Chen Zhang
CSC 242 Artificial Intelligence
Project 3
Due Date: 04/15/16
No partners

Exact Inference and Inexact Inference on Bayesian Network

Introduction

Behaving under uncertainly is always a difficult endeavor for both human beings and A.I due to unpredictable future. In this project, our main goal is to develop/implement algorithms to teach A.I to solve those problems based on Bayesian network. Bayesian network is a directed graph that can be used to represent quantitative probability information. A typical Bayesian network usually consists one or more nodes that corresponds o a random variable; a set of directs links can be interpreted as parent and children relationship. Additionally, no cycles are allowed. Overall, each node has a conditional probability distribution P(X| parent(X). Within Bayesian network, exact inference and inexact inference are implemented to predict the probability of a variable in a given observed phenomena.

Basic Concepts for Inference by enumeration

Recall that, our goal here is to compute the posterior probability distribution for a set of query variable given observed evidence variables. (In this implementation, it's only one query variable but it's easy to implement to a set of variable by using an array of query variables)

In enumeration method, it basically calculated the summing terms from the full joint distribution in the following equation:

$$P(X \mid e) = \alpha P(X, e) = \alpha \sum_{y} P(X, e, y) \quad \text{Eq.1}$$

where the X is the query variables, e is observed evidence, and y is unobserved variables. Now a query can be answered using a Bayesian network by computing sums of products of each node at network (conditional probabilities). For instance, if a query is to calculate probability of burglary (B) given John and Mary both (j, b) calls you can be expressed in Eq.2

$$P(B \mid j, m) = \alpha P(B, j, m) = \alpha \sum_{e} \sum_{a} P(B, j, m, a, e) \quad \text{Eq.2}$$

where e is the earthquake and a stands for alarm.

Figure 1 describes enumeration algorithm form the textbook(Figure 14.9). For *function*

The Enumeration Algorithm

```
function ENUMERATION-ASK(X, e, bn) returns a distribution over X
  inputs: X, the query variable
          e, observed values for some set of variables E
          bn, a Bayes net
  Q ← a distribution over X, where Q($x_i$) is P($X=x_i$)
  for each value $x_i$ that X can have do
      Q($x_i$) ← ENUMERATE-ALL(bn.VARS, $e_{x_i}$), where $e_{x_i}$ is the evidence e plus the assignment $X=x_i$
  return NORMALIZE(Q)

function ENUMERATE-ALL(vars, e) returns a probability (a real number in [0,1])
  inputs: vars, a list of all the variables
          e, observed values for some set of variables E
  if EMPTY(vars) then return 1.0
  Y ← FIRST(vars)
  if Y is assigned a value (call it y) in e then
      return P(Y=y | values assigned to Y's parents in e) × ENUMERATE-ALL(REST(vars), e)
  else
      return $\sum_{y_i}$ [P($Y=y_i$ | values assigned to Y's parents in e) × ENUMERATE-ALL(REST(vars), $e_{y_i}$)],
             where $e_{y_i}$ is the evidence e plus the assignment $Y=y_i$
```

Figure 1. The enumeration algorithm

ENUMERATE-ALL, I slightly revised the *if else* indent and summation to make this algorithm easier to understand than the original book one. The thing we need to play attention to is the helper function ENUMERATE-ALL. At each call of this function, we look at the first variable Y from vars (if there is no Y left, it is the break case of recursion just return 1.00). There are two cases. In the first case, Y is already assigned in e to some value y, so P(e) is just P(Y=y | the rest of e) × P(the rest of e). In the second case, Y is not assigned, so we have to sum over all possible values y_i in Y's domain.

Implementation for Inference by enumeration (BNExactInferencer.java)

Since Prof. Ferguson provided his schema for Bayesian network and other *util* class, I would like to emphasize my own codes and how I used/interpreted his codes.

First of all, I need to read from input files. In this project, we have .XML and .BIF. I wrote a simple if-else statement to check if the first argument (args[0]) contains ".xml". If it's the case, simply call *XMLBIFParser* and use this parser to read network from input file. If not, just call BIFParser and do the same thing. And an arraylist(RVlist) was used to contain all the topologically sorted random variables from BN. After this, a for loop was applied to convert other command line arguments into queries and evidences. A if statement was wrote to first check if the arg is in RVlist: if it is, check the next arg whether in RV domain, if both are true, it is evidence; other wise it must be a query variable. (Note: this if statement can be avoided if we can ask the user ALWAYS put args[1] as query variable and remaining are the evidence variables. If I did this way, this should be even faster and easier. However, my implementation is more generic and it can be easier to extend to multiple query variables instead of one variable).

Now I have evidences (*assignment class*), query (random variable class) and a list of random variables (topologically sorted). In my implementation of *enumerationAll()* method, it takes into a random variable list and an assignment. Getting the next random variable means getting the 0 index item from RVlist. Then use *evidence.variableSet().contains()* to check the next random variable is in evidence list. If so, go ahead to multiple P(FirstRV=y | the rest of e) × P(the rest of e). Otherwise, sum over the possible unobserved values. Another helper function *rest()* returns random variables from index 1 to the size.

In *ask()* method, I iterated over query variable domain and call *enmerateAll* function by passing the evidence into it. A nontrivial point is that the evidence list is different for different domain because new evidence must be added to the list for the specific domain. Last, create a distribution object to contain query variable domain and its corresponding probability.

Basic Concepts for Inexact Inference

Rejection sampling is a basic technique used to generate observations from a distribution. First, it generates samples from the prior distribution (directed sampling) in a Bayesian network. Then, it check if the sample is consistent with evidence. Finally, one can estimate P (X = x | e) is obtained by counting how often X = x occurs in the remaining samples. As more samples are random generated, the estimate will converge to the true answer. This algorithm is shown in Figure 2, where generating directed sample is in Figure 3

```
function REJECTION-SAMPLING($X$, $\mathbf{e}$, $bn$, $N$) returns an estimate of $\mathbf{P}(X|\mathbf{e})$
  inputs: $X$, the query variable
          $\mathbf{e}$, observed values for variables $\mathbf{E}$
          $bn$, a Bayesian network
          $N$, the total number of samples to be generated
  local variables: $\mathbf{N}$, a vector of counts for each value of $X$, initially zero

  for $j$ = 1 to $N$ do
      $\mathbf{x} \leftarrow$ PRIOR-SAMPLE($bn$)
      if $\mathbf{x}$ is consistent with $\mathbf{e}$ then
          $\mathbf{N}[x] \leftarrow \mathbf{N}[x]+1$ where $x$ is the value of $X$ in $\mathbf{x}$
  return NORMALIZE($\mathbf{N}$)
```

Figure 2 Rejection Sampling Algorithm(Figure 14.14 from AIMA)

```
function PRIOR-SAMPLE($bn$) returns an event sampled from the prior specified by $bn$
  inputs: $bn$, a Bayesian network specifying joint distribution $\mathbf{P}(X_1, \ldots, X_n)$

  $\mathbf{x} \leftarrow$ an event with $n$ elements
  foreach variable $X_i$ in $X_1, \ldots, X_n$ do
      $\mathbf{x}[i] \leftarrow$ a random sample from $\mathbf{P}(X_i \mid parents(X_i))$
  return $\mathbf{x}$
```

Figure 3 Prior Sample Algorithm (Figure 14.13 from AIMA)

The idea behind prior sample is to sample each variable at one time and make sure assigning to the conditional variable's parents value based on previously generating value.

Implementation for Inexact Inference

For *RejectionSampling.java*, the main method is similar to *BNExactInferencer.java*, the only difference is converting one more arguments (args[0]) to number of samples. In *priorSample()* method, it returns an assignment that is random generated based on probability. It iterates over the entire random variables (rvlist), and generate domain for each variable by calling a helper function *generateRandomAssigment()*. Actually, the most challenging of rejection sampling is this helper function. First, I defined a *range* arrayList that is used to create the numeric border of each domain. For example: if one random variable's domain is<snowing, 95%, cloudy 4%, sunny 1%>. The values of *range* are 0.95, 0.99, 1.0 respectively. Then, I generated a random number from 0.0 to 1.0. Inside a for loop, I check which border this random number falls into. If this random number is 0.98, then a domain cloudy is generated. Since the items in *range* arrayList are in ascending order, we can simply check if *randomNumber* less than *range.get(i)*: if yes, return that index and break the loop.

One note is that this method is returning an Object since I don't know the exact type my domain I return.

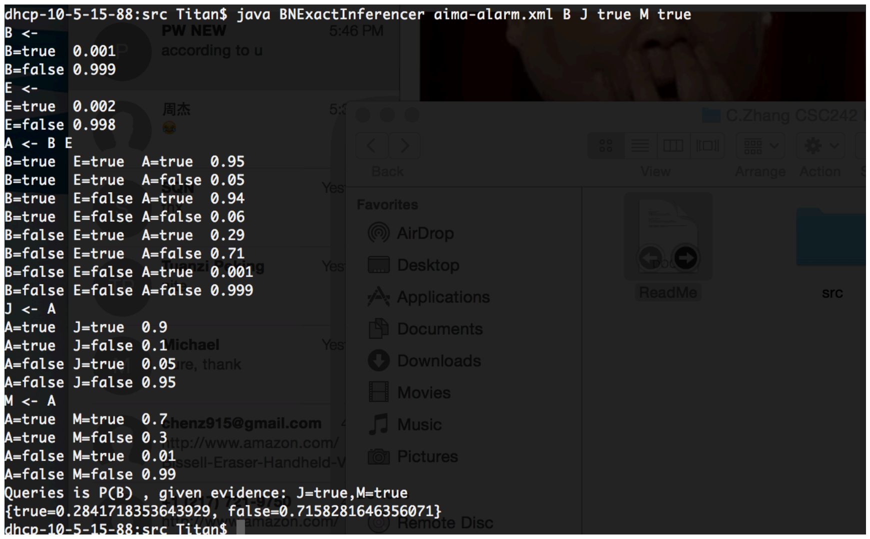
After obtaining the prior sample, I just follow the pseudo-code from Figure.2. Another helper function called isConsistent() was applied to check whether the random generated assignment is consistent with evidence. This is done in O(m+n) where m is size of assignment and n is the evidence size. What it does is to convert assignment/evidence entry into arrayList; then check assList.contains(eList). Finally, inside *ask()*, it loops through the number of argument times and count number of times that assignment matches with a specific domain. Similarly to *BNExactInference.java,* distribution class was called and used to normalize the distribution.

Bonus Points for LikelihoodWeighting.java

Likelihood is also implemented, please see read me for how to run Likelihood test. Likelihood implementation requires an inner class named *WeightedAssignment*. It is similar to assignment, but it has another parameter *weight(double type)* to keep track of the weight. Besides this inner class, another if-else was applied in *weightedSample()* method to check each node is in evidence list. If it is, the algorithm does NOT random generate a domain but set the domain as same as evidence. And keep accumulating the weights. After iterating each node, it returns a weightedAssignment object.

Result:

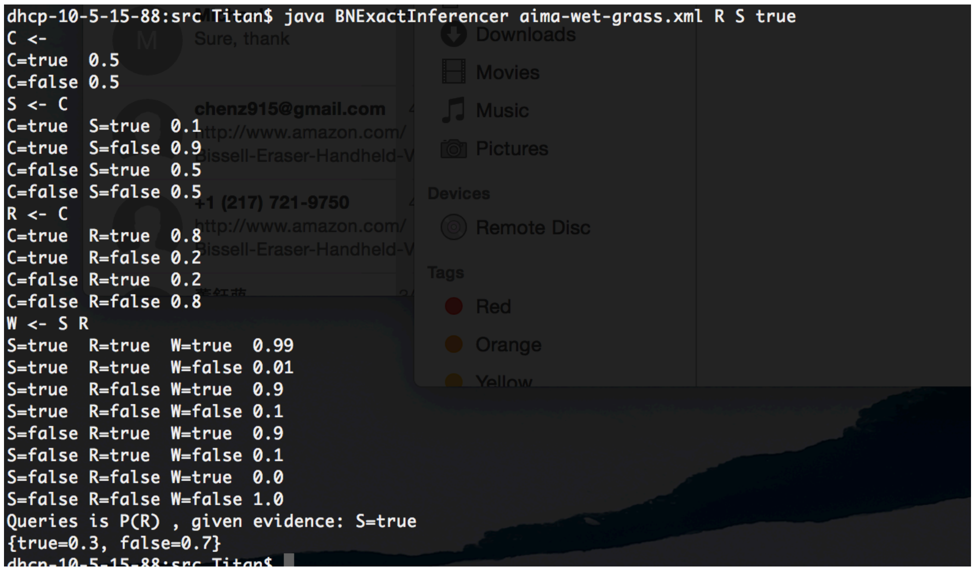
I only tested two simple cases AIMA-alarm and AIMA-wet grass by applying both exact and inexact inference. The reason is that I do not know correct probability for other Bayesian networks samples.



```
dhcp-10-5-15-88:src Titan$ java BNExactInferencer aima-alarm.xml B J true M true
B <-
B=true  0.001
B=false 0.999
E <-
E=true  0.002
E=false 0.998
A <- B E
B=true  E=true  A=true  0.95
B=true  E=true  A=false 0.05
B=true  E=false A=true  0.94
B=true  E=false A=false 0.06
B=false E=true  A=true  0.29
B=false E=true  A=false 0.71
B=false E=false A=true  0.001
B=false E=false A=false 0.999
J <- A
A=true  J=true  0.9
A=true  J=false 0.1
A=false J=true  0.05
A=false J=false 0.95
M <- A
A=true  M=true  0.7
A=true  M=false 0.3
A=false M=true  0.01
A=false M=false 0.99
Queries is P(B) , given evidence: J=true,M=true
{true=0.2841718353643929, false=0.7158281646356071}
dhcp-10-5-15-88:src Titan$
```

Figure 4. Exact Inference on AIMA-alarm.xml querying Burglary given John and Mary both call

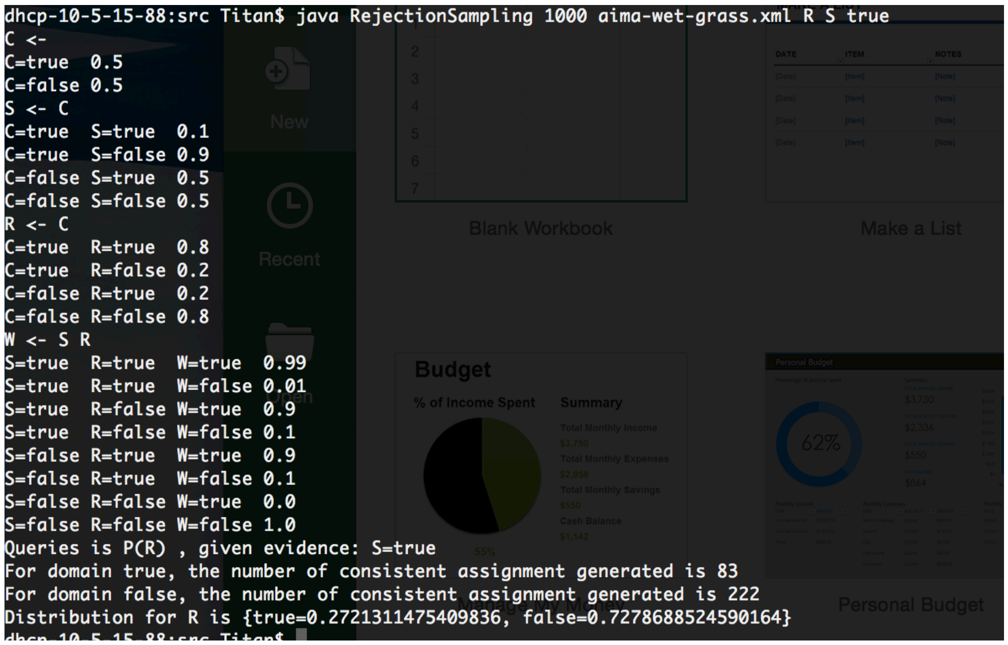
As shown in Figure 4, the probability of burglary happens when John and Mary both call is 0.28 and not a burglary is 0.72; which matches expected answer from the textbook.



```
dhcp-10-5-15-88:src Titan$ java BNExactInferencer aima-wet-grass.xml R S true
C <-
C=true  0.5
C=false 0.5
S <- C
C=true  S=true  0.1
C=true  S=false 0.9
C=false S=true  0.5
C=false S=false 0.5
R <- C
C=true  R=true  0.8
C=true  R=false 0.2
C=false R=true  0.2
C=false R=false 0.8
W <- S R
S=true  R=true  W=true  0.99
S=true  R=true  W=false 0.01
S=true  R=false W=true  0.9
S=true  R=false W=false 0.1
S=false R=true  W=true  0.9
S=false R=true  W=false 0.1
S=false R=false W=true  0.0
S=false R=false W=false 1.0
Queries is P(R) , given evidence: S=true
{true=0.3, false=0.7}
dhcp-10-5-15-88:src Titan$
```

Figure 5. Exact Inference on AIMA-wet-grass.xml querying Rainy given sprinkler is true

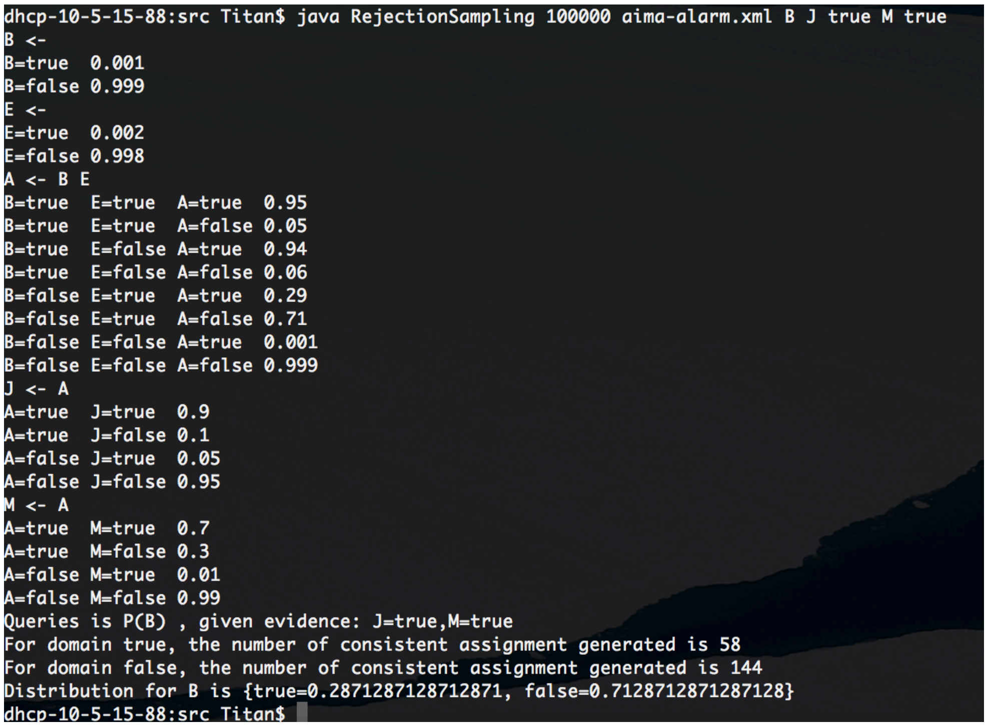
And the probability of rainy given sprinkler is true is 0.3 and not rainy is 0.7, which also matches the answer from textbook.



```
dhcp-10-5-15-88:src Titan$ java RejectionSampling 1000 aima-wet-grass.xml R S true
C <-
C=true  0.5
C=false 0.5
S <- C
C=true  S=true  0.1
C=true  S=false 0.9
C=false S=true  0.5
C=false S=false 0.5
R <- C
C=true  R=true  0.8
C=true  R=false 0.2
C=false R=true  0.2
C=false R=false 0.8
W <- S R
S=true  R=true  W=true  0.99
S=true  R=true  W=false 0.01
S=true  R=false W=true  0.9
S=true  R=false W=false 0.1
S=false R=true  W=true  0.9
S=false R=true  W=false 0.1
S=false R=false W=true  0.0
S=false R=false W=false 1.0
Queries is P(R) , given evidence: S=true
For domain true, the number of consistent assignment generated is 83
For domain false, the number of consistent assignment generated is 222
Distribution for R is {true=0.2721311475409836, false=0.7278688524590164}
```

Figure 6. Rejection sampling on AIMA-wet-grass.xml querying Rainy given sprinkler is true for 1,000 times

By sampling 1,000 times on AIMA-wet-grass example, total efficient samples are (222+83) = 305, and the distribution is 0.272 and 0.727.



```
dhcp-10-5-15-88:src Titan$ java RejectionSampling 100000 aima-alarm.xml B J true M true
B <-
B=true  0.001
B=false 0.999
E <-
E=true  0.002
E=false 0.998
A <- B E
B=true  E=true  A=true  0.95
B=true  E=true  A=false 0.05
B=true  E=false A=true  0.94
B=true  E=false A=false 0.06
B=false E=true  A=true  0.29
B=false E=true  A=false 0.71
B=false E=false A=true  0.001
B=false E=false A=false 0.999
J <- A
A=true  J=true  0.9
A=true  J=false 0.1
A=false J=true  0.05
A=false J=false 0.95
M <- A
A=true  M=true  0.7
A=true  M=false 0.3
A=false M=true  0.01
A=false M=false 0.99
Queries is P(B) , given evidence: J=true,M=true
For domain true, the number of consistent assignment generated is 58
For domain false, the number of consistent assignment generated is 144
Distribution for B is {true=0.2871287128712871, false=0.7128712871287128}
dhcp-10-5-15-88:src Titan$
```

Figure 7. Rejection sampling on AIMA-alarm.xml querying Burglary given John and Mary both call for 100,000

By sampling 100,000 times on AIMA-alarm example, total efficient samples are (148+54) = 202, and the distribution is 0.287 and 0.713.

Both rejection sampling examples actually yield pretty decent probability compared to exact inference. As indicated in Figure 8, generating 100 samples to 100,000 samples yield average value 0.295 for rainy given sprinkler is true.

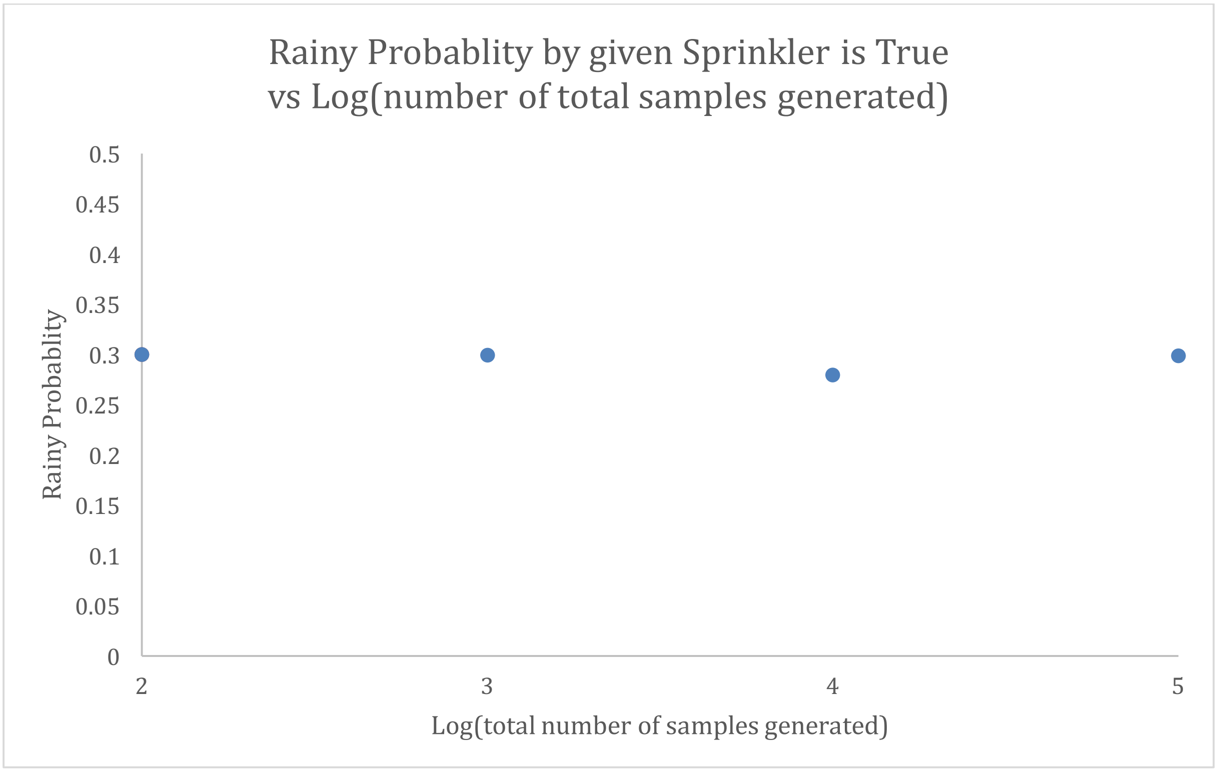


Figure 8. Rainy Probablity by given Sprinkler is True vs Log(number of total samples generated)

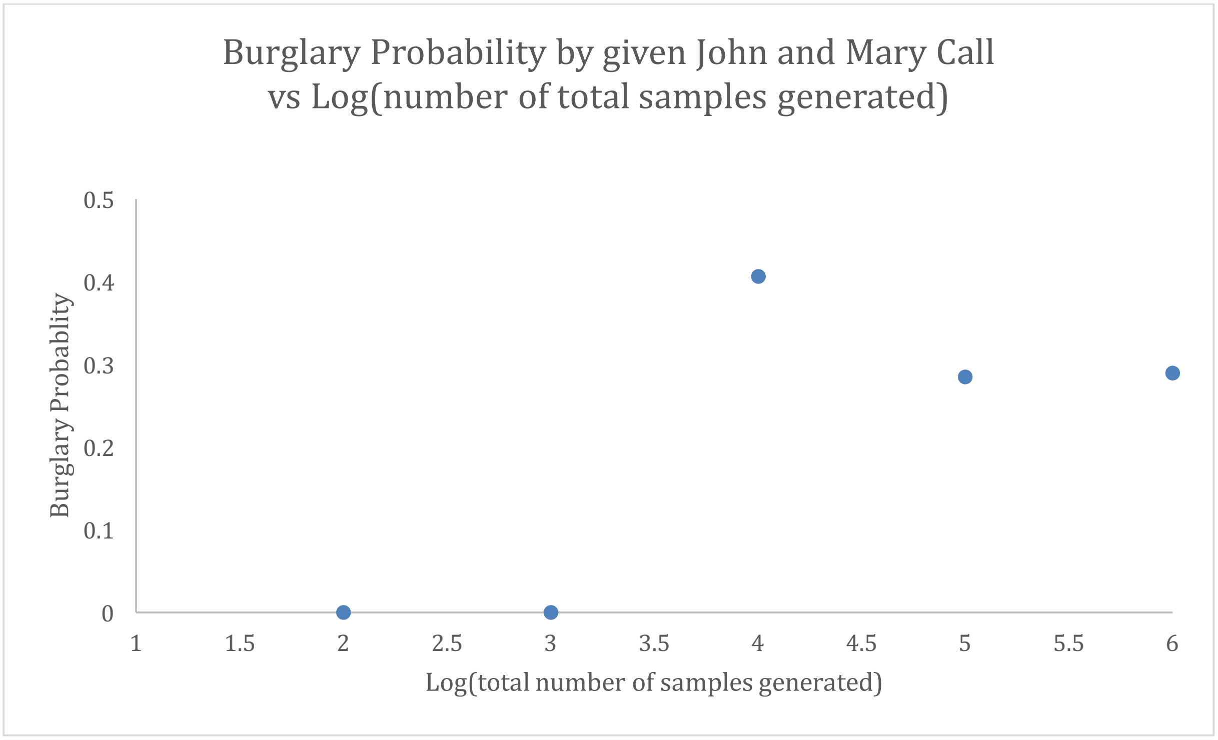


Figure 9. Burglary Probability by given John and Mary Call

vs Log(number of total samples generated)

Not surprisingly, generating only 100 to 10,000 samples are not enough for computing burglary probability because the chance for both John and Mary call you is extremely small. An analogy for thinking about this is how many times both of your parents have called you at same time for emergency, probably none. So in this case, our model was rejecting too many inconsistent samples. In another word, rejection sampling is inefficient in this model. Figure 10 shows number of consistent samples versus total number of samples generated.

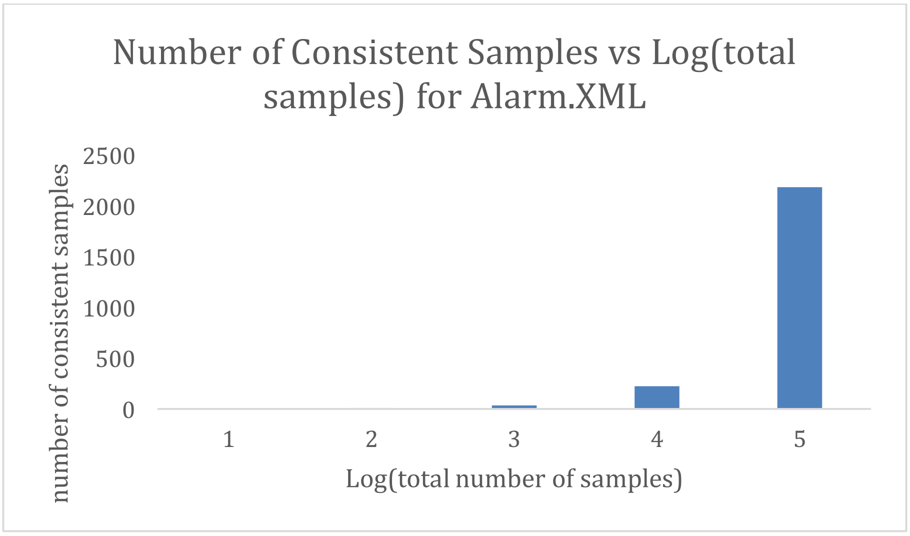
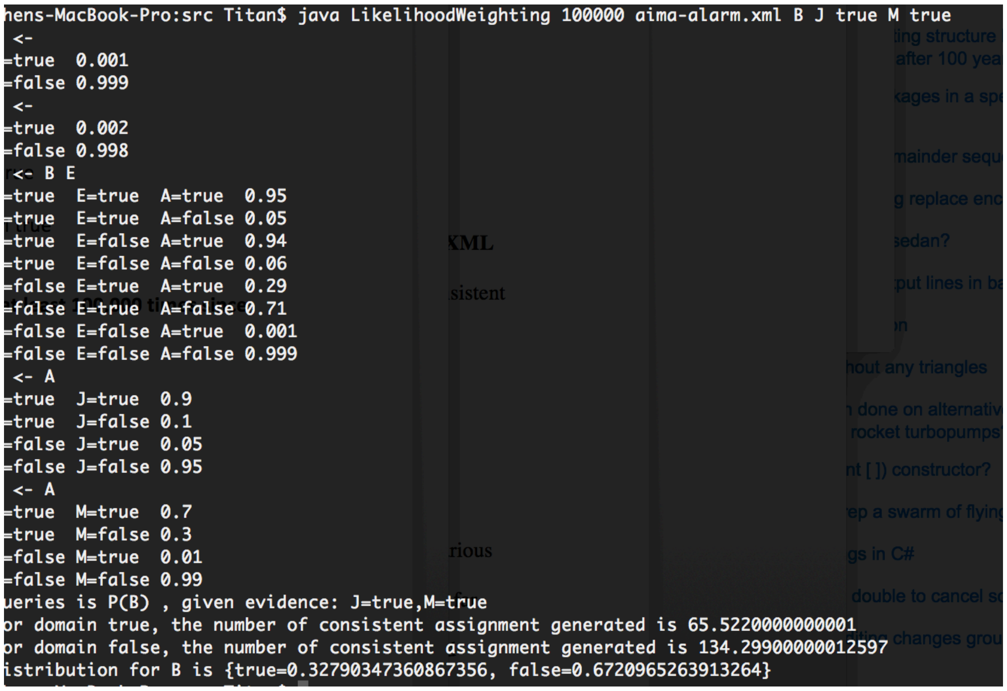


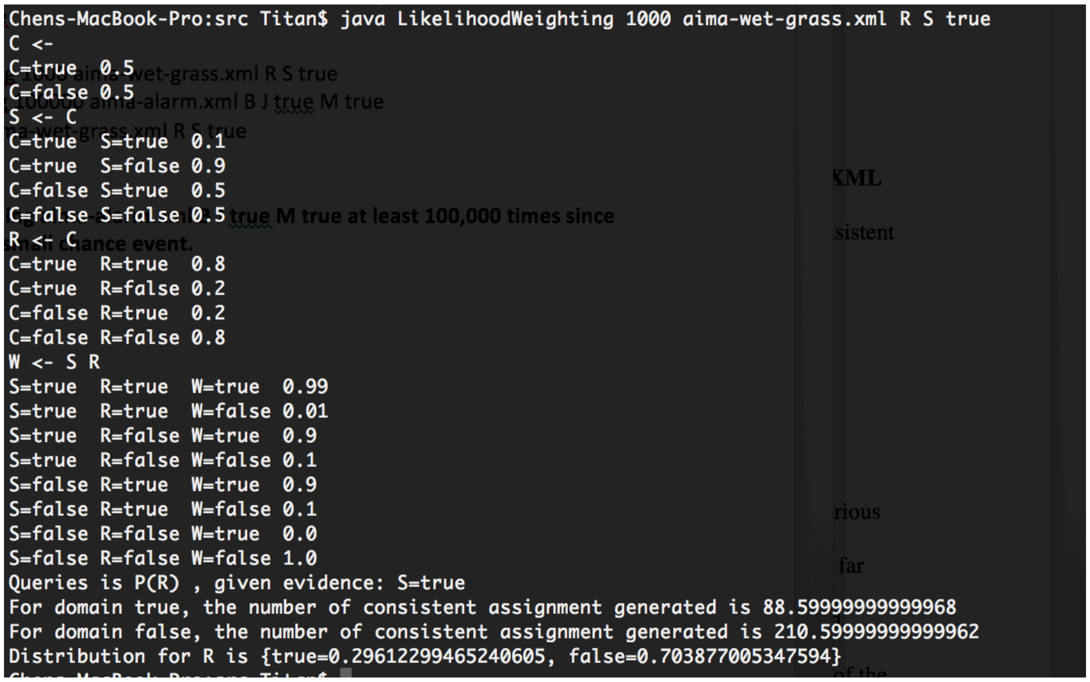
Figure 10. Number of Consistent Samples vs Log(total samples) for Alarm.XML

Also, likelihood tests were generated in Figure 11 and Figure 12. They were consistent with Figure 6 and Figure 7.



```
hens-MacBook-Pro:src Titan$ java LikelihoodWeighting 100000 aima-alarm.xml B J true M true
 <-
=true  0.001
=false 0.999
 <-
=true  0.002
=false 0.998
 <- B E
=true  E=true  A=true  0.95
=true  E=true  A=false 0.05
=true  E=false A=true  0.94
=true  E=false A=false 0.06
=false E=true  A=true  0.29
=false E=true  A=false 0.71
=false E=false A=true  0.001
=false E=false A=false 0.999
 <- A
=true  J=true  0.9
=true  J=false 0.1
=false J=true  0.05
=false J=false 0.95
 <- A
=true  M=true  0.7
=true  M=false 0.3
=false M=true  0.01
=false M=false 0.99
ueries is P(B) , given evidence: J=true,M=true
or domain true, the number of consistent assignment generated is 65.5220000000001
or domain false, the number of consistent assignment generated is 134.29900000012597
istribution for B is {true=0.32790347360867356, false=0.6720965263913264}
```

Figure 11. Likelihood Weighting on AIMA-alarm.xml querying Burglary given John and Mary both call for 100,000



```
Chens-MacBook-Pro:src Titan$ java LikelihoodWeighting 1000 aima-wet-grass.xml R S true
C <-
C=true  0.5
C=false 0.5
S <- C
C=true  S=true  0.1
C=true  S=false 0.9
C=false S=true  0.5
C=false S=false 0.5
R <- C
C=true  R=true  0.8
C=true  R=false 0.2
C=false R=true  0.2
C=false R=false 0.8
W <- S R
S=true  R=true  W=true  0.99
S=true  R=true  W=false 0.01
S=true  R=false W=true  0.9
S=true  R=false W=false 0.1
S=false R=true  W=true  0.9
S=false R=true  W=false 0.1
S=false R=false W=true  0.0
S=false R=false W=false 1.0
Queries is P(R) , given evidence: S=true
For domain true, the number of consistent assignment generated is 88.59999999999968
For domain false, the number of consistent assignment generated is 210.59999999999962
Distribution for R is {true=0.29612299465240605, false=0.703877005347594}
```

Figure 12. Likelihood Weighting on AIMA-wet-grass.xml querying Rainy given sprinkler is true for 1,000 times

Conclusion and Future Improvement

This experiment tested a number of different algorithms for solving inference in Bayesian networks. For exact inference, it is able to calculate the conditional probably in an efficient way since it does not need to random generate samples. It goes through each node and find the CPT at each node. For approximate inference, it can be concluded that both likelihood-weighting and rejection-sampling are inefficient, where the rejection-sampling is the worst. As shown in above, we need to generate at least 10,000 samples to get some acceptable answer for Alarm.XML query. For consistency and accuracy, 100,000 samples are required to generate in this case. One of the great drawbacks of rejection sampling used was that it rejects too many events that are inconsistent with the evidence. However, likelihood weighting avoid this inefficiency. Still, likelihood needs a relatively large input because we still need to randomly generate a certain amount of burglary(true) cases, which happens only 1 out of 1000 times.

I do not have time to implement Gibbs sampling, but in theory it should be the most efficient sampling among three approximate algorithm since it returns consistent estimates for posterior probabilities.