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

A Bayesian network is defined over the random variables $\{X\} \cup \mathbf{E} \cup \mathbf{Y}$, where

- X is the query variable
- $\mathbf{E}$ are the evidence variables
- $\mathbf{e}$ are the observed values for the evidence variables
- $\mathbf{Y}$ are the unobserved (or hidden) variables

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 (observed values of the evidence variables). In other words, compute the probability of 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.

Representational Preliminaries

You will need to represent Bayesian networks in Java (or whatever language you're using, but I recommend Java). Ask yourself what is a Bayesian network. **THINK ABOUT THIS YOURSELF NOW.** Make a list of objects (classes) and their properties. Then look at the following list.

- A Bayesian network is a directed acyclic graph (DAG) of random variables, with a probability distribution stored at each node.
- A random variable has a name and a domain: the set of its possible values.
- There is a node in the network for every random variable.
- The nodes are arranged in a DAG. In other words, each node has a set of child nodes.
- Root nodes (with no parents) store the prior probability distribution of their random variable.
- Non-root nodes store the conditional probability distribution of their random variable given their parents': $P(X_i \mid \textit{parents}(X_i))$. We will assume discrete domains for our variables, so these distributions can be represented as tables, however we choose to implement that.
- 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 (I 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, given the evidence. Note that distributions need to satisfy the axioms of probability.

I hope that your list of requirements looked like mine. Now you need to think about how you will represent these elements of the problem. **YOU WILL LEARN THE MOST IF YOU DO THIS YOURSELF.** Like representing propositional logic for purposes of doing inference, it's a great object-oriented programming exercise. However if you give it a solid try and just can't get it right, you can use the code that I will provide for this, once you understand how it works.

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: 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 don't think that the pseudo-code in the textbook makes 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 do 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:

```
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
3. Gibbs sampling

The first two are straightforward to implement once you have the representation of Bayesian networks and their components, but they can be very 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 algorithms to the exact algorithms. 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, as described below.

Project Submission

Your project submission MUST include the following:

1. A README.txt file or PDF document describing:
   (a) Any collaborators (see below)
   (b) How to build your project
   (c) How to run your project's program(s) to demonstrate that it/they meet the requirements
2. All source code and build files for your project (see below)
3. A writeup describing your work in PDF format (see below)

The TAs must be able to cut-and-paste from your documentation in order to build and run your code. **The easier you make this for them, the better grade your will be.**

Programming Practice

You may use Java, Python, or C/C++ for this project. I recommend that you use Java. Any sample code we distribute will be in Java. Other languages (Haskell, Clojure, Lisp, *etc.*) by arrangement with the TAs only. Do not use any non-standard libraries without consulting the TAs.

Use good object-oriented design. Comment your code liberally.

Your code must build on the CSUG machines. All CSC majors have CSUG accounts. If you are not a CSC major or have forgotten that you have a CSUG account, you can see Marty Guenther. The TAs are available to help with this, but **do not wait until the last minute**.

Writing Up Your Work

As noted above, it is crucial that you present your work clearly, honestly, and in its best light. We will give lots of credit for good writeups. We will not like submissions with sloppy, unstructured, hard to read writeups that were clearly written at the last minute.

Your goal is to produce a technical report of your efforts for an expert reader. You need to convince the reader that you knew what you were doing and that you did it as best you could in the (entire) time available.

Write up what you did (and why). If you didn't get to something in your code, write about what you might have done. (There's always *something* you might have done.) If something didn't work, write about why and what you would do differently. But don't write "I would have started sooner." Your readers already know that.

Your report *must be your own words*. Material taken from other sources must be properly attributed if it is not digested and reformulated in your own words. **Plagiarism is cheating.**

Start your writeup early, at least in outline form. Document your design decisions as you make them. That way half the write-up is done by the time you're running the program. Don't forget to include illustrations of the program in action (traces, screenshots) and some kind of evaluation.

Your report must be typeset (not handwritten). Using LaTeX and BibTeX makes things look nice and is worth learning anyway if you don't know it, but it's not a requirement. Your report must be submitted as a PDF file. If you want to use a word processor, be sure to generate and submit a PDF from it, not the document file.

The length of the report is up to you. For a CSC242 assignment, I would say that 3–5 pages is the *minimum* that would allow you to convince the reader. Be sure to include screenshots and/or traces (which wouldn't count towards the 3-5 pages minimum of actual text).

Late Policy

Don't be late. But if you are: 5% penalty for the first hour or part thereof, 10% penalty per hour or part thereof after the first.

Collaboration Policy

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

That said, collaboration on the coding portion of projects is permitted, subject to the following requirements:

- Groups of no more than 3 students, all currently taking CSC242.
- You must be able to explain anything you or your group submit, IN PERSON AT ANY TIME, at the instructor's or TA's discretion.
- One member of the group should submit code on the group's behalf in addition to their writeup. Other group members should submit only their writeup.
- All members of a collaborative group will get the same grade on the coding component of the project.

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.