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.

Recall that a Bayesian network is a 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 (observed) variables
- $\mathbf{Y}$ are the unobserved (or hidden) variables

The inference problem for Bayesian Networks is to calculate $\mathbf{P}(X \mid \mathbf{e})$, where $\mathbf{e}$ are the observed values for the evidence variables. That is, inference in Bayesian networks involves computing the posterior distribution of the query variable given the evidence. In other words, inference in Bayesian networks involves computing a probability for each possible value of the query variable, given the evidence.

In general, from the definition of conditional probability and the process of marginalization, 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, for a Bayesian network:

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

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.

Suggestions for Success

You will need to implement a representation of Bayesian networks and several inference algorithms that use the representation. The details are in the next section. But let's talk first about how to go about this.

You know what I'm going to say, right? Always start by designing your data structures.

So what are the elements of a Bayesian network. What about a Bayesian network inference problem? What about a Bayesian network inference problem solver?

THINK ABOUT THIS YOURSELF NOW, THEN READ ON.

DID YOU THINK ABOUT IT?

REALLY?

WITH A WHITEBOARD OR A PIECE OF PAPER IN FRONT OF YOU?

IF SO, READ ON.

I thought first of a directed graph. A graph is a set of nodes (vertices) that can have both parents and children. Everyone in CSC242 should be comfortable with graphs.

In a Bayesian network, each node in the graph is associated with a random variable. Each random variable has a name and a domain of possible values. We will assume discrete domains for this project, although you are welcome to try problems involving continuous domains also if you like.

Each node in the graph stores a probability distribution. 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 the values of their parents': $P(X_i \mid \textit{parents}(X_i))$.

A Bayesian network inference problem involves a query random variable and some evidence. The query variable 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 would call this an "assignment," in keeping with our terminology from Unit 2.

The answer to a Bayesian network inference problem is a posterior distribution for the query variable given the evidence. 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.

Finally, a Bayesian network inference problem solver takes a Bayesian network and a problem (query and evidence, using variables from that network), and has one or more methods that compute and return the posterior probability distribution of the query variable given the evidence. The solver may have additional properties or parameters if needed.

Ok. I hope that your list of requirements looks something like mine.

Implementation

Now it’s on to the implementation. You will need to represent these elements of Bayesian networks in Java (or whatever language you’re using, but I recommend Java). You’ll be using things like graphs, sets, and mappings (dictionaries), with which you should already be familiar. Give it a try. It’s a great object-oriented programming exercise.

SERIOUSLY, GIVE IT A TRY. THEN READ ON FOR SOME THOUGHTS.

Start writing some classes and/or interfaces. Don't sweat the details. Get the relationships between the elements right. Graphs, variables, domains, assignments: these should be fairly straightforward.

One tricky thing to design is the representation of probability distributions. Drill down a bit and implementations will suggest themselves.

A prior (or unconditional) probability distribution is a mapping from the possible values of a variable to the probability that the variable has that value. So a mapping... sounds like a hashtable, right? But also consider that the domains are generally not very large, so maybe a simple array or list of value-probability pairs would be easier and perhaps even faster in practice.

A posterior (or conditional) probability distribution is a mapping from tuples of values of the conditioning variables to a distribution with a probability for each possible value of the variable (like above). So again, it's a mapping, but again that doesn't necessary mean a hashtable. Hashtables are a bit tricky to use in Java except on builtin types (see the description of `Object.hashCode`).

Instead, I would think about how these are drawn in AIMA and in class: a conditional probability table. The size of the table (rows and columns) is known once the network topology is defined. A multicolumn table could be imlemented as a... multidimensional array. Think of how you would lookup an entry in such a table given an assignment of values to variables. Again, I think you'll find that a hashtable is not ideal for this.

Now go write some code.

TRY IT. THEN READ ON.

SERIOUSLY, TRY WRITING THE CODE.

Write each class separately and include simple tests in a `main` method that you can run. Do something else if you don't use Java, but why wouldn't you use Java for a project like this?

Ultimately, your data structures need to support the operations required by the inference algorithms that you need to implement. So start writing those, and adapt your data structures as necessary.

This is called "iterative design."

TRY IT. YOU WILL LEARN THE MOST IF YOU DO THIS YOURSELF.

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. But be warned that it may be harder to figure out how to use my code than just designing and writing your own.

Reading Networks From Files

You also need to think about how you will get Bayesian Networks into your program(s). This would be very, very painful to do by hand like you did for constraint satisfaction problems in Project 2.

We will give you Java code for parsers that read two quasi-standard file representations for Bayesian networks: the original BIF format (Cozman, 1996) and its successor XMLBIF (Cozman, 1998). The download for this project contains several example networks (see below for descriptions), and there are many others available on the Internet.

If you develop your own representation classes, which I highly recommend, you should definitely be able to use the XMLBIF parser with a little reverse-engineering. The BIF parser might be harder, since it was itself generated from a grammar of the BIF format using ANTLR.

Whatever you do, you **must** be able to read XMLBIF files and *should* be able to read BIF files. Explain in your README which formats your programs can handle.

Requirements

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.

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 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 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 -cp "./bin" 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 -cp "./bin" 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 must 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 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. You can also use `make` or similar tools for running your programs from the comamnd-line.

If you are using Eclipse and cannot for some reason also make the programs run from the command-line, you MUST setup Eclipse "run configurations" to run your classes with appropriate arguments, and document their use in your README.

Part II: Approximate Inference

For the second part of the project, you need to implement algorithms for *approximate 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).

You *must* implement the first two algorithms. You *may* implement Gibbs Sampling for extra credit up to 20%.

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 -cp "./bin" 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 in your README.

Bayesian Network Examples

You should test your inference problem solvers on the AIMA earthquake alarm example (Fig. 14.2) and the AIMA "wet grass" example (Fig 14.12). You should try different combinations of query variable and evidence. You can work out the correct answers by hand if necessary. Note that these both use only boolean variables, but in general variables may have any (discrete) domain.

Here are a few more examples of Bayesian networks. We may use some of these to test your program (in addition to the AIMA problems), and we may use some that are not listed here.

- The "dog problem" network from (Charkiak, 1991).
- The SACHS protein-signaling network from (Sachs et al., 2005).
- The ALARM network for determing whether to trigger an alarm in a patient monitoring system, from (Beinlich et al., 1989).
- The INSURANCE network from (Binder et. al, 1997).
- The HAILFINDER network from (Abramson et al., 1996).

All of these are provided in the download for this project.

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 for your project. Eclipse projects must include the project settings from the project folder. Non-Eclipse projects must include a `Makefile` or shell script that will build the program per your instructions, or at least have those instructions in your README.txt.
3. A completed copy of the submission form posted with the project description. Projects without this will receive a grade of 0. If you cannot complete and save a PDF form, submit a text file containing the questions and your (brief) answers.

Writeups other than the instructions in your README and your completed submission form are **not** required.

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. 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.

You may **not** use any non-standard libraries. Python users: that includes things like NumPy. Write your own code—you'll learn more that way.

If you use Eclipse, make it clear how to run your program(s). Setup Build and Run configurations as necessary to make this easy for us. Eclipse projects with poor configuration or inadequate instructions will receive a poor grade.

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 should use reasonable "object-oriented" design not a mish-mash of giant functions. If you need a refresher on this, check out the C for Java Programmers guide and tutorial. You **must** use "`-std=c99 -Wall -Werror`" **and** have a clean report from `valgrind`. Projects that do not follow both of these guidelines will receive a poor grade.

Late Policy

Late projects will **not** be accepted. Submit what you have by the deadline. If there are extenuating circumstances, submit what you have before the deadline and then explain yourself via email.

If you have a medical excuse (see the course syllabus), submit what you have and explain yourself as soon as you are able.

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 a README indicating who their collaborators are.
- All members of a collaborative group will get the same grade on the project.

Academic Honesty

Do not copy code from other students or from the Internet.

Avoid Github and StackOverflow completely for the duration of this course.

There is code out there for all these projects. You know it. We know it.

Posting homework and project solutions to public repositories on sites like GitHub is a violation of the University's Academic Honesty Policy, Section V.B.2 "Giving Unauthorized Aid." Honestly, no prospective employer wants to see your coursework. Make a great project outside of class and share that instead to show off your chops.

References

Abramson, B., J. Brown, W. Edwards, A. Murphy, and R. L. Winkler (1996). Hailfinder: A Bayesian system for forecasting severe weather. *International Journal of Forecasting*, 12(1):57-71.

Andreassen, S., R. Hovorka, J. Benn, K. G. Olesen, and E. R. Carson (1991). A Model-based Approach to Insulin Adjustment. In *Proceedings of the 3rd Conference on Artificial Intelligence in Medicine*, pp. 239-248. Springer-Verlag.

Beinlich I., Suermondt H.J., Chavez R.M., Cooper G.F. (1989). The ALARM Monitoring System: A Case Study with Two Probabilistic Inference Techniques for Belief Networks. *Proceedings of the 2nd European Conference on Artificial Intelligence in Medicine*, pp. 247–256.

Binder, J., D. Koller, S. Russell, and K. Kanazawa (1997). Adaptive Probabilistic Networks with Hidden Variables. *Machine Learning*, 29(2-3):213–244.

Charniak, E. (1991). Bayesian Networks Without Tears. *AI Magazine*, Winter 1991, pp. 50–63.

Cozman, F. (1996). The Interchange Format for Bayesian Networks [BIF]. Website at http://sites.poli.usp.br/p/fabio.cozman/Research/InterchangeFormat/xmlbif02.html (accessed October 2018).

Cozman, F. (1998). The Interchange Format for Bayesian Networks [XMLBIF]. Website at http://sites.poli.usp.br/p/fabio.cozman/Research/InterchangeFormat/

`index.html` (accessed October 2018).

Sachs, K., O. Perez, D. Pe'er, D. A. Lauffenburger and G. P. Nolan (2005). Causal Protein-Signaling Networks Derived from Multiparameter Single-Cell Data. *Science*, 308:523-529.