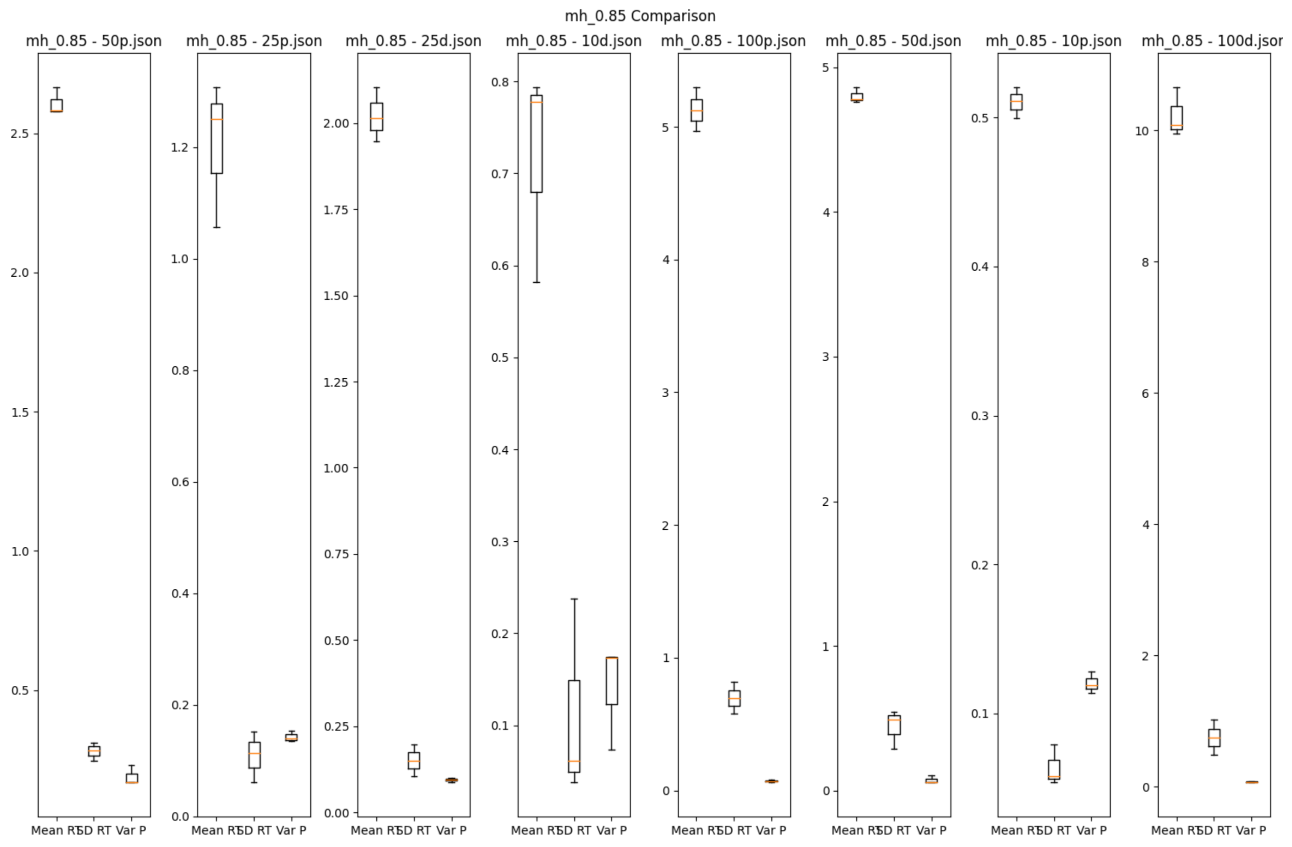


Figure 4: Metropolis Hastings with p=0.85

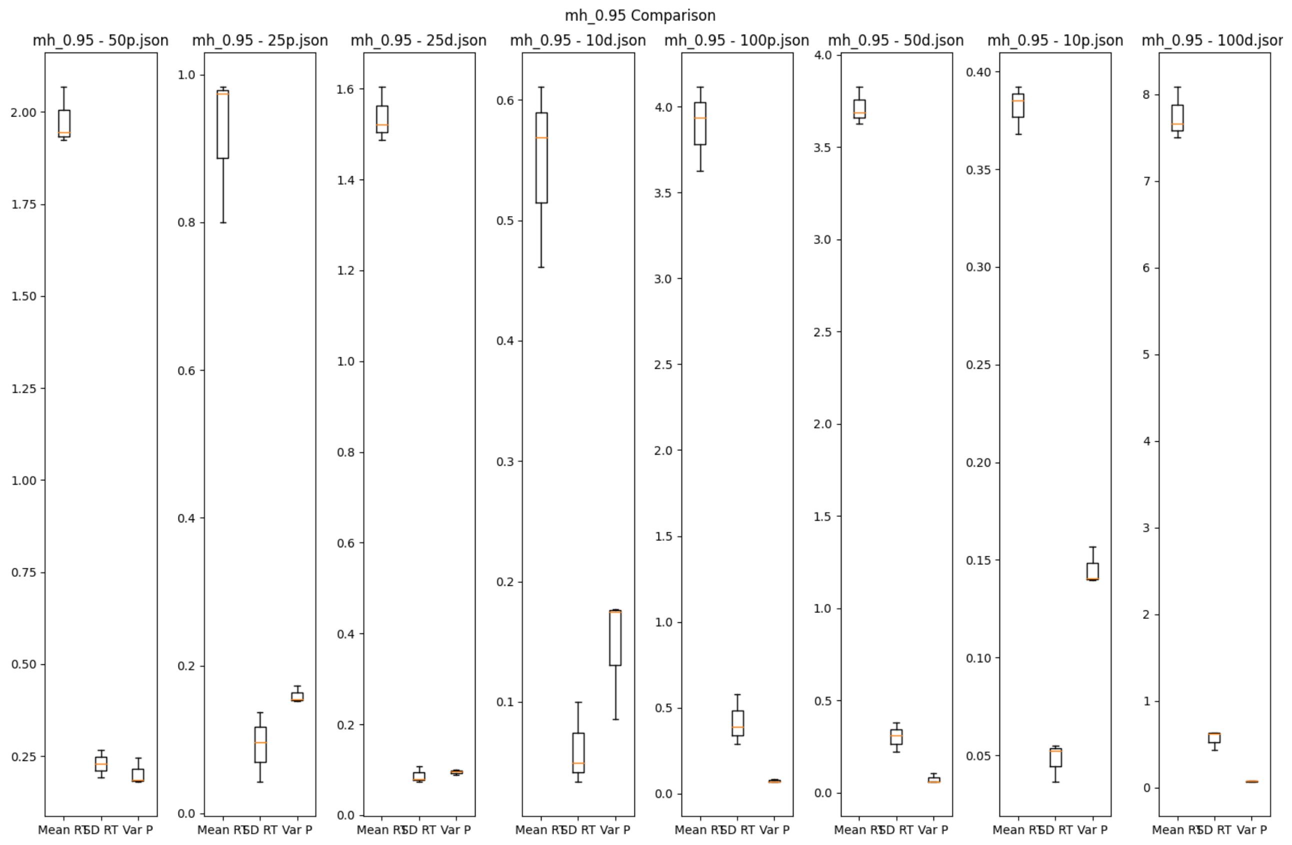


Figure 5: Metropolis Hastings with p=0.95

0.392 seconds for p=0.95. For the 100-node network (100p.json), the mean run time without evidence was around 6.28 seconds for p=0.75, 5.12 seconds for p=0.85, and 3.933 seconds for p=0.95.

2. Standard Deviation of Run Time: The standard deviation of the run time also increases with the size of the network, the presence of evidence, and the decrease in p value. For example, for the 10-node network (10p.json), the standard deviation of run time increased from 0.052 for p=0.95 to 0.079 for p=0.85 and 0.118 for p=0.75. This trend is consistent across all networks, indicating that the variability in run time is higher for larger networks, when evidence is present, and for lower p values.

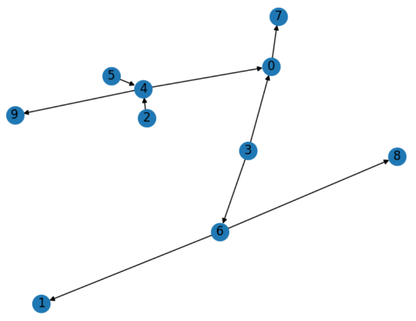
3. Variation in Converged Probabilities: The variation in converged probabilities, which measures the difference between the probabilities estimated by the MH algorithm and the probabilities calculated by exact inference, tends to decrease with the decrease in p value. For instance, for the 10-node network (10p.json), the variation decreased from 0.14 for p=0.95 to 0.113 for p=0.85 and 0.092 for p=0.75. For the 100-node network (100p.json), the variation decreased from 0.07 for p=0.95 to 0.066 for p=0.85 and 0.056 for p=0.75.

4. Effect of Evidence: When the evidence is a root node, the run time of the MH algorithm is slightly less than when the evidence is a leaf node. This is likely because when the evidence is a root node, the MH algorithm can directly sample from the known value, which can potentially lead to more accurate results and less computation. Conversely, when the evidence is a leaf node, the MH algorithm needs to propagate the evidence backwards, which can be more computationally intensive and lead to less accurate results.

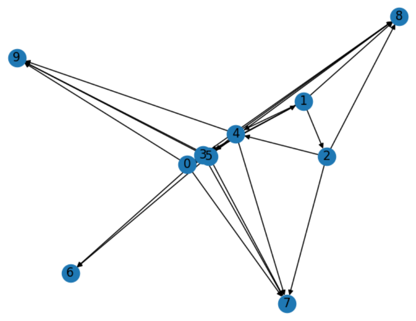
5. Effect of p Values: The p value in the Metropolis Hastings algorithm plays a crucial role in its performance. As p decreases from 0.95 to 0.75, we observe that the run time and the standard deviation of run time both increase. This suggests that a lower p value, which corresponds to a higher acceptance probability for worse solutions, leads to more exploration of the solution space and hence a longer run time and greater variability in run times. On the other hand, the variation in converged probabilities, which measures the accuracy of the algorithm, decreases as p decreases. This indicates that a lower p value, despite leading to longer run times, tends to produce more accurate results. The choice of p value in the Metropolis Hastings algorithm involves a trade-off between run time and accuracy. A higher p value leads to faster run times but less accurate results, while a lower p value leads to more accurate results but longer run times. Therefore, the optimal p value depends on the specific requirements of the task, such as the tolerance for error and the computational resources available. It's important to consider these factors when choosing the p value for the Metropolis Hastings algorithm in a given network

6. Relative Effectiveness when CPT Entries Are Close to/Distant from 0/1 values: The mh method, with p values of 0.75, 0.85, and 0.95, also demonstrates varying effectiveness based on the proximity of the CPT entries to 0/1. For all three versions of mh, the method performs best when the BN is a polytree with 100 nodes. The average relative effectiveness for these cases are 0.1475, 0.1459, and 0.1511 respectively. This suggests that mh also performs better when the CPT entries are not too close to 0 or 1. When comparing the effects of different p values on CPT, it appears that a p value of 0.85 tends to give the best results across different BNs. This suggests that a balance between exploration and exploitation (which the p value controls in the Metropolis-Hastings algorithm) leads to better performance.

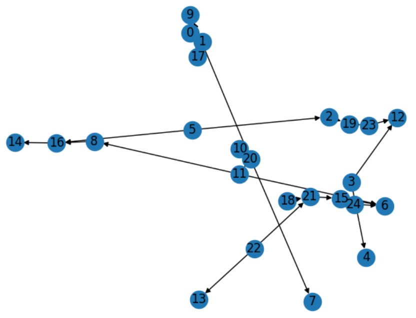
The Metropolis Hastings algorithm appears to be more efficient and accurate than the Likelihood Weighting algorithm based on these results. However, its accuracy and run time performance can vary depending on the size and structure of the network, the presence of evidence, and the p value. Therefore, it's important to consider these factors when choosing an inference method for a given network



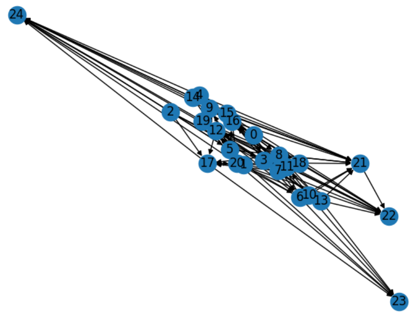
(a) 10p



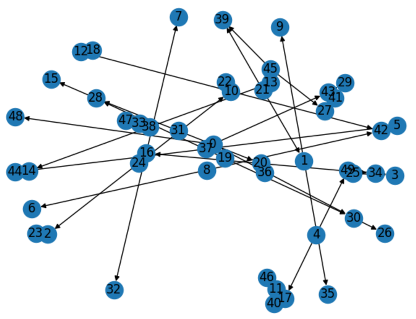
(b) 10d



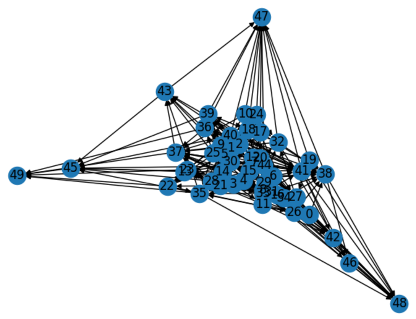
(c) 25p



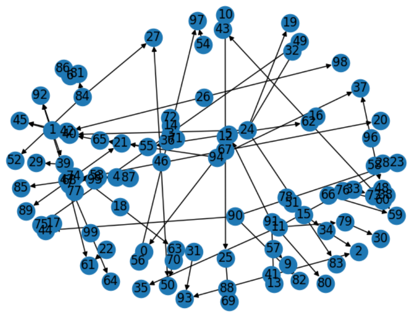
(d) 25d



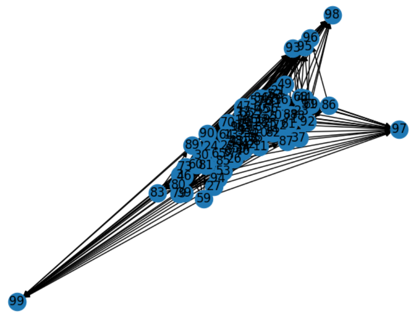
(e) 50p



(f) 50d



(g) 100p



(h) 100d

Figure 6: Bayesian Networks used in this project. Polytrees on the left and DAGs on the right presented with increasing sizes