Nicholas Allaire, A10639753
Son Tang, A11370127
Marcel Aguiar, A10904034

CSE 150 Programming Assignment 4 Write-up

Problem 1:

P(W+ | R+, C+) = P(W+, R+, C+) / P(R+, C+)
= Sum over s of P(W+, R+, C+, S) / .8
= P(S+ | C+) * P(R+ | C+) * P(W+ | S+, R+) + P(S- | C+) * P(R+ | C+) * P(W+ | S-, R+)
= ((0.1 * 0.8 * 0.99) + (0.9 * 0.8 * 0.9)) / (0.8)
= .7272 / 0.8
= **.909**

Problem 2:

P(S+ | R+) = Sum over c P(S+, R+) / Sum over c P(R+)
= Sum over c P(S+, R+, C) / Sum over c P(R+, C)
= P(C+) * P(S+ | C+) * P(R+ | C+) + P(C-) * P(S+ | C-) * P(R+ | C-) /
(P(C+) * P(R+ | C+) + P(C-) * P(R+ | C-))
= ((0.5 * 0.1 * 0.8) + (0.5 * 0.5 * 0.2)) / ((0.5 * 0.8) + (0.5 * 0.2))
= .09 / 0.5
= **.18**

Problem 3:

Rejection Sampling

Algorithm: Iterate through all the specified number of samples, and reject those that aren't consistent with the evidence. Return the normalized number of samples to be generated.

Implementation: To implement rejection sampling, we loop through the number of samples. We use *self.priorSample()* to check against the prior sample for consistency. If consistent, we increment *normalizedCount*. If consistent AND the prior query variable is true, we increment count. Lastly, we return the count divided by the *normalizedCount*.

Weighted Sampling:

Algorithm: For likelihood weighted sampling, we use a similar approach to rejection sampling except that instead of checking all of the samples, we fix the values for the given variables in the assignment event. By doing this, we are eliminating all of the bad samples and only focusing on events with the correct values. For each event a weight is calculated if the node is an evidence variable and then the event and weight are returned. If the query variable is true in the returned event, the weight is then summed up and is normalized over all of the returned weights, giving us an estimated probability.

Implementation: We implemented this algorithm by looping the number of times the parameter numSamples specifies. In each iteration, a helper function is called which returns the event and weight associated with the event. The helper function works by first creating an empty event and assigning the evidence variables to the event. We have to search in topological order so we start by looping through all of the root nodes, calculating their weight, and adding them to a queue. Then we loop until the queue is empty, and during each iteration we get the next node, add its children to the queue. If the current node is an evidence variable, we calculate the weight and multiply it by the current weight. If the node is not an evidence variable, a random sample is chosen and a value is chosen from that sample.

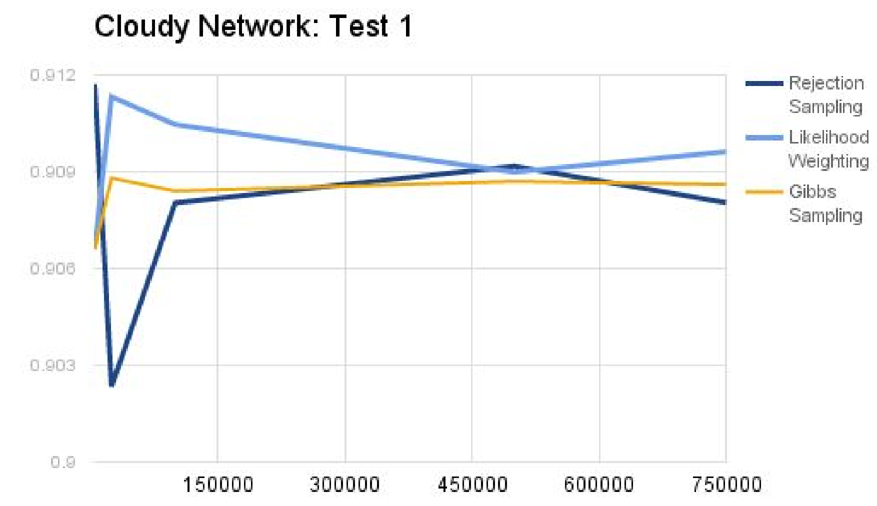
Gibbs Sampling

Algorithm: For Gibbs Sampling, you start off by creating an event with the given variables and assigning the nonevidence variables to random values. You then randomly select a nonevidence variable and determine the probability of it given its markov blanket. Once you loop through all of the nonevidence variables, you check the current state to see if the query variable is true, if so increment the count. Finally once you go through all of the trials, you normalize the count of events with the query variable being true.

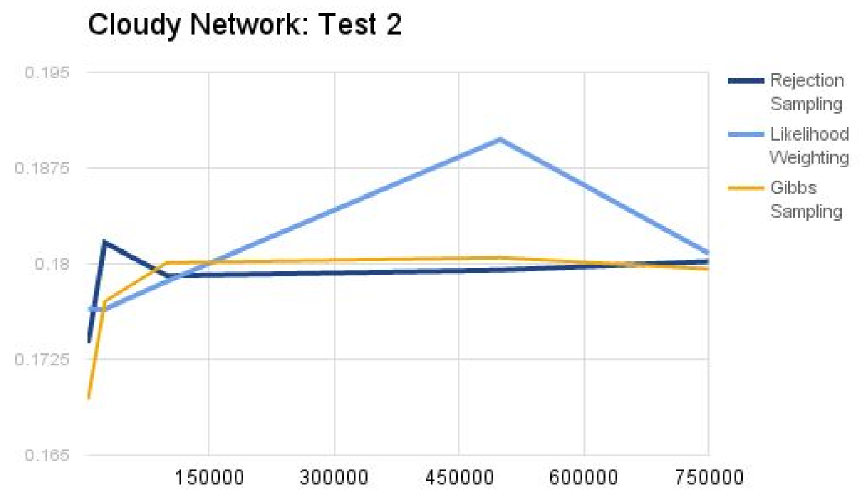
Implementation: We implemented Gibbs Sampling by creating a list of nonevidence variables and assigning the current state with the given variable values. We then randomly assign the nonevidence variables in the current state. You then loop the amount of times according to the number of trials parameter. In each iteration, we randomly chose a nonevidence variable, calculate the probability of the nonevidence variable given the markov blanket of the nonevidence variable. To do this you first calculate the probability of the nonevidence variable to be true, and then calculate it when it's false, and normalize the true probability by dividing by the sum of the true and false probability. We then assign the nonevidence variable a value based on the probability and update the current state with the new value. After you do this for all nonevidence variables, you keep track of when the number of times the query variable is true and normalize the value by dividing by the number of trials based in.

Analysis of Graphs for CloudyNetwork.py

Test 1: For both rejection sampling and likelihood weighted, it appears that at around 500,000 samples we get really close to our estimate of 0.909 and then after 500,000 samples the data tends to move away from the estimate we calculated. As for Gibbs sampling, it appears to reach within .01% of the estimate of 0.909 after only 25,000 samples.



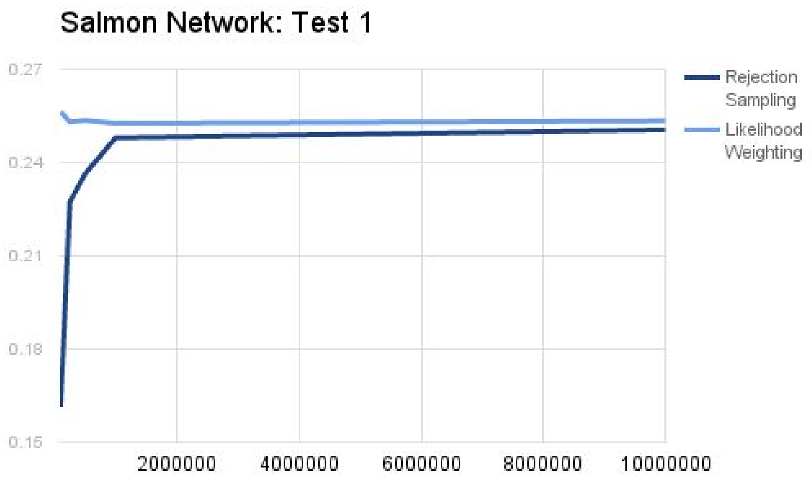
Test 2: Likelihood weighting seems to have a very interesting graph. It climbs really fast and goes way above the predicted probability, and then slowly creeps down to the predicted value. Rejection Sampling on the other hand is slow and gradually gets closer and closer to the predicted value as the number of samples increase. Both algorithms for this data set seem to converge to .18 at roughly 750,000 samples. Gibbs on the other hand seems to be extremely quick for this test data to arrive at an answer. After a mere 100,000 samples, it arrives at the predicted value. So for this data set, Gibbs Sampling is the definite choice to optimize your answer quickly.



Problem 4:

Analysis of Graphs for SalmonNetwork.py

<u>Test 1</u>: Likelihood weighting was consistently .25xxxx for the 5 tests ranging from 100,000 - 10,000,000 samples, meaning that we get an accurate probability by using at most 100,000 samples. Rejection sampling on the other hand was slow to converge to .25, taking upwards of 10,000,000 samples to get it at/or above .25xxx, meaning rejection sampling is way slower and inefficient for this data set.



Problem 5:

Contributions

Nicholas Allaire: Helped work on solving the probabilities for the first and second problem. Helped the group code all of the sampling problems, as well as debug them. Worked with the TA to properly implement the markov blanket calculation. I learned that as the sample size gets large, rejection sampling can take quite a long time, so optimization is essential to get quick results by using alternative approaches, i.e. weighted sampling. I also helped make the graphs for the write up and explain the weighted sampling algorithm along with our implementation.

Son Tang: Confirmed the probabilities of the first two problems by solving by hand. Worked to write and debug the three sampling methods. Figured out what was causing our likelihood weighting to return an incorrect probability in the Salmon Network. I learned how useful sampling can be when dealing with probabilities. I also learned how to optimize sampling rate by using alternate algorithms and how it is useful for reducing the time it takes to produce an accurate probability from large samples.

Marcel Aguiar: Calculated the conditional probabilities for problems 1 and 2. After we all had matching answers, I moved on to problem 3. I attempted converting the textbook's psuedo-code into python for rejection sampling, likelihood weighting, and Gibbs sampling. We ran into issues with the value our Gibbs sampling was returning for Test 2, so I brainstormed a solution to bring down our answer of .30 to the expected .18. Lastly, I contributed to the write-up.