Mark Gray
mmg3339
markmgray2010@gmail.com

Sampling: Approximate Inference in Bayes Network

Introduction:

Implementing this project was difficult to do in MATLAB. My first thought of the implementation was going to make use of Object Oriented languages. However, MATLAB doesn't have a pass by reference function so after implementing many aspects of the assignment I scrapped my MATLAB implementation and switched to Java. Both Gibbs and Rejection sampling are fast algorithms with respect to the size of the network and the amount of evidence you are given. I built a query interface that allows me to ask for the probability of an arbitrary number of variables given some evidence that could contain an arbitrary number of variables as well. An example of this would be: $P(+b \mid +c)$ but there could be many variables on either the left or right side of the query.

I worked with Vismay, Eddy, and Pulkit to build a large Bayes Network that we could test with. We did this simply to ensure that each of us was getting approximately the correct output from our own implementations. This network has no realistic meaning, but contains 10 nodes interconnected as a directed acyclic graph as required for a Bayes Network. This was a good test for my implementation to ensure that it was designed well enough to handle the larger sized networks and still perform correctly.

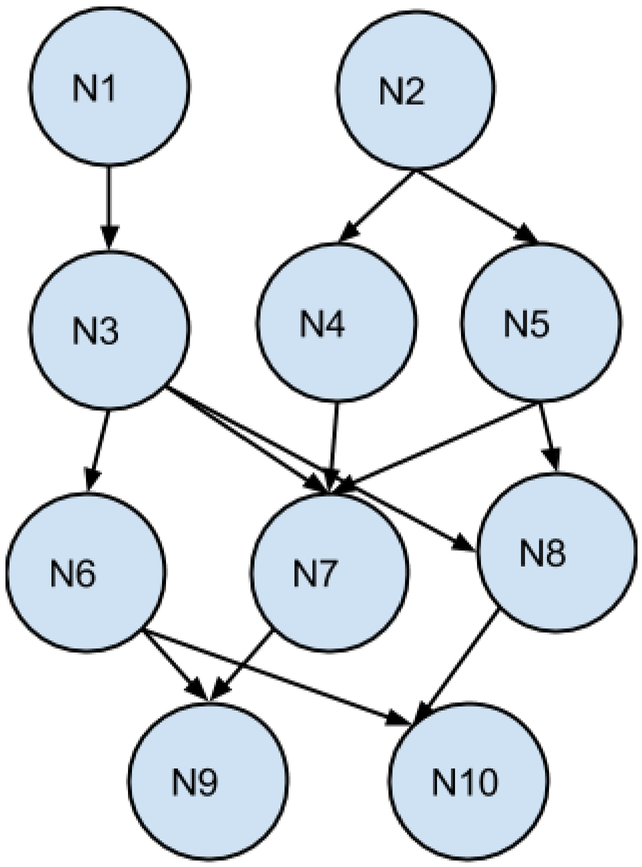
Implementation:

My implementation involves a Bayes Network class and a Node class and a simulation file. The node class contains all information for each point in the graph including, probability table, parents, childrens, and the id of that node. It also contains several functions that allow me to look up its probability based on the status of its parents and compute the Gibbs conditional probability with respect to everything in the network. This is done by holding a reference to every object that is a parent or a child of the current node object.

Next is the Bayes Network class that contains two main functions: queryNetworkRejection() and queryNetworkGibs(). These two functions are the implementations of Rejection sampling and Gibbs sampling respectively. These were written according to the algorithms in both the Bishop book and found in a youtube video posted on piazza. My Gibbs sampling computes the probability using the Markov blanket setup described in Bishop and this gives a marginal speedup in the algorithm.

Finally is my simulation file is where I implement both classes together to create a functional Bayes network. I time several executions of each algorithm with various bun-in times and sample sizes for Gibbs and various sample sizes for rejection sampling. My results of all of my tests is described in detail below. In general I initially found that Rejection sampling was faster than Gibbs sampling, but with a slightly higher error rate.

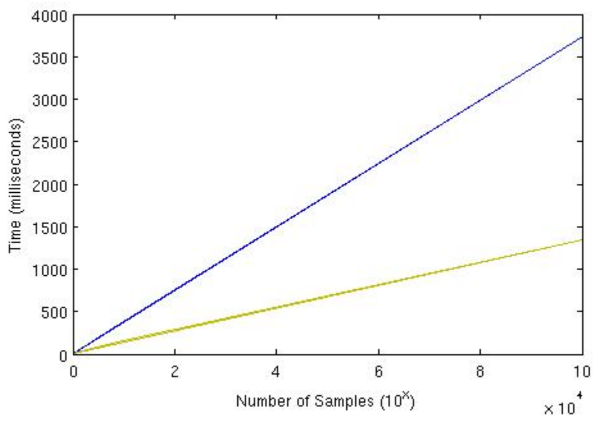
Mark Gray
mmg3339
markmgray2010@gmail.com



Graphical representation of the tested Bayes Network

Experiments

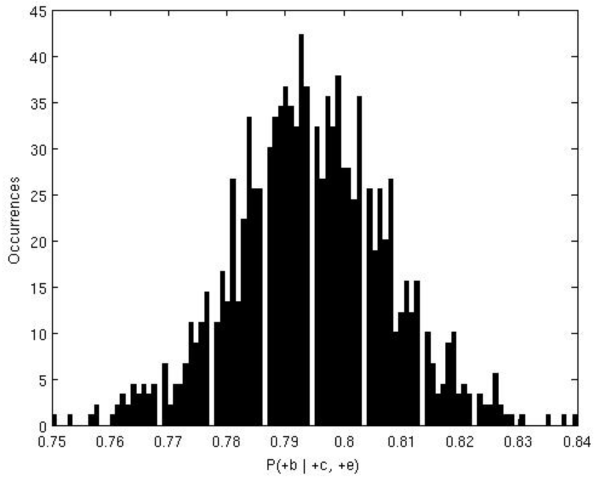
The above graph represents the Bayes Network that my tests were run on. The main thing I wanted in this network was that one node, n7, had at least 3 parents. The conditional probability tables are encoded into my program and remain the same on every run. My first experiment was checking out the influence of the number of samples on the running time of each algorithm with a fixed number of burn-in iterations for Gibbs and fixed queries. Below is the graph that shows my findings. I took the average of multiple runs for each amount of samples. It came out to a linear graph which was expected. Blue is Gibbs sampling and yellow is Rejections.

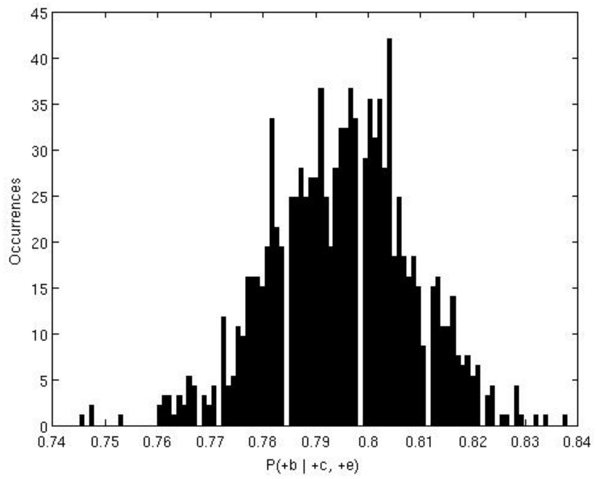


Mark Gray
mmg3339
markmgray2010@gmail.com

I believe the above results were rather obvious so I didn't spend a lot of time testing the speed of the algorithms with just the number of samples as a parameter. Another question was what happens to the time of execution when there are a different number of evidence variables. My conjecture is that the more evidence variables you have in your query the longer Rejection sampling might take. The idea behind this thought is the algorithm itself has a fundamental issue. It builds a random sample and every part of the sample must be equal to the evidence. Because of this property the algorithm depends on the stochastic probability that everything in the built sample will line up with the evidence. To test this using the Bayes network above I set the first nine nodes as evidence and the tenth node as the query variable and ran both Gibbs and Rejection sampling using 1000 samples. For one test Gibbs ran in 68 milliseconds and Rejection ran in 4534 milliseconds. My tests showed that on average Rejection ran around 60 times slower than Gibbs when given nine pieces of evidence.

Rather than focus on the time the algorithm took to run I wanted to determine what my program was actually producing. I decided to run my Gibbs and Rejection sampling a thousand times on the same query and record all of the outputs and plot them. I expected something that resembled a uniform distribution with the mean being the actual value of the query. Below are the first two graphs I ran with their queries on the x-axis and the occurrences on the y. On the left is Rejection Sampling, and Gibbs on the right.



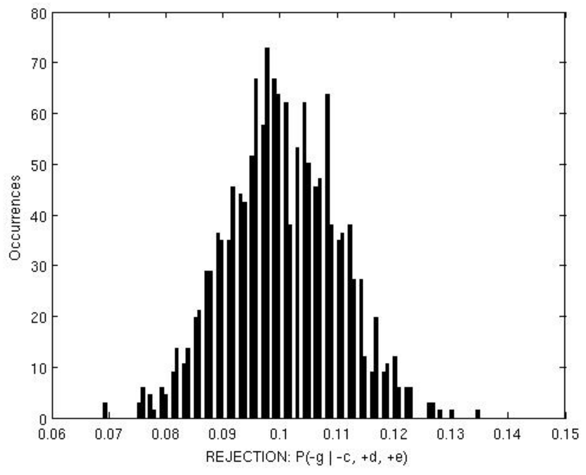


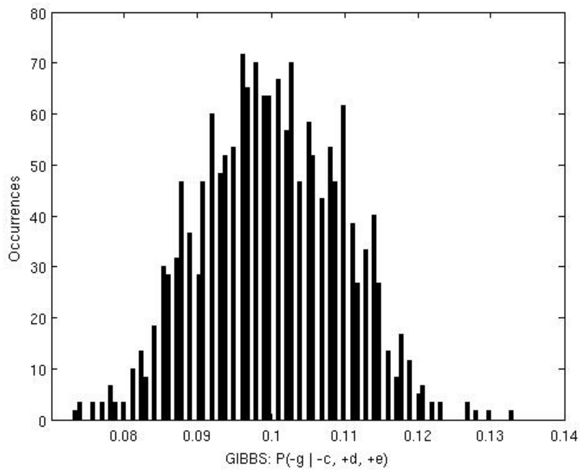
These graphs confirmed my suspicion that taking many samples and plotting them would give you a Gaussian distribution. To make sure I was correct I ran another simulation on a query with a known result. My hopes were that the mean of the distribution would be approximately the correct value for that query. For the two above graphs the means were 0.7944 and 0.7949 respectively.

I decided to take this evidence and confirm its validity by hand computing a query. Below are the graphs that resulted from running that query. (Rejection left, Gibbs right) After taking all the values for both graphs I calculated the mean of each and it came up 0.1003 and 0.1001 respectively. My hand computed solution to the query $P(-g \mid -c, +d, +e)$ was 0.1

Mark Gray
mmg3339
markmgray2010@gmail.com

and this confirmed my hypothesis. Despite both means being slightly off they were very close to the correct value. Both of these sampling methods gave very accurate results in all of my tests.





Conclusion

After looking at all my results and analyzing both algorithms I have concluded that Gibbs is the better of the two algorithms. Accuracy can be reached for both of them by running a very large number of samples and using a burn-in number between 20-100 for Gibbs. This burn-in increases time of execution but doesn't necessarily mean increased accuracy past a certain point so I had to tune it carefully. The reason I believe that Gibbs is the better algorithm is that it has a standard run time in the general case as opposed to the conditional run time of the Rejection sampling algorithm. Rejection runs extremely slow compared to Gibbs when there is a lot of evidence known. The reason for this is explained at the beginning of my experiments section of my report. I have found that Rejection and Gibbs both give really good accuracy with around 1,000 samples taken.