Advanced AI

Project 1

Jordan White

1.1 Overview

In this project, we were tasked with implementing three approximate inference algorithms for Bayesian networks: Likelihood weighting, Gibbs sampling, and Metropolis-Hastings. The purpose of these algorithms is to estimate the value probabilities of a queried variable within a Bayes net, taking into consideration any given evidence variables.

The first of these algorithms, Likelihood weighting, accepts a query variable name, a Bayesian network, a set of evidence variables, and a number of desired samples N, and outputs the estimated probabilities for each possible value of the query variable (in this case, the probabilities for the variable being 0 or 1). The way it works is by developing a topological order for the variables in the Bayes net (any valid order works), and then creating N event samples (an event has an assignment for each variable). It creates each event by assigning the given values for each evidence variable, then traversing the topological order from the earliest nodes to the latest, assigning a value for each by sampling from the distribution for each variable that is based on the values of its parent nodes. Each event also has a weight based on how likely that event is to not be rejected by rejection sampling. Once N events have been created, the query variable values are counted from the weighted sum of the samples and the result is normalized to return the probability table for the query variable.

The second algorithm, Gibbs sampling, works by starting with one event (in which the evidence variables are fixed) and changing the variables one by one based on their respective probabilities in the current state. The value of the query variable is counted from each of these samples, and after N samples have been created the value counts of the query variable is normalized to return its probability distribution.

The third algorithm, Metropolis-Hastings, could be implemented in multiple ways. I chose to implement it by starting with a random event (with values fixed from the evidence variables) and then performing Gibbs sampling at each step with probability *p,* or else jumping to a new event state using the weighted sampling algorithm with probability. This prevents the sampling algorithm from getting stuck in sampling loops, which is possible with Gibbs sampling.

1.2 Results

Figure 1 displays the results likelihood weighting, gibbs sampling, metropolis-hastings, and enumeration-ask on a polytree of node size 10 and a Directed Acyclic Graph (DAG) of size 10. Each algorithm, in each of the figures, was tested 10 times, each time using a sample count of 1000 events.

As you can see, likelihood weighting and metropolis-hastings perform well, close to the value found by the exact inference (enumeration-ask) algorithm: All three results are within 2% of each other (~85% for the polytree, ~64% for the DAG), and the standard deviation of the likelihood weighting and metropolis-hastings algorithms are below 1%.

Gibbs sampling, on the other hand, diverges slightly on the DAG (~57%), and in both cases Gibbs sampling maintains by far the highest standard deviation (~6% and ~13%, respectively). This indicates that Gibbs sampling is not very reliable for sample counts as high as 1000.

In Figure 2, the performance of each algorithm is demonstrated on polytrees of various sizes (5 and 15 nodes). Not surprisingly, the execution time increases with the node count. The increase in execution time that results from increasing the node count seems to be more extreme for likelihood weighting than it is for gibbs sampling and metropolis-hastings. This makes sense, because the in gibbs sampling each sample can be created knowing only the Markov blanket of a given node, whereas the entire Bayesian net must be consulted to create each new sample in likelihood weighting. This would imply that the time complexity of likelihood weighting increases more exponentially than gibbs sampling with respect to node count. This is something I might want to explore further to confirm my intuitions.

In Figure 3, the performance of metropolis-hastings is given for various values of p (0.95, 0.85, and 0.75). Surprisingly, the execution time increases as p increases, despite the fact that a higher p value seems to indicate more gibbs sampling, which is generally faster than likelihood weighting. Also, Metropolis-hastings is much more accurate than gibbs-sampling based on my data, and gibbs-sampling is theoretically a special case of metropolis-hastings in which $p = 1.0$. However, once p is reduced below 1.0, the accuracy seems to be unaffected by the change in p, as shown in Figure 3.

In Figure 4, the performance of each algorithm is shown when fed upstream evidence vs when fed downstream evidence. Figure 5 shows the Bayesian net used for these measurements. The query variable was node 3, and when measuring upstream evidence, the value of node 1 was supplied; When measuring downstream evidence, the value of node 9 was supplied (not the arrows indicate parent, not child, relations). The most surprising result here is the poor performance of metropolis-hastings given downstream evidence, when compared to the good performance of likelihood weighting: I would have assumed the opposite to be the case, considering likelihood weighting only takes ancestor nodes into account during variable assignment. This is another inquiry that may require further investigation.

1.3 Figures

		Polytree (10 nodes)		DAG (10 nodes)	
		Speed (sec)	% chance QV = True	Speed (sec)	% chance QV = True
Likelihood weighting	Mean	0.0219	85.40%	0.0375	64.68%
	STD	0.0018	0.93%	0.00504	1.73%
Gibbs sampling	Mean	0.0082	86.74%	0.0246	57.33%
	STD	0.0011	6.30%	0.00317	13.37%
Metropolis-hastings	Mean	0.2110	84.82%	0.6972	64.65%
	STD	0.2024	0.68%	0.5414	1.05%
Enumeration-ask	Mean	0.9562	85.27%	0.9582	64.51%
	STD	0.0624	0.00%	0.0545	0.00%

Table 1: Performance metrics for each inference algorithm measured on a polytree (10 nodes) and a DAG (10 nodes). (NOTE: Averages and STD DEV values are taken from 10 tests. 1000 samples were created for each test of the approximation algorithms)

		Polytree (5 nodes)		Polytree (15 nodes)	
		Speed (sec)	% chance QV = True	Speed (sec)	% chance QV = True
Likelihood weighting	Mean	0.0187	59.57%	0.05404	25.51%
	STD	0.0043	1.88%	0.01169	0.88%
Gibbs sampling	Mean	0.0123	59.90%	0.01329	28.32%
	STD	0.0032	6.46%	0.00231	13.75%
Metropolis-hastings	Mean	0.2411	59.53%	0.31	25.20%
	STD	0.1490	0.81%	0.2158	1.12%
Enumeration-ask	Mean	0.8969	59.16%	1.9035	25.23%
	STD	0.0777	0.00%	0.1583	0.00%

Table 2: Performance metrics for each inference algorithm measured on a polytree of 5 nodes and a polytree of 15 nodes. (NOTE: Averages and STD DEV values are taken from 10 tests. 1000 samples were created for each test of the approximation algorithms)

P	Speed (sec)	% chance QV = True
0.75	0.9646	52.30%
0.85	1.1601	52.29%
0.95	3.1784	52.36%
Exact inference	0.9577	51.98%

Figure 3: Performance of Metropolis-Hastings given various values for p, contrasted with the exact-inference performance.

	Downstream Evidence, Polytree (10 nodes)		Upstream Evidence, Polytree (10 nodes)	
	Speed (sec)	% chance QV=True	Speed (sec)	% chance QV=True
Likelihood weighting	0.0214	22.92%	0.0255	14.32%
	0.0023	1.17%	0.0076	1.00%
Gibbs sampling	0.0102	20.39%	0.0108	11.67%
	0.0020	9.69%	0.0044	4.60%
Metropolis-hastings	0.3414	47.79%	0.3245	20.60%
	0.2665	2.52%	0.2237	6.53%
Enumeration-ask	0.7032	23.30%	0.5582	14.61%
		0.00%		0.00%

Figure 4: Performance of approximate inference algorithms on upstream vs downstream evidence cases.

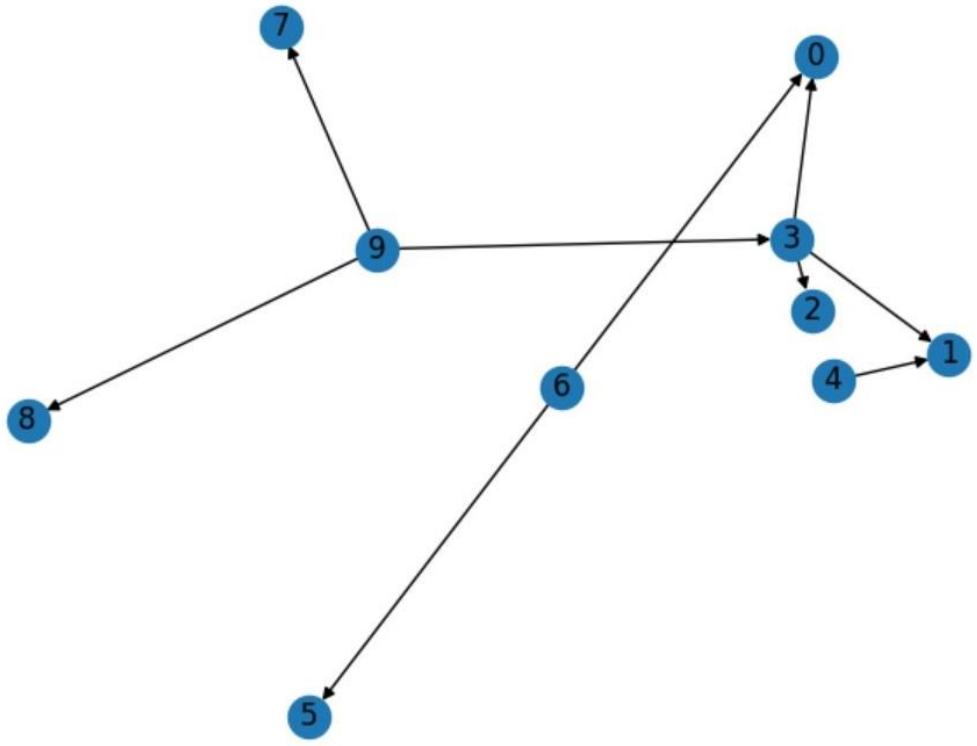

Figure 5: The polytree used when testing the metropolis-hastings algorithm on upstream and downstream evidence.