CSC442 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

Recall that a Bayesian network is directed acyclic graph whose vertices are 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, Python, or C++ (or whatever language you're using, by special arrangement with the TAs.) Ask yourself what is a Bayesian network. **THINK ABOUT THIS YOURSELF NOW.** Make a list of classes and their properties. Then compare to 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 parent nodes and a set of child nodes (both possibly empty).
- 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 multi-column tables.
- The query variables for a Bayesian network inference problem are the random variables 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. Like representing propositional logic for purposes of doing inference, it's a great programming exercise.

You also need to think about how you will get problems into your program. You will need to process the semi-standard `XMLBIF` representation as discussed in class. The project zipfile includes some example XMLBIF representations of networks from the textbook. Additional networks may be posted over time.

Part I: Exact Inference

For the first part of the project, you must implement 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.

Your implementation must be able to handle different problems and queries. For exact inference, your program must accept the following arguments on the command-line:

- The filename of the XMLBIF encoding of the Bayesian network. You may assume that these filenames will end in "`.xml`".
- 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 Python program, you might have the following to invoke it on the alarm example:

```
python mybninferencer.py 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 $true$.

Similarly, if you choose to use Java, then 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 algorithms for each of the *approximate inference* techniques for 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

For the algorithms that you implement, explain your design and implementation briefly, and evaluate the results. 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. See the textbook for examples. No graphs: points off.

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

Rubric

- 15% – XMLBIF parsing and network representation.
- 15% – Exact Inference
- 10% – Rejection Sampling
- 10% – Likelyhood Weighting Sampling
- 15% – Gibbs Sampling
- 35% – Writeup and Experimental Work
- *10% – Bonus Points for Variable Elimination*

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)

We must be able to cut-and-paste from your documentation in order to build and run your code. **The easier you make this for us, the better grade your will be.** It is your job to make both the building and the running of programs easy and informative for your users.

Programming Practice

Use good object-oriented design. No giant `main` methods or other unstructured chunks of code. Comment your code liberally and clearly.

You may use Java, Python, or C/C++ for this project. Other languages (Haskell, Clojure, Lisp, *etc.*) by arrangement with the TAs only. Do **not** use any non-standard libraries which are against the spirit of the assignment. E.g., don't download a propositional theorem prover from github and import it into your code!

If you use Eclipse, you *must* make it so that we can build and run your project without Eclipse. Document exactly what needs to be done in your README (Makefile, `javac` or `gcc` incantations, whatever). Eclipse projects with no build instructions will receive 0.

Your code must build on Fedora Linux using recent versions of Java, Python, or `gcc`.

- This is not generally a problem for Java projects, but don't just submit an Eclipse project without the build and run instructions detailed above.

- Python projects must use Python 3 (recent version, like 3.6.x). Mac users should note that Apple ships version 2.7 with their machines so you will need to do something different.
- If you are using C or C++, you must use "`-std=c99 -Wall -Werror`" and have a clean report from `valgrind`. And you'd better test on Fedora Linux or expect problems.

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 CSC442 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. If there are extenuating circumstances please contact me (Adam) as soon as possible!

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 CSC442.
- 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. For the submission this means that EVERYONE must submit SOMETHING to blackboard – even if it's just a PDF of the writeup and a note that someone you worked with will be submitting the code.