Danielle DeLooze

CSC 242 Project 3: Uncertain Inference

April 5, 2018

George Ferguson

Representation

The majority of the data structures needed for this project were provided to us by Professor George Ferguson. There were six main classes needed to represent Bayesian networks and implement uncertain inference algorithms: Assignment, BayesianNetwork, CPT, Distribution, Domain, and RandomVariable.

Assignment: A linked hashmap that that maps variables to a value
BayesianNetwork: A directed acyclic graph represented by a set of Nodes. Each node contains a list of its parents, a set of its children, the random variable it represents, and its cpt.
CPT: A conditional probability table that corresponds to and random variable and contains a set of entries for each value of that variable
Distribution: A linked hashmap of values to their probabilities
Domain: An arraylist of the possible values for a random variable
RandomVariable: A string identifying the random variable and a domain

We were also provided all of the example Bayesian networks in XML/bif format.
I've created 4 different classes to complete this project. They are all under the inference package. BNInferencer is the main method that handles all of the input arguments. EnumerationAsk implements the inference by enumeration algorithm (exact inference). RejectionSampling implements the rejection sampling algorithm for approximate inference. LikelihoodWeighting implements the likelihood weighting algorithm for approximate inference.

Inference by Enumeration (Exact Inference)

My implementation of this algorithm was almost exactly the sample as the pseudocode given in our textbook. It takes in a Bayesian Network, a query random variable, and an assignment of evidence variables. It creates an initially empty distribution. It then iterates over all of the values in the domain of the given query variable. For each value, it extends the original assignment and calls a method Enumerate_all() and puts the value of this call into the distribution. Enumerate_all() returns the probability of the given assignment. It does this by first sorting all of the variables into a list in topological order, so that parent variables/nodes always come before the children. Then for each variable in this list it checks if the variable is contained in the assignment already (known value). If it is contained it returns the probability of this variable given its parents (obtained from Bayesian Network) and multiples it by a recursive call to Enumerate_all() on the rest of the variables. If it is not contained the variable is a hidden

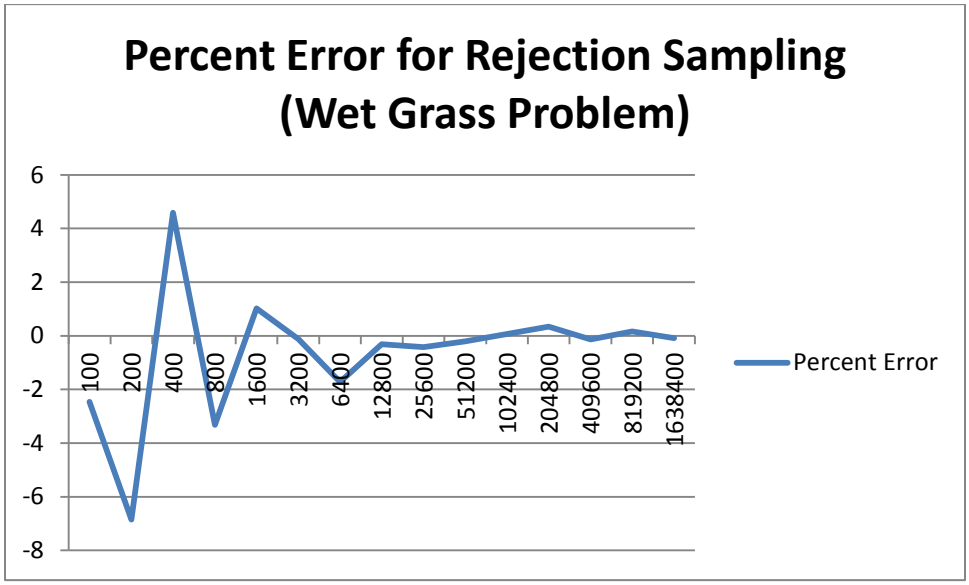
variable. It returns the summation over all values of the variables of the probability of the variable given its parents multiplied by a recursive call to Enumerate_all() on the rest of the variables in the list.

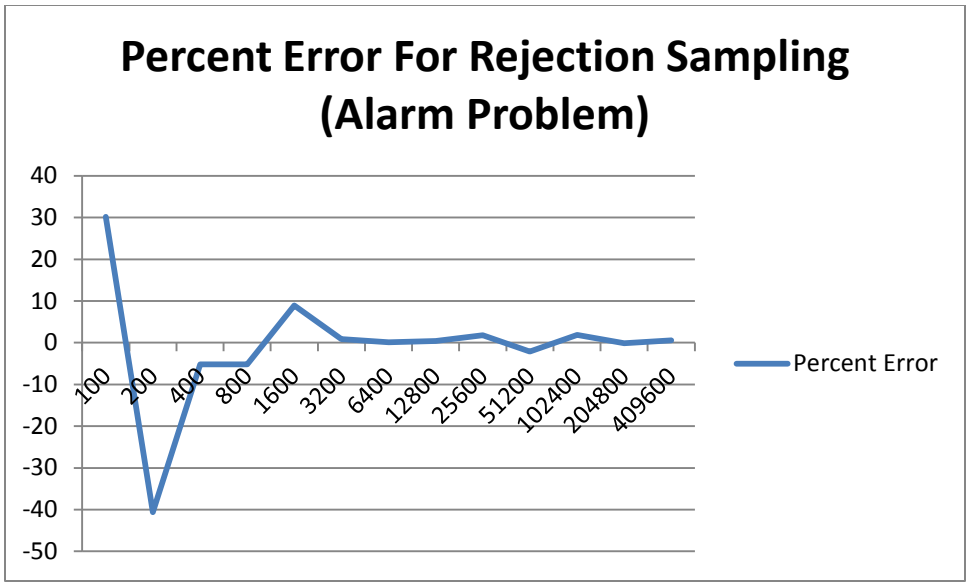
Rejection Sampling

Rejection sampling was fairly simple to implement. The hardest part of it was writing a function to generate a random assignment of variables. Rejection Sampling takes in a query random variable, an assignment for evidence variables, a Bayesian network, and a number of trials to run. It creates an empty Distribution to start. Then for the number of trials to be run, it generates a random assignment of the variables in the Bayesian Network and checks if it is consistent with the given assignment. It is consistent if all of the evidence variables in the random assignment match the values in the given assignment. If it is consistent, it increments the double for the value of the query value in the distribution by 1. At the end this distribution is normalized and returned.

My random assignment is generated by creating a distribution for each variable in the network that is populated with the probability that each value will occur. Then it iterates through the distribution keeping track of a sum of all the probabilities iterated through. If a random double 0.0-1.0 is less than the current value probability, it will set the current variable to that value.

Here are two graphs I have made to demonstrate how the approximation becomes more accurate as the trial number gets larger. The alarm problem was run with B as the query variable and J and M true. The wet grass problem was run with C as the query variable and R and W true. Percent Error was calculated by comparing it to the value provided by inference by enumeration. Percent Error in this case was the average of the true false percent errors ((exact value-measured value)/exact value)*100





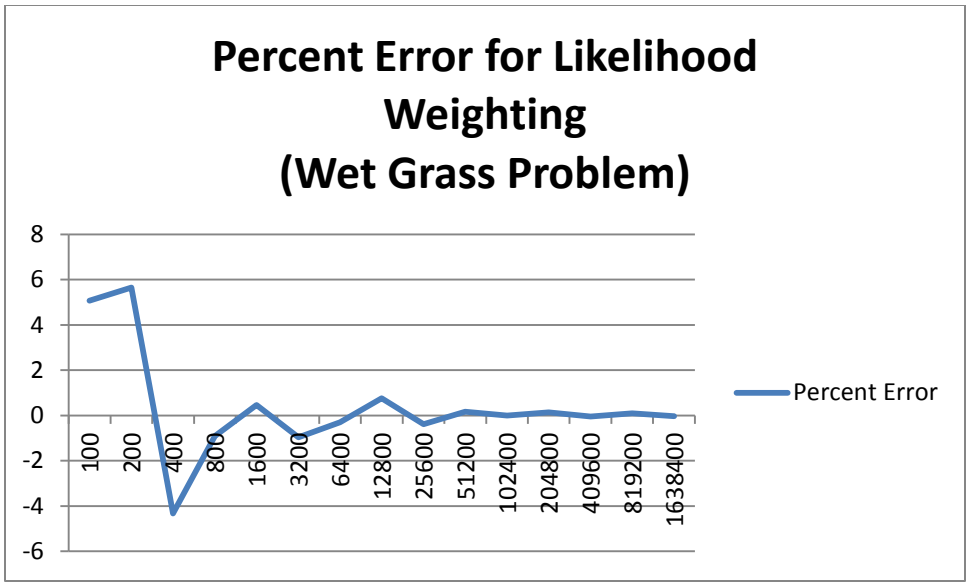
Likelihood Weighting

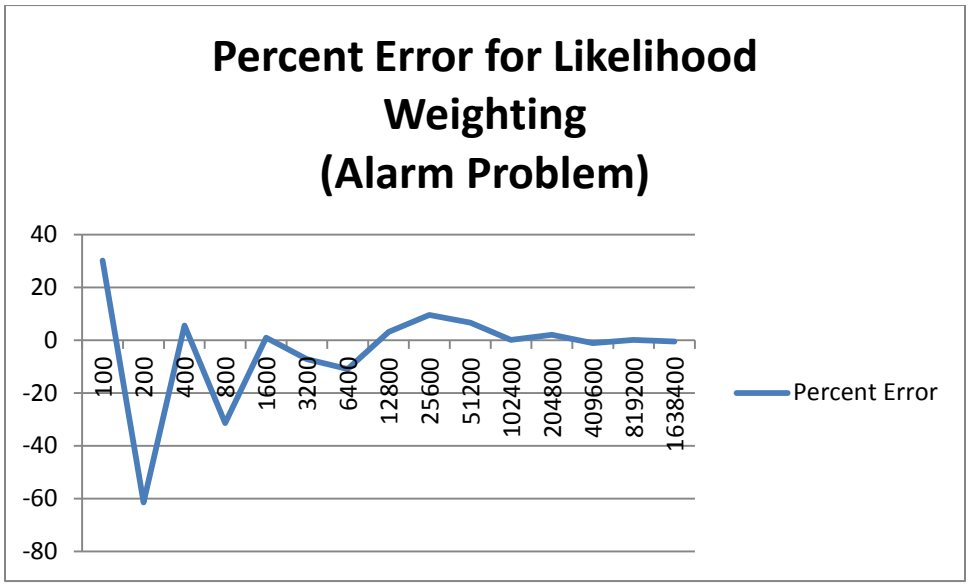
For likelihood weighting I first needed to implement a new class WeightedSample. WeightedSample just contains an assignment e, and the weight of this assignment (double).

LikelihoodWeighting takes in a query random variable, an assignment of evidence variables, a Bayesian network, and the number of trials to run. It creates an empty Distribution . Then LikelihoodWeighting generates a WeightedSample for each trial. It then adds the weight of this WeightedSample to the double in the Distribution for the value of the query variable in the WeightedSample. The distribution is normalized and returned.

A Weighted Sample is generated by first copying the original assignment into a new assignment and sorting the variables in topological order. Then for all of the random variables: If it is contained in the assignment, the weight is multiplied by the probability of this variable given its parents. If it is not contained, the variable is set to a random value. The random value is determined by creating a distribution for the given variable for each of its value. Then iterating through the distribution while keeping track of the sum of probabilities and comparing it to a random double value 0.0-1.0.

Here are some graphs I created for the two problems and assignments from above, but with likelihood weighting used for the approximation.





Results

For both of the approximate inference methods used, we see a data trend that matches what we expect. As the trial number is increased the percent error decreases to an almost match the exact probability.

We can even see one of the downsides with both of these approximation methods from the graphs. As the number of variables increase, these methods become much less reliable. For rejection sampling, the number of consistent assignments generated will decrease. Giving you less samples to base the probability on and making it less accurate. For likelihood weighting, the weights given to the assignments will decrease and become dominated by samples that have higher weights. These effects can be seen by the higher range in percent error for the alarm problem (1 more variable). The alarm problem ranges from +%30 to -40/-60% while the wet grass problem ranges from +5% to -5/-7%.

Conclusion

Overall, I'm pretty satisfied with my implementations of the algorithms. Though given more time I would have liked to implement some speedups for the inference enumeration method. The inference by enumeration method was taking a long time to run for the larger Bayesian networks. The variable elimination algorithm would hopefully speed this process up more.

I might also return to the methods I created to create random assignments/values. I don't think that this was the most efficient way to do this and it is what takes the majority of time in both the rejection sampling and likelihood weighting.