CSC 442 Project 3 - Uncertain Inference

Name : Tushar Kumar
NetId : tusharku

November 27, 2018

Abstract

In this project I have implemented four inference methods for bayesian networks and demonstrated their implementation on few sample problems of inference using the provided xmlbif files. The four methods that I have implemented are Enumeration, Likelihood Weighting, Rejection Sampling and Gibbs Sampling. I discuss below my implementation both from design and thought perspective and also provide a detailed analysis of the results of my analysis based on different metrics.

1 Introduction

Bayesian network is directed acyclic graph whose vertices are the random variables $X \cup E \cup Y$, where

- X is the query variable
- E are the evidence variables
- e are the observed values for the evidence variables
- Y are the unobserved (or hidden) variables

Its a kind of probabilistic graphical model which can very efficiently be used to represent the causal relationships between different kinds of variable involved. In order to communicate across the model of the system, one needs to have a well defined formulation of how to represent a graphical model. For this purpose we use the XMLBIF file format. It is a XML-based format for exchanging Bayesian networks and is quite well known.

Our goal is to build a program which, given a XMLBIF file, representing a bayesian network, parses it and creates an internal representation of the network as a graph data structure. Post that, the program should be able to compute the posterior distribution of any query variable conditioned on a bunch of evidences provided as input to it.

2 Design

The basic design can be understood from the Class Diagram 1. In order to be able to apply different kinds of inference algorithms, I created different strategies following the obvious Strategy pattern for algorithms. The Bayesian Network is implemented as a graph which has nodes and the each node can have children or parents which basically encompasses the edge relationship in the graph. There is a factory method for creating the necessary inference algorithm instance leaving the clients agnostic to what behavior the inference algorithm is going to follow and only care about the returned probability distribution.

2.1 Key Design and High Level Implementation Details

a) Interface to handle any kind of inference algorithm

b) Built a **Bayesian Network graph** that captures the parsed BIF file into a graph representation.

c) **No restriction on the domain of variables. Can be any discrete category list and not just true or false.**

d) **Built a Parser** to read the different BIF files provided.

e) **Implemented Enumeration** method for completing Part I of project requirement

f) **Implemented Likelihood Weighting, Rejection Sampling and Gibbs Sampling** for completing Part II of project requirement

g) **Built a Simulator to create complex graph** to test the different methods with respect to increase in node size and **can test upto 10,000 nodes**

h) **Evaluated the different methods based on complexity of graph(in terms of variables involved)** and time taken to evaluate a query.

i) **Evaluated the approximate methods** based on the **KL Divergence** distance of the true distribution(from enumeration) to the approximate distribution from the approximate inference methods

j) **Analyzed the effect of burn-in** on Gibbs Sampling method of inference

k) **Graceful Error Handling** and providing descriptive errors to direct the user in the correct way of running the program.

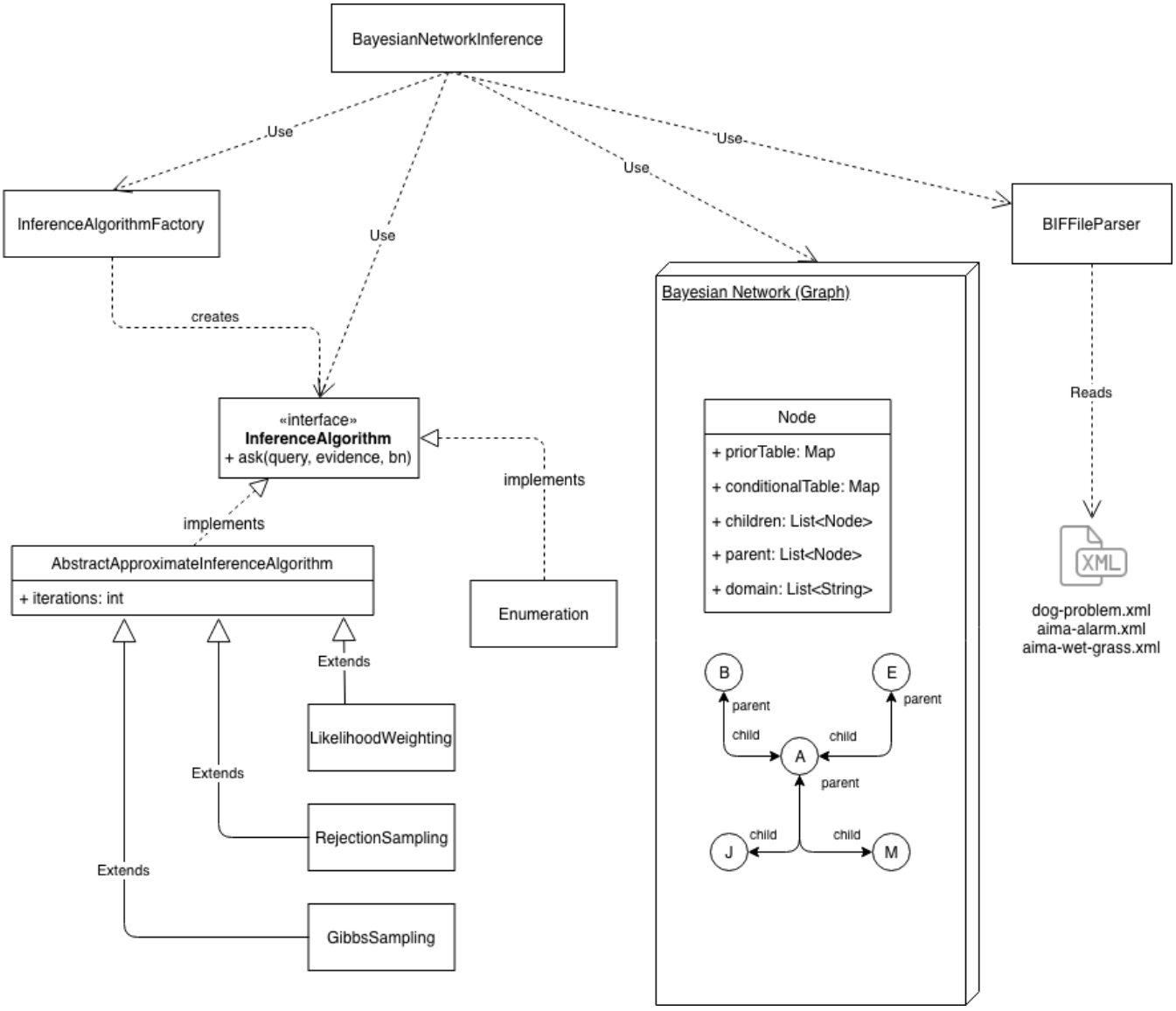

Figure 1: Class Diagram for Uncertain Inference Project

```java
/**
 * Method to recursively enumerate all possible
 * values of each variable and computing the probability
 * using all values of all non-evidence variables and
 * specified values of evidence variables
 * @param extendedEvidence
 * @param variables
 * @return
 */
private static Double enumerateAll(Map<String, String> extendedEvidence, List<BayesianNetwork.Node> variables) {
    if(variables.isEmpty()) {
        return 1.0;
    }
    else {
        List<BayesianNetwork.Node> variablesCopy = new ArrayList<BayesianNetwork.Node>(variables);
        Node selectedVariable = variablesCopy.remove(0);
        if(extendedEvidence.containsKey(selectedVariable.getName())) {
            return computeProbability(selectedVariable, extendedEvidence) *
                    enumerateAll(extendedEvidence, variablesCopy);
        }
        else {
            double sum = 0.0;
            for(String possibleValue : selectedVariable.getDomain()) {
                Map<String, String> extendedEvidenceCopy = new HashMap<String, String>(extendedEvidence);
                extendedEvidenceCopy.put(selectedVariable.getName(), possibleValue);
                sum += computeProbability(selectedVariable, extendedEvidenceCopy) *
                        enumerateAll(extendedEvidenceCopy, variablesCopy);
            }
            return sum;
        }
    }
}
```

Figure 2: Snapshot of Enumeration Method

3 Exact Inference

3.1 Enumeration

I have implemented the Enumeration method for Exact inference based on the pseudocode in AIMA Fig 14.9 3rd Ed. The procedure basically involves iterating through all the possible domain values of the query variable in order to build the probability distribution for that. For each possible value, we add that possible value to the evidence and then enumerate all possible options for the remaining variables. We iterate the graph in a topological manner and whenever we have a variable which is also a evidence, we compute the probability of that variable having that particular value. In case the variable is not having a fixed value, meaning its not in the evidences, we just compute the probability using the sum rule. This is basically achieving marginalization of that variable since its not present in the query or in evidence list. In order to completely marginalize this variable we also recursively enumerate and compute the probabilities of the nodes present below the current node in the topologically sorted graph. I provide below the snapshot of the main method for enumeration in Figure 2

3.1.1 Example Queries

Once the numbers have been computed for the possible values, I normalize those values to make them a probaility distribution. Here are some particular examples of running the

```
================================================
Result of Running Inference Algorithm : Enumeration Exact Inference

Evidence Given : {A=true, E=false}

Probability for B being true = 0.48478597215059305

Probability for B being false = 0.515214027849407

Time Taken for running the algorithm (in ms) = 2
```

Figure 3: Snapshot of Enumeration Method on aima-alarm

```
================================================
Result of Running Inference Algorithm : Enumeration Exact Inference

Evidence Given : {C=true}

Probability for W being true = 0.7452

Probability for W being false = 0.2548

Time Taken for running the algorithm (in ms) = 2
```

Figure 4: Snapshot of Enumeration Method on aima-wet-grass

enumeration method on the three possible queries for each of the sample xmlbif files provided.

aima-alarm

Evidence : A(Alarm) = true, E(Earthquake) = false
Query : B(Burglary)
I have attached the output for this particular example problem in Figure 3

aima-wet-grass

Evidence : C(Cloudy) = true
Query : G(Grass being wet)
I have attached the output for this particular example problem in Figure 4

dog-problem

Evidence : hear-bark = true
Query : family-out
I have attached the output for this particular example problem in Figure 5

4 Approximate Inference

The mechanism of enumerationm, while seemingly quick for small graphs is not a scalable one. The complexity of the enumeration solution is exponential because we are practically evaluating all possible combinations of nodes in graph. Hence a graph with N nodes, where

```
================================================
Result of Running Inference Algorithm : Enumeration Exact Inference

Evidence Given : {hear-bark=true}

Probability for family-out being true = 0.3346363608428431

Probability for family-out being false = 0.665363639157157

Time Taken for running the algorithm (in ms) = 1
```

Figure 5: Snapshot of Enumeration Method on dog-problem

each node has a domain size of K, will cause the complexity to be K^N, which will definitely not work for graphs with say 100 nodes even if the domain is just true or false, in fact it will also not work on graphs with 30 nodes for that matter. This requires the development of inferences algorithms which instead of finding the exact probability distribution, provide with us a 'good ' approximation of that distribution in polynomial time. This is exactly what the below provided methods give us.

4.1 Rejection Sampling

Rejection sampling is basically just randomly drawing out samples and rejecting any sample that is inconsistent with the provided evidence. So to use the example bif file, say we had to find the probability of grass being wet given that its cloudy. One way to get this, albeit requiring huge patience, is to monitor the weather every day and keep track of grass being wet each day. Since we are only interested in the posterior distribution given that its cloudy, we will just monitor days where it was indeed cloudy and keep the observing grass being wet in those days. After few years we would definitely have the right probaility distribution of grass being wet given that its cloudy. Rejection sampling is exactly this concept. I have implemented the Rejection sampling method analogous to the pseudocode in AIMA Fig 14.14 3rd Ed. The procedure basically involves iterating through all the nodes in the graph and computing the probability values given its parent using prior values if the nodes do not have any parent and using conditional probability distribution values in case they do. Finally we check for consistency between our generated sample and evidence and accept the sample, if its indeed consistent. I provide below the snapshot of the main method for rejection sampling in Figure 6

4.1.1 Example Queries

Once the numbers have been computed for the possible values, I normalize those values to make them a probability distribution. Here are some particular examples of running the rejection sampling method on the three possible queries for each of the sample xmlbif files provided.

aima-alarm

Evidence : A(Alarm) = true, E(Earthquake) = false

```java
/**
 * Overridden method for asking the network for
 * the distribution of a query variable using the
 * provided evidences.
 */
@Override
public Map<String, Double> ask(String query, Map<String, String> evidences,
        BayesianNetwork network){
    Map<String, Double> distribution = new HashMap<String, Double>();
    for(int sampleCount=1; sampleCount <= numberOfIterations; sampleCount++) {
        Map<String, String> sample = getSampleFromBayesianNetwork(network);
        if(isConsistent(evidences, sample)) {
            distribution.put(sample.get(query),
                    distribution.getOrDefault(sample.get(query), 0.0) + 1.0);
        }
    }
    for(String value : network.getHeaderTable().get(query).getDomain()) {
        if(!distribution.containsKey(value)) {
            distribution.put(value, 0.0);
        }
    }
    return BayesianNetworkUtils.normalize(distribution);
}
```

Figure 6: Snapshot of Rejection Sampling Method

```
================================================
Result of Running Inference Algorithm : Rejection Sampling Approximate Inference

Evidence Given : {A=true, E=false}

Probability for B being true = 0.5212121212121212

Probability for B being false = 0.47878787878787876

Time Taken for running the algorithm (in ms) = 217
```

Figure 7: Snapshot of Rejection Sampling Method on aima-alarm

Query : B(Burglary)
Number of Samples : 100000 (More samples because we need to capture enough samples for burglary being true which is rare event)
I have attached the output for this particular example problem in Figure 7

aima-wet-grass

Evidence : C(Cloudy) = true
Query : G(Grass being wet)
Number of Samples : 10000
I have attached the output for this particular example problem in Figure 8

dog-problem

Evidence : hear-bark = true
Query : family-out
Number of Samples : 10000
I have attached the output for this particular example problem in Figure 9

```
================================================
Result of Running Inference Algorithm : Enumeration Exact Inference

Evidence Given : {C=true}

Probability for W being true = 0.7452

Probability for W being false = 0.2548

Time Taken for running the algorithm (in ms) = 2
```

Figure 8: Snapshot of Rejection Sampling Method on aima-wet-grass

```
================================================
Result of Running Inference Algorithm : Rejection Sampling Approximate Inference

Evidence Given : {hear-bark=true}

Probability for family-out being false = 0.667595818815331

Probability for family-out being true = 0.33240418118466897

Time Taken for running the algorithm (in ms) = 53
```

Figure 9: Snapshot of Rejection Sampling Method on dog-problem

4.2 Likelihood Weighting

A major disadvantage of Rejection Sampling is that we are really throwing away a lot of samples. whereas instead we could have used them but with a much smaller weight. For instance say the probability of an evidence of having a particular value is 0.4 instead of trying 100 samples and being able to use 40 of them, we could just use all 100 of them. We would just to give these samples a weight of 0.4 instead of 1. This is the principle behind Likelihood Weighting. I have implemented the Likelihood Weighting method analogous to the pseudocode in AIMA Fig 14.15 3rd Ed. The procedure basically involves iterating through all the nodes in the graph and computing the probability values given its parent using prior values if the nodes do not have any parent and using conditional probability distribution values in case they do. Whenever we hit a variable which is present in the evidence, we find the probability of that node having the desired value given its parents in the sample and re-weight the current sample by that amount. We don't really need to check for consistency between our generated sample and evidence because we are enforcing all samples to match with evidence. I provide below the snapshot of the main method for likelihood weighting in Figure 10

4.2.1 Example Queries

Once the numbers have been computed for the possible values, I normalize those values to make them a probability distribution. Here are some particular examples of running the rejection sampling method on the three possible queries for each of the sample xmlbif files provided.

aima-alarm

```
/**
 * Helper method to get a sample from the given bayesian network
 * @param network
 * @param evidences
 * @return
 */
private Map<String, String> getSampleFromBayesianNetwork(BayesianNetwork network,
        Map<String, String> evidences) {
    Map<String, String> sample = new HashMap<String, String>();
    List<BayesianNetwork.Node> nodeList = BayesianNetworkUtils.getNodesInTopologicalOrder(network);
    currentSampleweight = 1.0;
    for(Node node : nodeList) {
        if(evidences.containsKey(node.getName())) {
            sample.put(node.getName(), evidences.get(node.getName()));
            currentSampleweight = currentSampleweight * computeProbability(node, sample);
        }
        else {
        if(node.getPriorTable().isEmpty()) {
            Map<List<Integer>, Map<Integer, Double>> cpt = node.getConditionalTable();
            List<Integer> parentIndex = new ArrayList<Integer>();
            for(Node parent : node.getParents()) {
                String parentVal = sample.get(parent.getName());
                parentIndex.add(parent.getDomain().indexOf(parentVal));
            }
            Map<Integer, Double> cptForParentValue = cpt.get(parentIndex);
            BayesianNetworkUtils.generateSampleFromDistribution(sample, node, cptForParentValue);
        }
        else {
            Map<Integer, Double> prior = node.getPriorTable();
            BayesianNetworkUtils.generateSampleFromDistribution(sample, node, prior);
        }
        }
    }
    return sample;
}
```

Figure 10: Snapshot of Likelihood Weighting Method

```
================================================
Result of Running Inference Algorithm : Likelihood Weighting Approximate Inference

Evidence Given : {A=true, E=false}

Probability for B being false = 0.48685954924722247

Probability for B being true = 0.5131404507527775

Time Taken for running the algorithm (in ms) = 242
```

Figure 11: Snapshot of Likelihood Weighting Method on aima-alarm

```
================================================
Result of Running Inference Algorithm : Likelihood Weighting Approximate Inference

Evidence Given : {C=true}

Probability for W being true = 0.7469

Probability for W being false = 0.2531

Time Taken for running the algorithm (in ms) = 46
```

Figure 12: Snapshot of Likelihood Weighting Method on aima-wet-grass

Evidence : A(Alarm) = true, E(Earthquake) = false
Query : B(Burglary)
Number of samples : 100000 (More samples because we need to capture enough samples for burglary being true which is rare event)
I have attached the output for this particular example problem in Figure 11

aima-wet-grass

Evidence : C(Cloudy) = true
Query : G(Grass being wet)
Samples : 10000
I have attached the output for this particular example problem in Figure 12

dog-problem

Evidence : hear-bark = true
Query : family-out
Samples : 10000
I have attached the output for this particular example problem in Figure 13

4.3 Gibbs Sampling

Even though we could use the samples in their entirety using Likelihood Weighting, the issue that arises is that for networks which have very rare events or even networks which have too many nodes because of a lot of small probabilities being multiplied together till we cover all the nodes, the weight of that sample ends up being very small. So it practically does

```
================================================
Result of Running Inference Algorithm : Likelihood Weighting Approximate Inference

Evidence Given : {hear-bark=true}

Probability for family-out being false = 0.6580703936042327

Probability for family-out being true = 0.3419296063957672

Time Taken for running the algorithm (in ms) = 55
```

Figure 13: Snapshot of Likelihood Weighting Method on dog-problem

not remain of much use because we would have to get many such samples again, to finally have any effect on the overall probability distribution. A way out of this scenario is using Gibbs Sampling which is built as a special case of Metropolis-Hastings method. With Gibbs sample we try to sample from a new probability distribution given the old sample we had gotten. So say we first get a sample x' , we then try to get another sample from a distribution Q(x|x'). Even though the conditional probability might seem intimidating specially for large networks, we actually only need to deal with the markov blanket of the node that we are trying to get a probability distribution of. The best part about this method is that you are guaranteed to converge no matter what is your starting point. This is because of the ergodicity of the algorithm which implies that you end up reaching a stationary distribution Q, no matter what your starting distribution is where the stationary distribution implies that the following balanced state condition holds

$$Q(x')P(x|x') = Q(x)P(x'|x)$$

I have implemented the Gibbs Sampling method analogous to the pseudocode in AIMA Fig 14.16 3rd Ed. The procedure basically involves starting with a random sample and then randomly choosing another non-evidence node and computing its conditional distribution based on the values of its markov blanket in the sample. We then sample a new value for this node based on that probability distribution and repeat the steps for the given number of samples. We also have to deal with burn-in, that is the time or number of samples that it takes for the algorithm to reach to the balanced state condition. I provide below the snapshot of the main method for Gibbs Sampling in Figure 14 . One thing to point out in order to do a fair comparison between all the methods regarding the number of samples, I purposely tweak the gibbs sampling code to modify the number of samples provided to it as input by dividing it with the number of non evidence variables. The reason being I want like all other inference methods, every iteration to produce one sample but gibbs at every iteration goes through all non evidence variables and instead produces as many samples as there are non evidence variables. If I use the AIMA pseudocode as it is then it would produce N sample per iterations where N is the number of non evidence nodes in the network. Hence I divide the input number of samples by N to ensure that for the same input gibbs, likelihood, rejection sampling, all generate same number of samples.

4.3.1 Example Queries

Once the numbers have been computed for the possible values, I normalize those values to make them a probability distribution. Here are some particular examples of running the

```java
/**
 * Overridden method for asking the network for
 * the distribution of a query variable using the
 * provided evidences.
 */
@Override
public Map<String, Double> ask(String query, Map<String, String> evidences,
        BayesianNetwork network){
    List<BayesianNetwork.Node> nodeList = BayesianNetworkUtils.getNodesInTopologicalOrder(network);
    List<BayesianNetwork.Node> nonEvidenceNodes = new ArrayList<BayesianNetwork.Node>();
    for(Node node : nodeList) {
        if(!evidences.containsKey(node.getName())) {
            nonEvidenceNodes.add(node);
        }
    }

    Map<String, String> sample = new HashMap<String, String>();
    for(Node node : nonEvidenceNodes) {
        int randomDomainIndex = new Random().nextInt(node.getDomain().size());
        sample.put(node.getName(), node.getDomain().get(randomDomainIndex));
    }

    for(String key : evidences.keySet()) {
        sample.put(key, evidences.get(key));
    }

    Map<String, Double> distribution = new HashMap<String, Double>();
    int totalSamples = (int) numberOfSamples/nonEvidenceNodes.size();
    int sampleCount = 0;
    for(int iter = 1; iter <= totalSamples; iter++) {
        for(Node nonEvidenceNode : nonEvidenceNodes) {
            updateSample(nonEvidenceNode, sample, network);
            if(sampleCount >= burnInSamples) {
                distribution.put(sample.get(query),
                        distribution.getOrDefault(sample.get(query), 0.0) + 1.0);
            }
            sampleCount++;
        }
    }

    for(String value : network.getHeaderTable().get(query).getDomain()) {
        if(!distribution.containsKey(value)) {
            distribution.put(value, 0.0);
        }
    }

    return BayesianNetworkUtils.normalize(distribution);
}
```

Figure 14: Snapshot of Gibbs Sampling Method

```
================================================
Result of Running Inference Algorithm : Gibbs Sampling Approximate Inference

Evidence Given : {A=true, E=false}

Probability for B being true = 0.4846206915360521

Probability for B being false = 0.515379308463948

Time Taken for running the algorithm (in ms) = 755
```

Figure 15: Snapshot of Gibbs Sampling Method on aima-alarm

```
================================================
Result of Running Inference Algorithm : Gibbs Sampling Approximate Inference

Evidence Given : {C=true}

Probability for W being false = 0.2535472616362404

Probability for W being true = 0.7464527383637596

Time Taken for running the algorithm (in ms) = 169
```

Figure 16: Snapshot of Gibbs Sampling Method on aima-wet-grass

rejection sampling method on the three possible queries for each of the sample xmlbif files provided.

aima-alarm

Evidence : A(Alarm) = true, E(Earthquake) = false
Query : B(Burglary)
Number of Samples : 100000 (More samples because we need to capture enough samples for burglary being true which is rare event)
I have attached the output for this particular example problem in Figure 15

aima-wet-grass

Evidence : C(Cloudy) = true
Query : G(Grass being wet)
Samples : 10000
I have attached the output for this particular example problem in Figure 16

dog-problem

Evidence : hear-bark = true
Query : family-out
Samples : 10000
I have attached the output for this particular example problem in Figure 17

```
================================================
Result of Running Inference Algorithm : Likelihood Weighting Approximate Inference

Evidence Given : {hear-bark=true}

Probability for family-out being false = 0.6580703936042327

Probability for family-out being true = 0.3419296063957672

Time Taken for running the algorithm (in ms) = 55
```

Figure 17: Snapshot of Gibbs Sampling Method on dog-problem

5 Simulation on Auto Generated Graphs of 10K nodes

In order to be able to test the working of my algorithm with upto 10000 nodes, I built a simulator which generates a random graph og given number of nodes. It also generates a random query and a random evidence and attempts to then run inference algorithms on the generated inference problem.
Since enumeration is exponential in number of variables, I provide two different ways of running the simulation. Details on how to execute these methods are provided in ReadMe.

a) **Comparing inference on auto generated graphs with all inference algorithms**
This method basically runs inference on auto generated graphs of all sizes from 5 to given node size and keeps track of the time taken by all inference algorithm on all node sizes and then generates the output as the list of time taken by each algorithm for each nodesize from 5 nodes till given maxnode size. For example if the node size given is 20, then the method would first generate a graph of 5 nodes , a random query and evidence and run all inference algorithms on this graph of 5 nodes. This would then be repeated for 6, 7 ,...., 20(which will be provided as input). Figure 18 provides a sample output of running all 4 inference algorithms on auto generated graphs of node sizes 5-20 nodes. One can clearly see how enumeration takes huge amount of time as node size increases.

b) **Running inference on an auto generated graphs with a inference algorithm**
This method runs the specified algorithm on auto generated graph of a specified node size. Unlike previous method this is more of an absolute run than a comparative run. I generate a random graph for even sizes of 10K nodes and ask a random query backed by some random evidence. Figure 19 provides a sample output of one such run for gibbs sampling with a graph of 10000 nodes. This kind of graph would never be possible to run inference on using enumeration, but using gibbs sampling, its really easy to approximate the distribution. Do note that approximate inference is run with 1000000 samples and a burn in of 10000

6 Experimental Analysis

In this section, I analyze the different algorithms based on different measures of performance and draw a comparison between each of the methods. Specifically I analyze the algorithms on the following aspects.

```
Running simulation with node count of 5
Running simulation with node count of 6
Running simulation with node count of 7
Running simulation with node count of 8
Running simulation with node count of 9
Running simulation with node count of 10
Running simulation with node count of 11
Running simulation with node count of 12
Running simulation with node count of 13
Running simulation with node count of 14
Running simulation with node count of 15
Running simulation with node count of 16
Running simulation with node count of 17
Running simulation with node count of 18
Running simulation with node count of 19
Running simulation with node count of 20


Results are for the following node counts
[5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]
============================


Enumeration Timings in ms
--------------------------
[10, 0, 3, 173, 0, 38, 34, 9, 8, 36, 0, 0, 32, 2314, 4734, 523]


Likelihood Weighting Timings in ms
----------------------------
[62, 21, 71, 70, 8, 38, 31, 27, 27, 31, 20, 34, 27, 41, 38, 29]


Rejection Sampling Timings in ms
----------------------------
[36, 19, 47, 59, 7, 35, 32, 23, 27, 33, 21, 32, 29, 33, 37, 30]


Gibbs Sampling Timings in ms
----------------------------
[12, 14, 26, 19, 6, 13, 13, 9, 8, 8, 7, 10, 9, 10, 10, 5]
```

Figure 18: Output of inference on auto generated graphs of node sizes ranging from 5 to 20.

```
Running simulation with node count of 10000
Generated a random query for node variable named 9539
Generated a random evidence for query : {8255=true, 1760=false, 7978=false}


Results are for the following node counts
[10000]
============================


Timings in ms for Gibbs Sampling Approximate Inference
-------------------------
[3006]


Distribution inferenced : {true=0.6218100832688905, false=0.3781899167311094}
```

Figure 19: Output of inference on an auto generated graph of 10000 nodes using gibbs sampling

```java
/**
 * Utility method to find the information lost
 * when a given distribution is to be replaced by its
 * approximate distribution.
 * @param dist1 - Observed distribution
 * @param dist2 - Approximate distribution
 * @return
 */
public static double KLDivergence(Map<String, Double> dist1, Map<String, Double> dist2) {
    double KLDivergence = 0.0;
    for(String key : dist1.keySet()) {
        //Adding epsilon to avoid zero values
        double value1 = dist1.get(key) + 0.0000001;
        double value2 = dist2.get(key) + 0.0000001;
        KLDivergence += value1 * Math.log( value1 / value2 );
    }
    //return information lost in bits
    return KLDivergence / Math.log(2);
}
```

Figure 20: Snapshot of KL Divergence Method

a) Compare the Approximate methods on the basis of the **KL divergence distance between the distribution** achieved by the approximate method and that achieved by enumeration(exact inference). This can be thought of a measure of accuracy.

b) Compare all the methods on the basis of **time taken for the algorithm** to run.

c) Compare the methods on the **basis of memory consumption**.

d) Evaluate the **effect of burn-in samples on Gibbs Sampling** Method

6.1 Accuracy of the Approximate Inference Methods

In order to measure the accuracy of each of the approximate inference methods, I use the KL Divergence method. Kullback-Leibler(KL) Divergence(p,q) can be thought of as the amount of information in bits one would lose if they would approximate a distribution p with another distribution q. This is actually just what we would want as a measure of accuracy. We are using the approximate inference methods as a proxy for the exact inference method because of the time limitations it comes with. This will give us a measure of how much we would lose by using the approximate inference method. KL divergence is given by the below mentioned formula

$$KL(p||q) = \int_{\infty}^{\infty} p(x)log(\frac{p(x)}{q(x)})dx$$

If we use the log base as 2 this would give us the bits of information lost. Our goal is to get this loss to as small as possible. I provide my implementation of KL divergence score in Figure 20

6.1.1 Experiment Setup

I test the accuracy of the approximate inference methods for two types of events one being very rare whereas one being not so rare. I execute all three approximate inference methods for a range of samples from 100-10,000. For each run, I calculate the distance between

```
private void evaluateInferenceAlgorithm() throws ParserConfigurationException, SAXException, IOException {
    String query = "J";
    String FILE_DIRECTORY_PATH = "src/com/uofr/course/csc442/hw/hw3/examples/";
    String filePath = null;
    Map<String, String> evidence = new HashMap<String, String>();
    evidence.put("E", "true");
    filePath = FILE_DIRECTORY_PATH + "aima-alarm.xml";
    BayesianNetwork network = BIFFileParser.getNetworkFromFile(new File(filePath));
    List<String> averageKLD = new ArrayList<String>();
    List<Integer> sample = new ArrayList<Integer>();
    List<Integer> time = new ArrayList<Integer>();
    int stepSize = 100;
    int maxIter = 100;

    double KLDSum = 0.0;
    long startTime = System.currentTimeMillis();
    for(int rep = 1; rep<=100; rep++) {
        KLDSum += BayesianNetworkUtils.KLDivergence(new Enumeration().ask(query, evidence, network),
                new GibbsSampling(1).ask(query, evidence, network));
    }
    averageKLD.add(Double.toString(KLDSum*100));
    time.add((int) (System.currentTimeMillis() - startTime));
    sample.add(1);
    stepSize = 200;
    for(int i=200; i<=10000; i += stepSize) {
        KLDSum = 0.0;
        startTime = System.currentTimeMillis();
        for(int rep = 1; rep<=100; rep++) {
            KLDSum += BayesianNetworkUtils.KLDivergence(new Enumeration().ask(query, evidence, network),
                    new GibbsSampling(i).ask(query, evidence, network));
        }
        averageKLD.add(Double.toString(KLDSum*100));
        time.add((int) (System.currentTimeMillis() - startTime));
        sample.add(i);
    }

    System.out.println("KL Divergence score for comparing Enumeration and Gibbs Sampling with respect to number of samples");
    System.out.println("==================================================");
    System.out.println("\n");
    System.out.println("Sample Counts");
    System.out.println(sample);
    System.out.println("\n");
    System.out.println("KL Divergence scores");
    System.out.println(averageKLD);
}
```

Figure 21: Snapshot of Inference Simulator Method

the probability distribution returned by enumeration and that returned by approximate inference. The scenarios that I tested are the following:

a) Calculating the probability distribution of John calling in alarm example given that alarm is false.

b) (Rare event) Calculating the probability of John calling given that earthquake is true (this is rare because earthquakes are rare)

I created an InferenceSimulator which does the this task. I provide the snapshot for doing this experiment for gibbs sampling method in Figure 21

I first evaluate the methods conditioned on alarm being true and query being john calling. Figure 22 provides the plot of the KL divergence distance on a scale of 1E-4 vs the number of samples.

Takeaways from this plot:

a) Initially Gibbs sampling has a huge distance from true distribution. This is because the Gibbs Sampling method requires few 1000 samples to reach to steady state.

b) Since the event is not rare, rejection sampling has no issue being able to gather the samples for this event

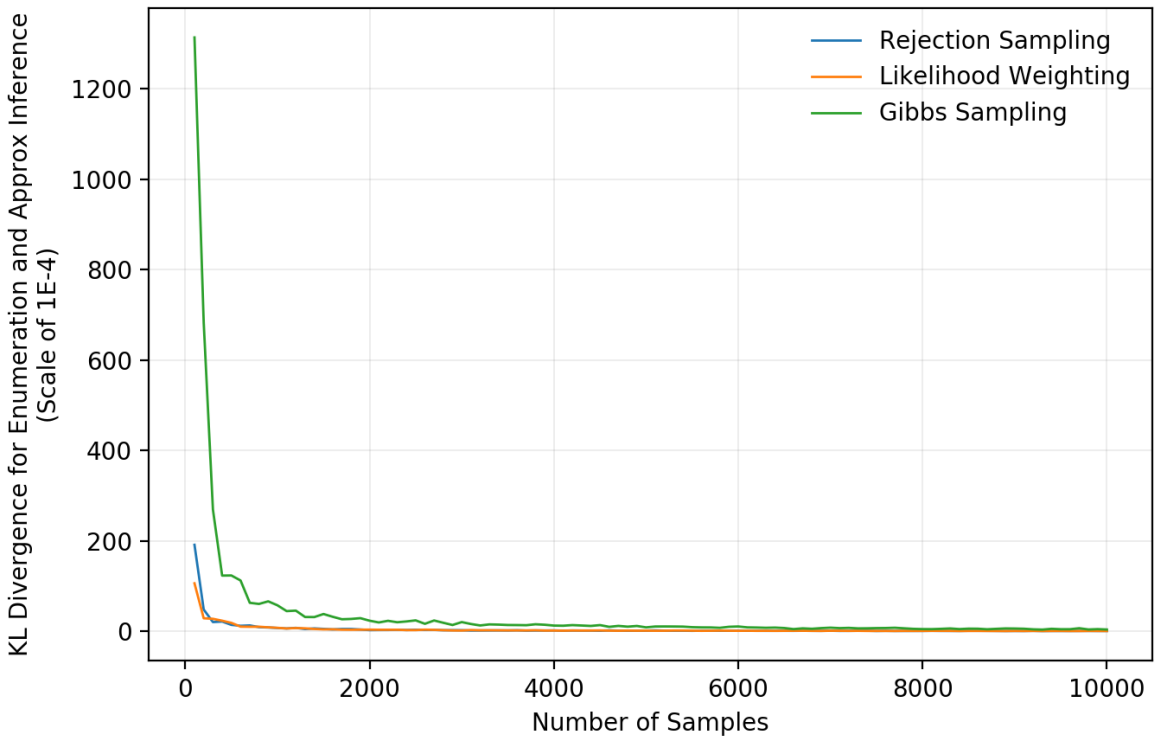

Figure 22: Accuracy of Approx Inference for John calls given Alarm is false

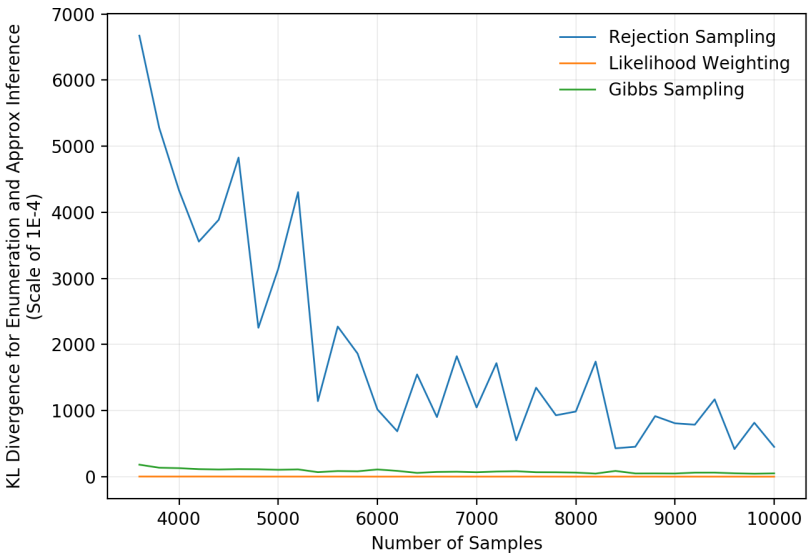

Figure 23: Accuracy of Approx Inference for John calls given Earthquake is true(Rare)

c) Given enough samples all approximate inference algorithm converge to the true distribution for this event.

Figure 23 provides the plot of the KL divergence distance on a scale of 1E-4 vs the number of samples for the rare event of earthquake being true and the query being John calling.

Takeaways from this plot:

a) Because the event of earthquake is so rare(0.002), even after 10K samples. rejection sampling is not able to be as close to true distribution as other methods. This is just because its pretty much rejecting all the samples.

b) Given enough samples likelihood and gibbs sampling algorithm converge to the true distribution for this rare event. Even rejection sampling is converging as its getting more samples, and when we power it up to 100K samples it indeed converges to the true distribution.

6.2 Time taken for the Inference Methods

In order to measure the time taken by each inference method we just keep track of execution times for each iterations and see the amount of time it took for each iteration.

6.2.1 Experiment Setup

I evaluate the inference methods on sample sizes sizes ranging from 1 to 10K with a step size of 200. For each sample size, I run each inference method 100 times in order to get a more

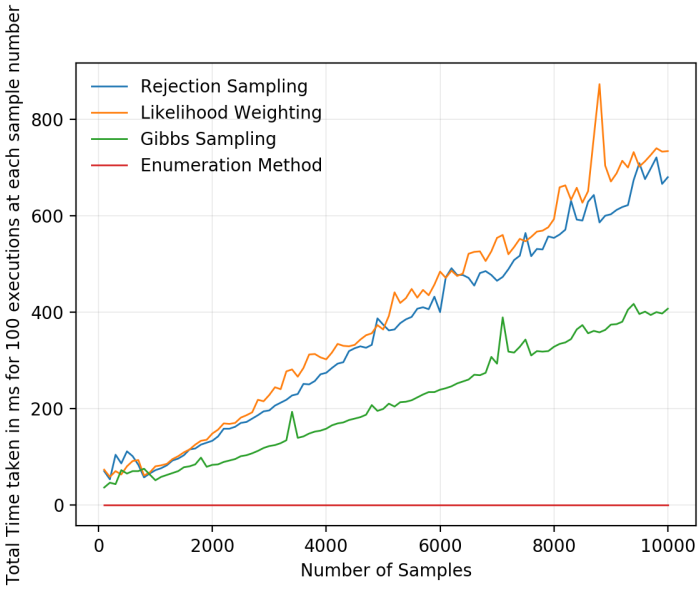

Figure 24: Time taken for Inference methods for simple network (aima-alarm)

accurate estimate of time. The scenarios that I tested are the following:

a) Time taken for a simple graph like aima-alarm example.

b) Time taken for complex graphs with more than 5 nodes.

I first evaluate the methods on the aima-alarm network with the evidence of alarm being false and query being that John calls. Figure provides the plot of time taken for inference methods for the alarm example.

Takeaways from this plot:

a) Enumeration is the quickest. The reason this is because one the graph is very simple and second enumeration does not really have any effect of sample sizes. So its timing would just be constant with respect to sample sizes. We would soon see how enumeration completely blows up when graph is complex.

b) Gibbs Sampling ends up taking lesser time than other two approximate inference methods but we can see that all the approximate inference methods increase linearly with number of samples required.

In order to find **how the algorithms behave once the graph becomes complex**, I built a graph generator which would build a graph of a desired number of nodes and I use that to find out how the algorithms behave when the node size increases to larger values. Figure provides the plot of the time taken for the inference methods vs the size of the graph(number of nodes). Once the graph is generated I randomly select for a bunch of nodes and assign them evidence values randomly and then I select a query node randomly. Do note that the time taken is in log scale.

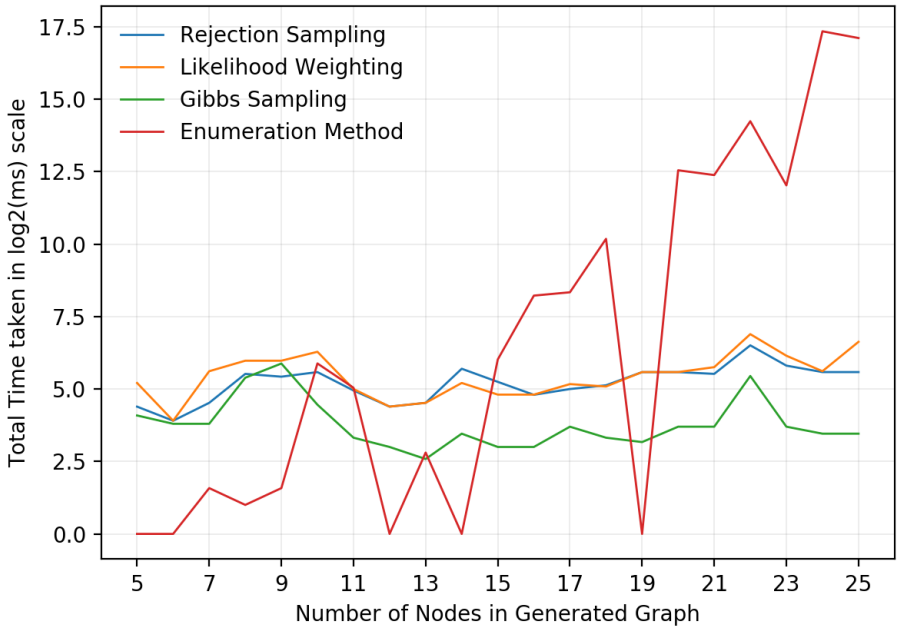

Figure 25: Time taken for Inference methods vs number of nodes in Graph

Takeaways from this plot:

a) Since the log of time taken by enumeration increases linearly with number of nodes, it means that actual time taken increases exponentially with the number of nodes.

b) For other approximate inference methods, the time taken is a straight line which means the only complexity involved with them is the number of samples and they scale really well with respect to number of nodes in network.

6.3 Effect of Burn-in on Gibbs Sampling

In this section I delve briefly into the aspect of burn in and its impact on Gibbs Sampling. As we know that Gibbs sampling reaches into the steady state of true distribution allowing us to get a good approximate inference results for any query on a given network. However an important question to ask is WHEN does it reach its steady state. In practical applications we would have to do this by using our returned distribution on some other goal based task and then waiting for more samples if our accuracy on that task is poor(assuming that we still have not reached the steady state. However, here, given that we know the true distribution of a query using enumeration method, I use that to find out if we can figure out the right burn-in period for gibbs sampling.
We use the Aima-alarm example as an aspect to measure this effect.

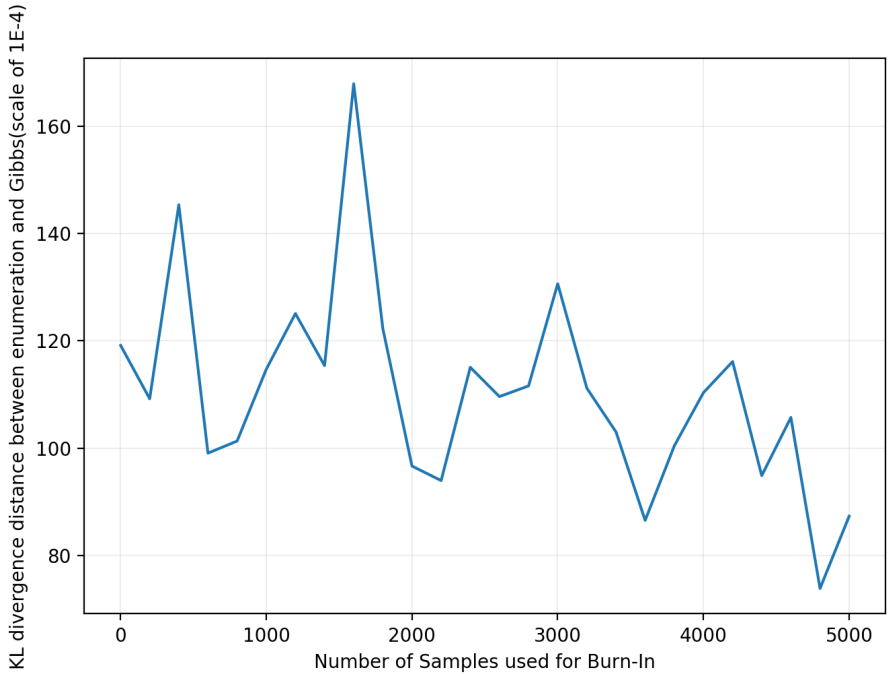

Figure 26: Plot of Accuracy of Gibbs versus BurnIn Quantity

Query being John calling and evidence same as before being Alarm being false.

6.3.1 Experiment Setup

I modify the Inference simulator mentioned earlier to instantiate the gibbs sampling method with a burn in period and then use the same setup as was used to evaluate accuracy. I test for burn-in samples(K) being in the range of 0 to 5000. For each burn-in sample quantity(K), I reject the first K samples and then evaluate the KL divergence of the distribution approximated by Gibbs Sampling using the next 2000 samples. So in effect, if my burn in quantity was 2000, I would generate 4000 samples, reject the first 2000 out of them and evaluate the KL divergence distance between enumeration and Gibbs sampling based on the remaining accepted 2000 samples.
Figure provides the plot of KL divergence score

Takeaways from this plot:

a) As one can clearly see from the plot, the accuracy of gibbs sampling is better once you have waited for more number of samples as your burn in period.

b) The graph goes through spikes because the nodes are being chosen at random and hence it can happen that it takes more samples to reach the same accuracy.

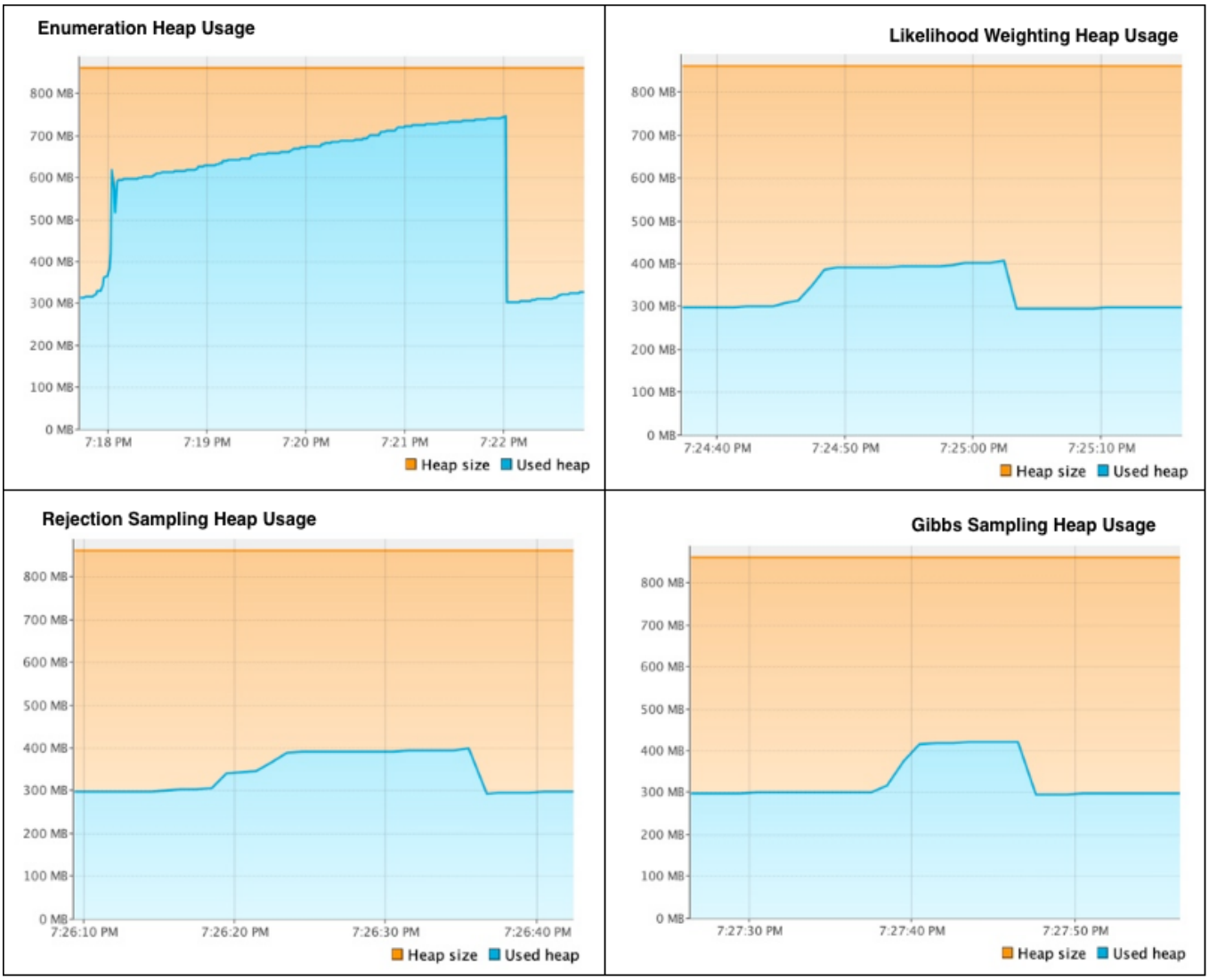

Figure 27: Memory Usage Comparison of Inference Methods on a 20 Node Network

6.4 Memory usage of algorithms

In this section I dive deep into analyzing the heap memory consumption of my java code for all the inference algorithms.

6.4.1 Experiment Setup

I create a network of 20 nodes and run all the inference methods and monitor the heap memoey usage using visualVM tool. I ensure that GC(garbage collection) happens before the start of program so that all the programs start at the same point of heap space.

Figure 27 provides plot of memory usage comparison of all algorithms.

Takeaways from this plot:

Sample	*True Dist(F,T)*	*Gibbs*	*Likelihood*	*Rejection*
2000	0.7158, 0.284	0.6493,0.3506	0.8481,0.15189	0.4,0.6
4000	0.7158, 0.284	0.67577,0.32433	0.6906,0.3093	0.666,0.3333
6000	0.7158, 0.284	0.6974605,0.30253	0.7700,0.2299	0.6923,0.3076
8000	0.7158, 0.284	0.70518,0.29481	0.7341,0.2658	0.6875,0.3125
10000	0.7158, 0.284	0.71269,0.28730	0.7326,0.2673	0.7,0.3

Table 1: Convergence of Distribution for Alarm Problem 1 with Samples

a) Enumeration method for larger network(here the node count was 20) leads to a huge memory usage of about 400 MB. The drop in the heap is actually garbage collection that happened by JVM post running of the program.

b) For all other approximate inference methods roughly similar amount of memory usage can be observed(close to 100MB in this istance).

7 Sample Problems and Results of Inference

In this section I pick up 2 sample problems from each of the example bif files provided and compare the accuracy of the approximate inference methods and the time taken for approximate inference algorithms. In general the obvious trend of reduction in error with more and more samples can be observed for all approximate inference algorithms.

7.1 AIMA-ALARM

7.2 Problem 1:

Query : Burglary is happening
Evidence : John is calling and Mary is calling

Table 1 demonstrates the convergence of probability distributions achieved with increase in samples for all approximate inference methods. A plot of error between the approximate and true distribution can be seen in Figure 28

7.3 Problem 2:

Query : Mary is calling
Evidence : Alarm is true, Burglary is true and Earthquake is false

Table 2 demonstrates the convergence of probability distributions achieved with increase in samples for all approximate inference methods. A plot of error between the approximate and true distribution can be seen in Figure 28

Sample	*True Dist(T,F)*	*Gibbs*	*Likelihood*	*Rejection*
2000	0.7, 0.3	0.7702,0.22977	0.7195,0.2805	0.5,0.5
4000	0.7, 0.3	0.7017,0.298	0.7112,0.2887	0.666,0.3333
6000	0.7, 0.3	0.7010,0.2989	0.7123,0.2876	0.75,0.25
8000	0.7, 0.3	0.6923,0.30767	0.7102,0.28975	0.75,0.25
10000	0.7, 0.3	0.6927,0.3072	0.70779,0.2922	0.7272,0.2727

Table 2: Convergence of Distribution for Alarm Problem 2 with Samples

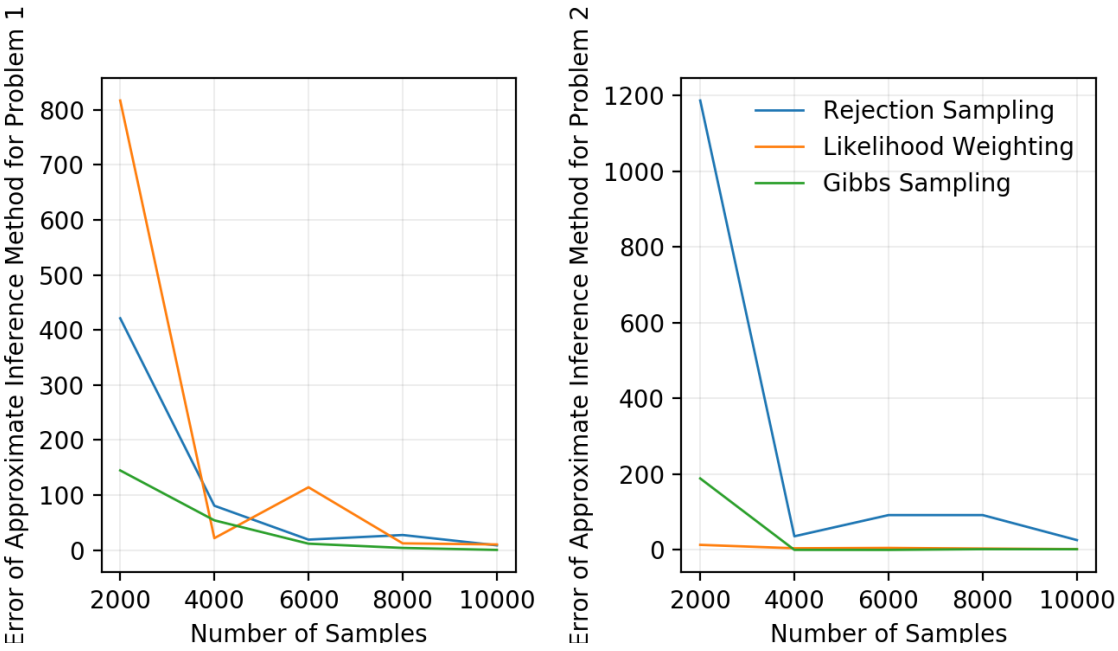

Figure 28: Plot of Error for Approximate Inference with samples on Alarm Problems

Sample	*True Dist(T,F)*	*Gibbs*	*Likelihood*	*Rejection*
2000	0.7079, 0.2920	0.63436,0.365634	0.69230,0.307692	0.7274,0.2725
4000	0.7079, 0.2920	0.5851,0.414861	0.6944770,0.305522	0.7135,0.2864
6000	0.7079, 0.2920	0.63907,0.36092	0.699422,0.30057	0.711,0.2888
8000	0.7079, 0.2920	0.66133,0.3386	0.6996,0.30036	0.71055,0.2894
10000	0.7079, 0.2920	0.67848,0.321519	0.70159,0.29840	0.70819,0.2918

Table 3: Convergence of Distribution for Wet Grass Problem 1 with Samples

Sample	*True Dist(F, T)*	*Gibbs*	*Likelihood*	*Rejection*
2000	0.82, 0.18	0.78421,0.2157	0.805347,0.19465	0.831663,0.16833
4000	0.82, 0.18	0.8020,0.19793	0.80748,0.19251	0.8310,0.1689
6000	0.82, 0.18	0.81943,0.180563	0.811605,0.1883	0.82571,0.17428
8000	0.82, 0.18	0.8188,0.18111	0.81383,0.1861	0.82434,0.1756
10000	0.82, 0.18	0.81802,0.18197	0.8149,0.18504	0.82058,0.17941

Table 4: Convergence of Distribution for Wet Grass Problem 2 with Samples

7.4 AIMA-WET-GRASS

7.5 Problem 1:

Query : Rain
Evidence : Grass is wet

Table 3 demonstrates the convergence of probability distributions achieved with increase in samples for all approximate inference methods. A plot of error between the approximate and true distribution can be seen in Figure 29

7.6 Problem 2:

Query : Sprinkler is on
Evidence : Rain is true

Table 4 demonstrates the convergence of probability distributions achieved with increase in samples for all approximate inference methods. A plot of error between the approximate and true distribution can be seen in Figure 29

7.7 DOG-PROBLEM

7.8 Problem 1:

Query : family-out
Evidence : hear-bark is True

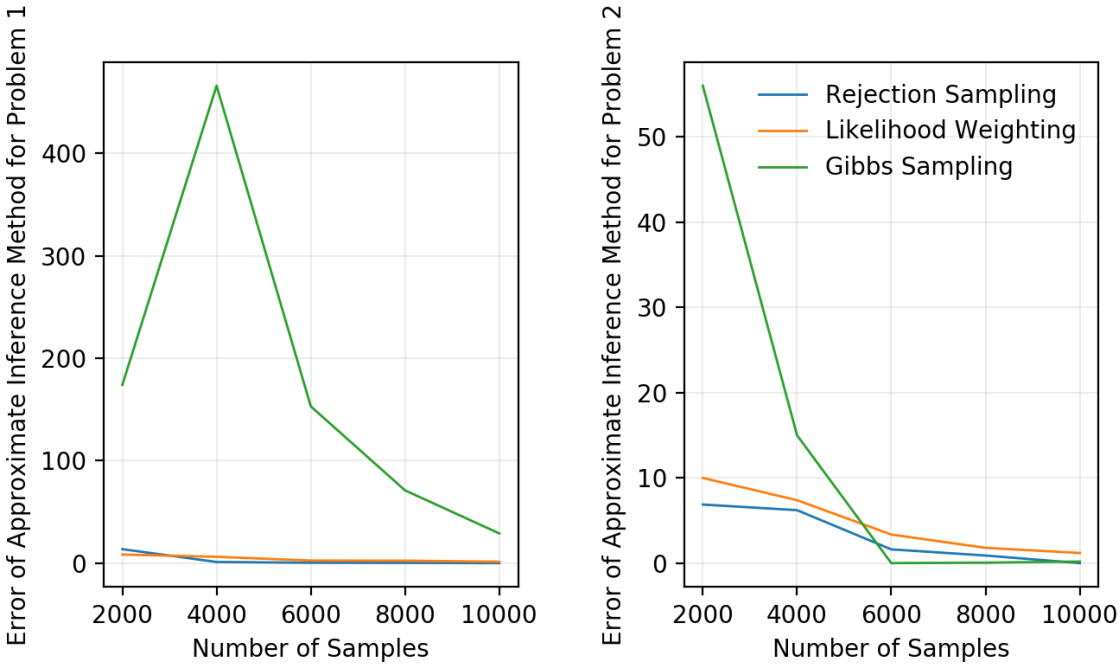

Figure 29: Plot of Error for Approximate Inference with samples on Wet Grass Problems

Sample	*True Dist(F, T)*	*Gibbs*	*Likelihood*	*Rejection*
2000	0.66536, 0.33463	0.690309,0.309690	0.630988,0.369011	0.61552,0.38447
4000	0.66536, 0.33463	0.6814,0.31856	0.64830,0.35169	0.6264,0.37355
6000	0.66536, 0.33463	0.67406,0.32593	0.6499,0.35003	0.62432,0.37567
8000	0.66536, 0.33463	0.67561,0.3243	0.651977,0.3480	0.6432,0.3567
10000	0.66536, 0.33463	0.657149,0.34285	0.65518,0.34481	0.6470,0.35294

Table 5: Convergence of Distribution for Dog Problem 1 with Samples

Table 5 demonstrates the convergence of probability distributions achieved with increase in samples for all approximate inference methods. A plot of error between the approximate and true distribution can be seen in Figure 30

7.9 Problem 2:

Query : light-on
Evidence : bowel-problem is true

Table 6 demonstrates the convergence of probability distributions achieved with increase in samples for all approximate inference methods. A plot of error between the approximate and true distribution can be seen in Figure 30

Sample	*True Dist(F, T)*	*Gibbs*	*Likelihood*	*Rejection*
2000	0.8675, 0.1325	0.85114,0.148851	0.881500,0.118499999	0.789473,0.21052
4000	0.8675, 0.1325	0.87704,0.12295	0.871250,0.128749	0.8571428,0.142857
6000	0.8675, 0.1325	0.87502,0.1249	0.871666,0.1283333	0.846153,0.15384
8000	0.8675, 0.1325	0.8587,0.141265	0.87112,0.128875	0.8593,0.140625
10000	0.8675, 0.1325	0.867014,0.13298	0.870400,0.1295999	0.870129,0.129870

Table 6: Convergence of Distribution for Dog Problem 2 with Samples

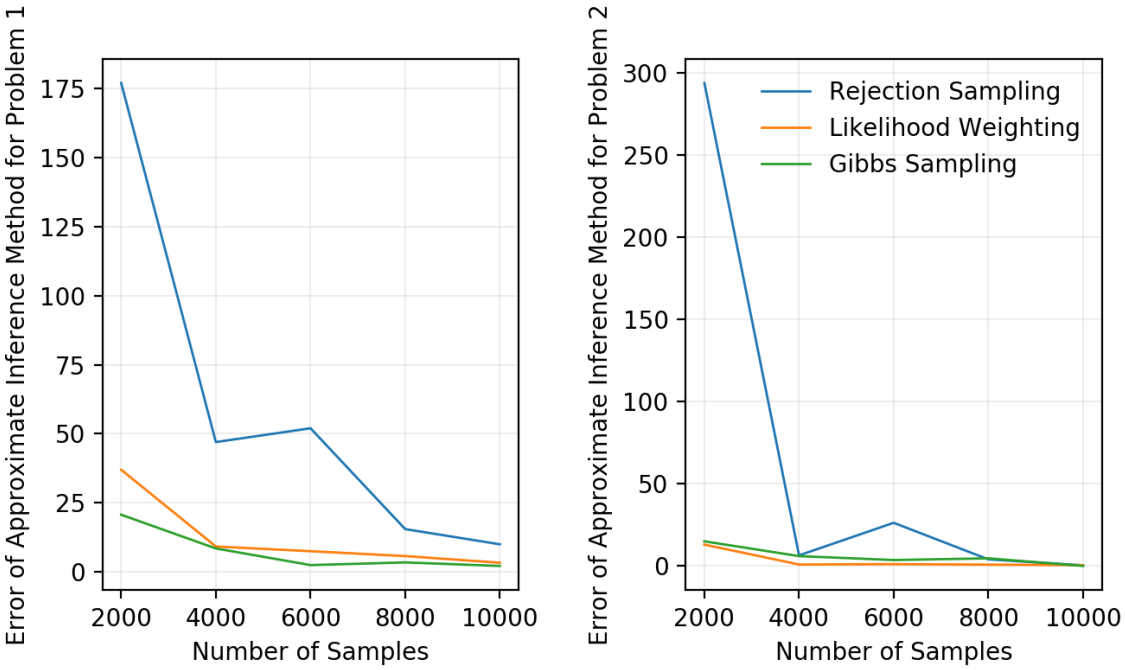

Figure 30: Plot of Error for Approximate Inference with samples on Dog Problems