CSC242 Intro to AI
Project 3: Uncertain Inference

In this project you will get some experience with uncertain inference by implementing some of the algorithms for it from the textbook and evaluating your results. We will focus on Bayesian networks, since they are popular, well-understood, and well-explained in the textbook. They are also the basis for many other formalisms used in AI for dealing with uncertain knowledge.

Background

Let a Bayesian network be defined over the random variables $\{X\} \cup \mathbf{E} \cup \mathbf{Y}$, such that

- Query variable is X,
- Evidence variables are $\mathbf{E}$,
- Values for the evidence variables are $\mathbf{e}$,
- Unobserved (or hidden) variables are $\mathbf{Y}$.

Then the inference problem for Bayesian Networks is to calculate $\mathbf{P}(X \mid \mathbf{e})$, that is, the conditional distribution of the query variable given the evidence. In other words, compute the probability for each possible value of the query variable, given the evidence.

In general, we have that:

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha\, \mathbf{P}(X, \mathbf{e}) = \alpha \sum_{\mathbf{y}} \mathbf{P}(X, \mathbf{e}, \mathbf{y}) \qquad \text{(AIMA Eq. 13.9)}$$

And for a Bayesian network you can factor that full joint distribution into a product of the conditional probabilities stored at the nodes of the network:

$$P(x_1, \ldots, x_n) = \prod_{i=1}^{n} P(x_i \mid \textit{parents}(X_i)) \qquad \text{(AIMA Eq. 14.2)}$$

Therefore:

$$\mathbf{P}(X \mid \mathbf{e}) = \alpha \sum_{\mathbf{y}} \prod_{i=1}^{n} P(x_i \mid \textit{parents}(X_i))$$

Or in words: “a query can be answered from a Bayesian Network by computing sums of products of conditional probabilities from the network” (AIMA, page 523). These equations are the basis for the various inference algorithms for Bayesian networks that we’ve seen in class and in the textbook.

Representational Preliminaries

You will need to represent Bayesian networks in Java (or whatever language you’re using, but I recommend Java). What is a Bayesian network? As we saw in class:

- A random variable has a name and a domain: the set of its possible values.
- There is a node for every random variable.
- The nodes are arranged in a directed acyclic graph (DAG). In other words, each node has a set of child nodes.
- Each node stores the conditional probability distribution of its random variable given its parents’. In symbols:

  $$P(X_i \mid \textit{parents}(X_i))$$

  We will assume discrete domains for our variables, so these distributions can be stored as tables (or some other data structure that encodes the same information, of course).
- The query for a Bayesian network inference problem is the random variable for which you want the posterior distribution.
- The evidence for a Bayesian network inference problem is a set of random variables and a value for each of them (you might call this an “assignment”).
- The unobserved or hidden variables are all the other variables used in the network.
- The answer to a Bayesian network inference problem is a distribution for the query variable. That is, it’s a mapping from each of the possible values in the domain of the variable to the probability that the variable has that value. Note that these need to satisfy the axioms of probability.

You should think about how you would represent these elements of the problem. I will provide some code for you to use for this, but as always, you will learn more if you do it yourself, or at least try to do it yourself. Like representing propositional logic for purposes of doing inference, it’s a great exercise in object-oriented design.

You also need to think about how you will get problems into your program. I will give you code for parsers that read two quasi-standard file representations for Bayesian networks: the original BIF format and its successor XMLBIF. If you develop your own representation classes, which I highly recommend, you should be able to use them with my parsers with a little reverse-engineering. Having these at hand will allow you to test your programs on some of the many problems available on the Internet. My code bundle includes several example networks in these formats.

Part I: Exact Inference

For the first part of the project, you must implement at least one *exact inference* algorithm for inference in Bayesian networks. I recommend the “inference by enumeration” algorithm described in AIMA Section 14.4. Pseudo-code for the algorithm is given in Figure 14.9. Section 14.4.2 suggests some speedups for you to consider.

One comment from hard experience: As discussed in class, the inference by enumeration algorithm requires that the variables be set in topological order (so that by the time you need to lookup a probability in a variable’s node’s conditional probability table, you have the values of its parents). I think that the pseudo-code in the textbook does not make this clear.

I suggest you start by working on the Burglary/Earthquake Alarm problem (AIMA Figure 14.2) since it is well-described in the book and we did it in class. The XMLBIF encoding of the problem is included with my code.

Your implementation must have a `main` method (or whatever) that allows it to be used for different problems and queries. For exact inference, your program must accept the following arguments on the command-line:

- The filename of the BIF or XMLBIF encoding of the Bayesian network. You may assume that these filenames will end in “`.bif`” or “`.xml`”, as appropriate.
- The name of the query variable, matching one of the variables defined in the file.
- The names and values of evidence variables, again using names and domain values as defined in the file.

So for example, if this was a Java program, you might have the following to invoke it on the alarm example we did in class:

```
% java MyBNInferencer aima-alarm.xml B J true M true
```

That is, load the network from the XMLBIF file `aima-alarm.xml`, the query variable is B, and the evidence variables are J with value $true$ and M also with value $true$.

The “wet grass” example from the book (also included with my code) might be:

```
% java MyBNInferencer aima-wet-grass.xml R S true
```

The network is in XMLBIF file `aima-wet-grass.xml`, the query variable is R (for $Rain$) and the evidence is that S ($Sprinkler$) has value $true$.

The output of your program should be the posterior distribution of the query variable given the evidence. That is, print out the probability of each possible value of the query variable given the evidence.

Your writeup and/or README must make it very clear how to run your program and specify these parameters. If you cannot make them work as described above, you should check with the TAs before the deadline and explain the situation if for some reason it can’t be resolved. Note that for your own development, it is easy to setup Eclipse “run configurations” to run your classes with appropriate arguments, if you’re using Eclipse. You can also use `make` or similar tools.

Part II: Approximate Inference

For the second part of the project, you need to implement at least one *approximate inference* algorithm for inference in Bayesian networks. The algorithms described in the textbook and in class are: (1) rejection sampling, (2) likelihood weighting, and (3) Gibbs sampling. We have seen that the first two are fairly straightforward to implement (once you have the representational classes), but can be quite inefficient. Gibbs sampling is not that hard, although the part explained at the bottom of AIMA page 538 is a bit complicated. One warning: Gibbs sampling may require a large number of samples (look into the issue of “burn-in” in stochastic algorithms).

For running these approximate inferencers, you need to specify the number of samples to be used for the approximation. This should be the first parameter to your program. For example:

```
% java MyBNApproxInferencer 1000 aima-alarm.xml B J true M true
```

This specifies 1000 samples be used for the run. The distribution of the random variable B will be printed at the end of the run. If you need additional parameters, document their use carefully.

Writeup

Whichever algorithms you choose for each part, explain your choice, explain your design and implementation, and evaluate the results. You might well implement more than one algorithm in each category and compare them. You will need to run your implementations on multiple problems and document the results (which we can check by running your programs ourselves).

You should compare the approximate algorithm(s) to the exact algorithm(s). You should describe the performance of your implementations as the size of the problems grows. For the approximate algorithms, you should also evaluate and describe the performance (time, error) as the number of samples increases. Graphs are the best way to present this information—use them.

Additional general requirements for writeups are the same as for previous projects.

Grading

Exact Inference	35%
Approximate Inference	35%
Writeup	30%

Extra credit:

- Second approximate inference algorithm: up to 10%
- Gibbs sampling: up to 20%
- Other: Describe thoroughly in writeup; grader's discretion

Maximum possible extra credit is 20% total.

Collaboration Policy (same as before)

You will get the most out of this project if you write the code yourself.

You may discuss the project and program design with others, but you should try to write the code yourself.

If you do your own work, you *must* include the following statement near the start of your README file: “I did not collaborate on this project and all work is my own.”

If you feel it would be helpful, you may collaborate on the code with other students in groups of no more than three (3). You *must* report the names of your collaborators near the start of your README file.

You may NOT collaborate on your writeup. Your writeup must be your own work. Attribute any material that is not yours. Cite any references used in your writeup. Plagiarism is cheating.

Disclaimer

I promise you that I have used my code to implement all of these techniques and tested them on several of the problems included with the code.

Nonetheless, my code may contain bugs. Please report these clearly via email. Bug reports with fixes are particularly welcome. I will of course try to fix any bugs and share the results. However I will not be held responsible if someone discovers a bug the night before the project is due even if it makes their entire project fail.