Write-up on Project III

Working Content

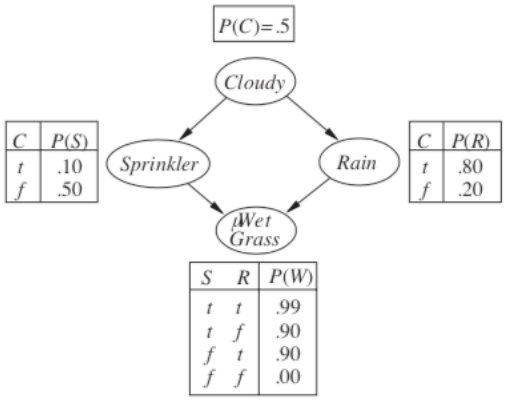
I finish this project independently. The working content includes:

1. XMLBIF parsing and Bayesian network building.
2. Exact inference algorithm.
3. Rejection sampling algorithm.
4. Likelihood weighting sampling algorithm.
5. Gibbs Sampling.
6. Write-up on this project.

XMLBIF parser and Bayesian network setup

The first step of project is to parse the given XML file. More specifically, to get the data and build our Bayesian network. The goal of XMLBIF format is to represent directed acyclic graphs that can be associated to conditional probability measures for discrete variables, with the possibility that decision and utility variables be present in the graph. Actually, the sequence of the appearance of variables is very convenient for us to build Bayesian network because the very first variable is always the one that don't have any parent.

I parse XMLBIF files using DOM dictionary provided by python. Before parsing files, I firstly read the content of file and delete the blank space and annotation, which helps to continue the following work using DOM. Given an example of Bayesian network:



We can easily find that a node should contain these attributes: parents of current node (if exists), children of current node (if exists), and probability distribution table, and hence a Bayesian network is an acyclic graph composed by many nodes. What puzzles me the most during the network building is how to represent the probability distribution table. One method I adopt is combining dictionary and tuple. More specifically, consider the node WetGrass, P(W | S=true, R=true) can be represented {('T', 'T'): 0.99}, notice that I only hold the probability when W is true, if ~W is asked, I can just return 1 – P(W | S=true, R=true).

Inference by enumeration

The basic task for any probability inference system is to compute the posterior probability distribution for a set of query variables, given some observed event. Besides, we should also specify the nonevidence variables. Thus, the complete set of variables are query variables, observed event, and nonevidence variables (hidden variables). A typical query asks for the posterior probability distribution **P**(X | e). More specifically, the query **P**(X | e) can be answered using the follow equation:

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha\, \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y})$$

In which X denotes query variables, e denotes observed event, and y denotes hidden variables.

The Bayesian network gives a complete representation of the full joint distribution. More specifically, the terms P(x, **e, y**) in the joint distribution can be written as products of conditional probabilities from the network by using the equation:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid parents(X_i))$$

Therefore, a query can be answered using a Bayesian network by computing sums of products of conditional probabilities from the network.

In the actual operation, we can evaluate the query expression by looping through the variables in order, multiplying conditional probability table entries as we go. For each summation, we also need to loop over the variable's possible values. The enumeration-ask algorithm evaluates such expressions using depth-first recursion. This algorithm is very similar to the truth-table enumeration algorithm in project II.

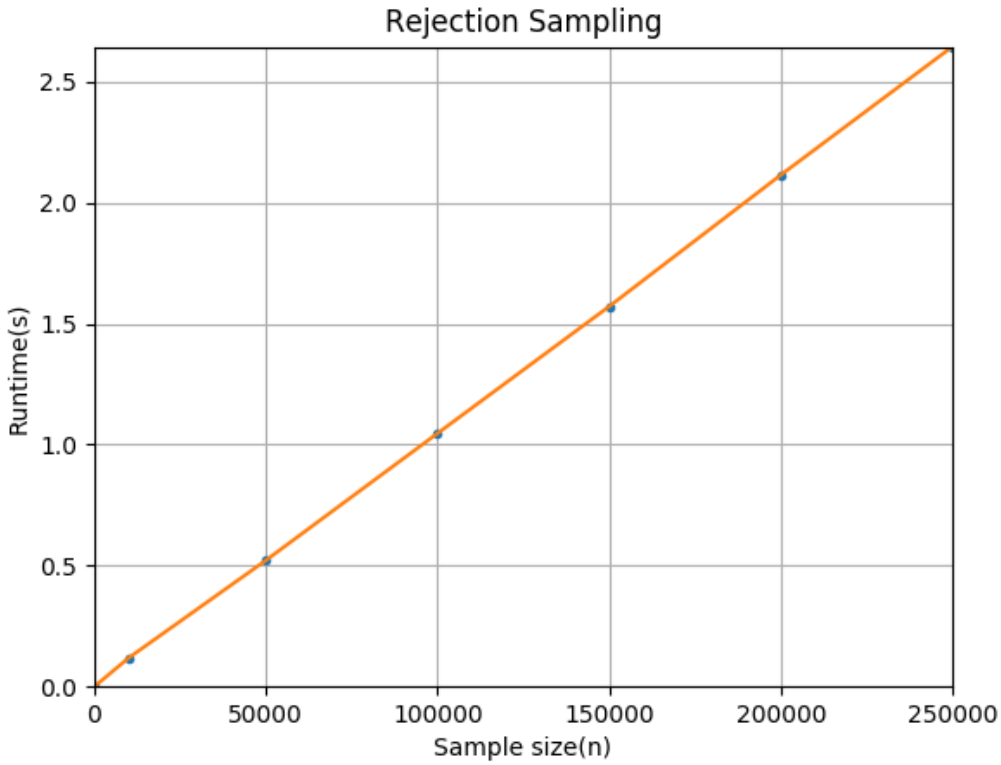
The space complexity of enumeration-ask algorithm is linear in the number of variables. Its time complexity is always $O(2^n)$ for a network with n Boolean variables.

Rejection sampling

Rejection sampling is one type of approximate inference algorithm, which can be used to compute conditional probabilities. First, it generates samples from the prior distribution specified by the network. Then, it rejects all those that do not match the evidence. Finally, the estimate P(X=x | **e**) is obtained by counting how often X=x occurs in the remaining samples.

The performance of rejection sampling algorithm is represented in the following graph. The runtime of algorithm is linear in the size of sample.

The biggest problem with rejection sampling is that it rejects so many samples. The fraction of samples consistent with the evidence e drops exponentially as the number of evidence variables grows, so the procedure is simply unusable for complex problems.



The accuracy of rejection sampling comparing to exact inference as the number of samples increases is also show as follows (e.g. P(B | j, m)) :

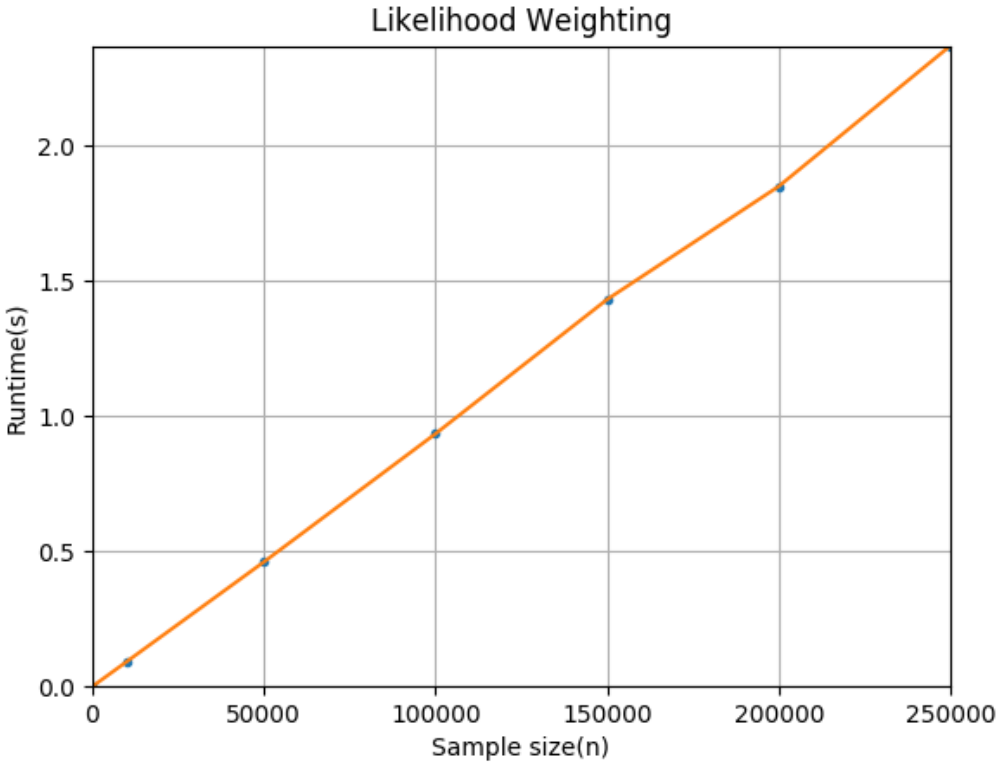
```
Exact Inference:  {'T': 0.2841718353643929, 'F': 0.7158281646356071}  Runtime: 0.00020813941955566406 s
Rejection Sampling (using 10000 samples): {'T': 0.35, 'F': 0.65}  Runtime: 0.1428208351135254 s
Rejection Sampling (using 50000 samples): {'T': 0.28695652173913044, 'F': 0.7130434782608696}  Runtime: 0.6299989223480225 s
Rejection Sampling (using 100000 samples): {'T': 0.31336405529953915, 'F': 0.6866359447004609}  Runtime: 1.3272440433502197 s
Rejection Sampling (using 150000 samples): {'T': 0.26666666666666666, 'F': 0.7333333333333333}  Runtime: 1.9395530223846436 s
Rejection Sampling (using 200000 samples): {'T': 0.24669603524229075, 'F': 0.7533039647577092}  Runtime: 2.6574878692626953 s
```

We can find that the estimate probability distribution is close to the result of exact inference algorithm. However, it sometime turns out to be less accurate from the true value.

Likelihood weighting

Likelihood weighting avoids the inefficiency of rejection sampling by generating only events that are consistent with the evidence e. in weighted-sample, each nonevidence variable is sampled according to the conditional distribution given the values already sampled for the variable's parents, while a weight is accumulated based on the likelihood for each evidence variable.

Likelihood weighting algorithm uses all the samples generated, so it can be more efficient that rejection sampling. The performance of the algorithm is shown as follow:



The accuracy of likelihood weighting comparing to exact inference as the number of samples increases is also show as follows (e.g. P(B | j, m)) :

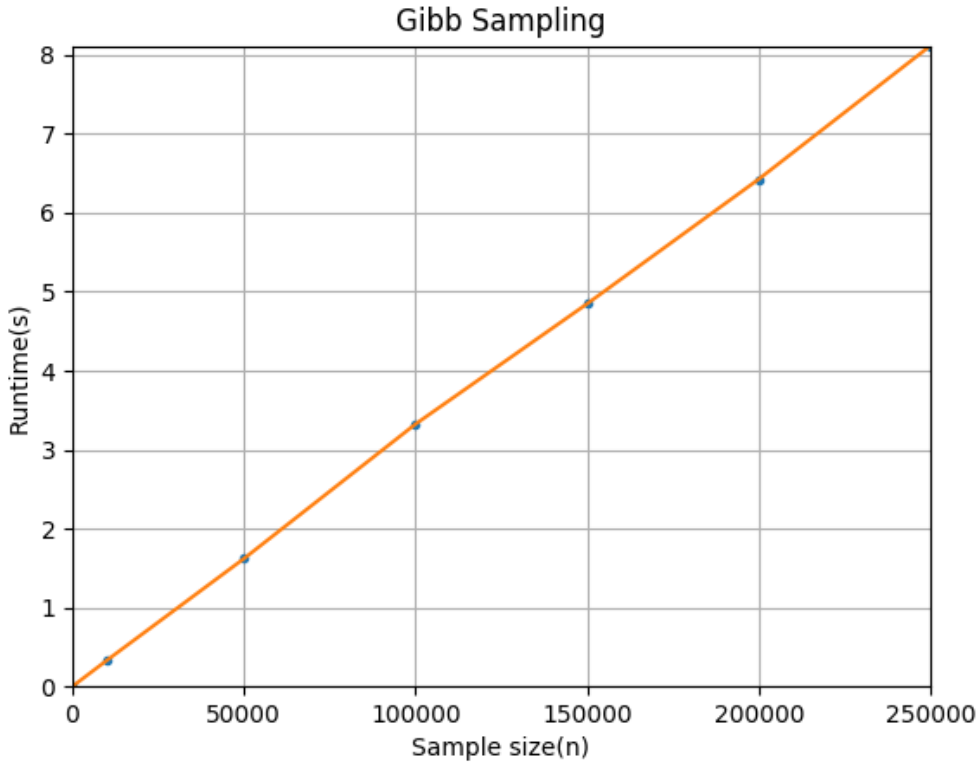
```
Exact Inference:  {'T': 0.2841718353643929, 'F': 0.7158281646356071}  Runtime: 0.00020813941955566406 s
Likelihood Weighting (using 10000 samples): {'T': 0.3037974683543878, 'F': 0.6962025316456122}  Runtime: 0.1075291633605957 s
Likelihood Weighting (using 50000 samples): {'T': 0.29026009381790013, 'F': 0.7097399061820998}  Runtime: 0.5463259220123291 s
Likelihood Weighting (using 100000 samples): {'T': 0.2723514235654271, 'F': 0.7276485764345729}  Runtime: 1.1676433086395264 s
Likelihood Weighting (using 150000 samples): {'T': 0.2511631223377854, 'F': 0.7488368776622146}  Runtime: 1.6454358100891113 s
Likelihood Weighting (using 200000 samples): {'T': 0.28071217666586473, 'F': 0.7192878233341352}  Runtime: 2.0688068866729736 s
```

The accuracy of likelihood weighting is similar to rejection sampling, they all sometimes departure from true result. However, we can find that the runtime of likelihood weighting is always less than rejection sampling in terms of same amount of samples.

Gibbs sampling

The Gibbs sampling algorithm starts with an arbitrary state (with the evidence variables fixed at their observed values) and generates a next state by randomly sampling a value for one of the nonevidence variables X_i. The sampling for X_i is done conditioned on the current values of the variables in the Markov blanket of X_i. The algorithm therefore wanders randomly around the state space, flipping one variable at a time, but keeping the evidence variables fixed.

The performance with respect to the size of sample and runtime of algorithm is shown as follows:



The accuracy of Gibbs sampling comparing to exact inference as the number of samples increases is also show as follows (e.g. P(B | j, m)) :

```
Exact Inference:  {'T': 0.2841718353643929, 'F': 0.7158281646356071}  Runtime: 0.00020813941955566406 s
Gibbs Sampling (using 10000 samples): {'T': 0.2834, 'F': 0.7166}  Runtime: 0.3789398670196533 s
Gibbs Sampling (using 50000 samples): {'T': 0.28208, 'F': 0.71792}  Runtime: 2.096114158630371 s
Gibbs Sampling (using 100000 samples): {'T': 0.28597, 'F': 0.71403}  Runtime: 4.387639760971069 s
Gibbs Sampling (using 150000 samples): {'T': 0.2841666666666667, 'F': 0.7158333333333333}  Runtime: 7.071847915649414 s
Gibbs Sampling (using 200000 samples): {'T': 0.28513, 'F': 0.71487}  Runtime: 7.800537109375 s
```

The accuracy of Gibbs sampling is constant and close to the real probability, whereas the runtime is always higher than other two approximate algorithms.

The performance comparison of three approximate algorithms is shown as follows: