6.825 Project 2

Due: Thursday, Nov. 9 at 5PM
Submit: In class or to box outside of Tomas' office (32-G492)

In this assignment, you will implement some Bayesian network inference algorithms and then study their performance in different conditions. To help in the implementation and testing of the algorithms, we have made available a set of Java tools. We strongly suggest that you use them, and write your programs in Java. But if you’d like to go it on your own with no support from us (except in clarifying the algorithms), then you may do this assignment in any language.

Note that the most important part of this assignment is running experiments with your code, making graphs from the data, and writing up some insights based on the data. You have a little over three weeks in total for this assignment, but with the midterm in the middle. One good strategy might be to allocate about one week to implementing and testing variable elimination, about a week to implement the sampling algorithms, and about one week to running the experiments and doing the write-up.

You may do this assignment in pairs. If you choose to do it by yourself, you still must complete all of the tasks.

1 Given Networks

You will be working with four different Bayesian networks; one is trivial and the other three are moderately complex in different ways.

The first is the Burglary network, extracted from Figure 14.2, page 494 of the second edition of AIMA. The network models two neighbors, John and Mary, who have promised to call you when they hear the alarm, which can be set off by either a burglary or an earthquake. The net can then be used to find the probability of a burglary happening given that both John and Mary call, i.e., \( P(\text{Burglary} | \text{JohnCalls} = \text{true}, \text{MaryCalls} = \text{true}) \). For the CPTs in the network, \( P(\text{Burglary} | \text{JohnCalls} = \text{true}, \text{MaryCalls} = \text{true}) = 0.2841718 \).

The second network is the Neighbor network, which is for a particular scenario of neighborhood events, that shows how even distant concepts have some connection. This network, shown in Figure 1, has 23 nodes.

The third network is a well-known Alarm network developed to monitor patients in the intensive care unit. As shown in Figure 2, it consists of 37 nodes and 752 parameters. For example, this Alarm network relates the probability of the BP (blood pressure) to other information about the patient. So, given positive responses for anaphylaxis and LVFAILURE (left ventricular failure), we can estimate the blood pressure distribution of the patient; \( P(\text{BP} | \text{ANAPHYLAXIS} = \text{TRUE}, \text{LVFAILURE} = \text{TRUE}) \) is the distribution shown in Table 1.

The fourth network provided is the Carpo network, which is designed for diagnosing carpal tunnel syndrome, shown in Figure 3. It was extracted from \http://www.cs.huji.ac.il/labs/compbio/Repository/networks.html\ and is an example of a non-loopy network. Unfortunately,
Table 1: Distribution over blood pressure given positive responses for anaphylaxis and left ventricular failure.

<table>
<thead>
<tr>
<th>Blood Pressure</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIGH</td>
<td>0.0122795</td>
</tr>
<tr>
<td>LOW</td>
<td>0.9670573</td>
</tr>
<tr>
<td>NORMAL</td>
<td>0.020663</td>
</tr>
</tbody>
</table>

this network was *sanitized* before being provided; i.e., for privacy reasons, the network variable names and their values were all converted to numbers.

## 2 Java Code

It’s a good idea to start by reading through the documentation and interfaces of the classes we provide; using them will simplify your job considerably. Here is an overview of the support infrastructure:

- **BayesNet** consists of a **BayesNetNodeSet**, which is a set of **BayesNetNodes**. Each **BayesNetNode** consists of a **FunctionVariable**, a **BayesNetNodeSet** of parents and a conditional probability table represented as a **Function**.

- A **Function** has a **FunctionVariableSet** of its input **FunctionVariables** and a table of **FunctionEntries** that enumerate the entire domain of the function. **Functions** are total over their domain.

- A **FunctionEntry** maps an **Assignment** to a **double**. **FunctionEntries** are totally ordered, based on the ordering of **Assignments**. To get the index of a particular assignment in a Function, you can use the computePosition() method of the Assignment class.

- An **Assignment** maps the **FunctionVariables** in a **FunctionVariableSet** to values in their respective **Domains**. **Assignments** are totally ordered, based on the total ordering of their **FunctionVariableSet** and the total ordering of the **Domains** that those **FunctionVariables** range over. The **FunctionVariableSet** class has an iterator method called assignmentIterator() that returns assignments over the variables in the **FunctionVariableSet** in the proper order.

- A **FunctionVariableSet** is a totally ordered set of **FunctionVariables** (which are ordered lexicographically). Remember to order the variables lexicographically when making a new **Assignment** (such as in Figure 4).

- A **Domain** contains **Comparable** values: that is, **Domains** are totally ordered. Iterating over a domain returns **DomainIndex** objects. You can use the getValue() method to find the **Comparable** for a **DomainIndex**.

You will need to import edu.mit.six825.bn.bayesnet.* and edu.mit.six825.bn.functiontable.* to use the above classes and you’ll want to make sure that the provided jar file (BN.jar) is in your class path.
Nets (edu.mit.six825.bn.inputs.Nets) has the Burglary, Neighbor, Alarm, and Carpo networks built in to it. A Bayes net can be queried using a Solver (edu.mit.six825.bn.bayesnet.Solver), as shown in Figure 4. While it is not necessary to inherit from this class, it provides helpful methods for storing and managing the net and evidence. An example solver, called the EnumerationSolver (edu.mit.six825.bn.bayesnet.EnumerationSolver) has been provided, and is used in the figure to query the net. The code has print statements added to it, so that executing the code in the figure should be instructive.

A utility class Random (techniques.utils.Random) has also been provided, which has simple methods to return a randomly generated number and to uniformly select one of a number of values given a probability distribution for them.

3 Grading

Grades for this project will be assigned based on two different components: a written report and an interview.

3.1 Report

Please hand in a coherently written, typed report, containing:

- Answers to all the questions. Feel free to include interesting implementation details or insights.
- Graphs with clearly labeled axes, scales and titles.
- An appendix listing your code. (We don’t intend to grade or read it, but like to have it just in case we have problems with non-uniqueness of solutions. Remember that each team must write all their own code.)

You should think of this assignment as a research project, in which the aim is to use your code as a tool to conduct experiments and then to write a paper reporting what you found. The quality of writing should be similar to that found in a conference or workshop paper. If possible, make it a fun read.

If you choose to work in a pair, you may submit 1 report for two people.

3.2 Interviews

We will conduct individual interviews of everyone in the class who hands in a written report. So even if you work in a pair, the interviews will be done separately. We will ask you questions about the algorithms and topics covered in the assignment. These questions may include implementation details from any part of the project: therefore, even if you work in a pair, it is important that both people in the pair thoroughly understand all the techniques in the assignment and know how all of the algorithms were implemented.

4 Tasks

1. We’ll begin with variable elimination, which is an exact algorithm. Start by implementing a procedure that, given a Bayesian network, assignments of values to some subset of the
variables \( e \), a query node \( X \), and an elimination order, calculates \( P(X|e) \) in that network. We recommend you follow the Koller and Friedman version of the algorithm, because the Russell and Norvig text hides a lot of the detail. In particular, create all the factors (restricted with respect to the evidence, if appropriate) before eliminating any variables.

We have given you the enumeration algorithm as a basis for comparison. Start by implementing \( \text{VE} \) and then make sure that your implementation is correct by comparing the results in the Burglary network on the following queries:

\[
\begin{align*}
P(\text{Burglary}|\text{JohnCalls} = \text{true}, \text{MaryCalls} = \text{true}) \\
P(\text{Earthquake}|\text{JohnCalls} = \text{true}, \text{Burglary} = \text{true})
\end{align*}
\]

and then comparing the result in the Alarm network on the following query with the ones in Table 1:

\[
P(\text{BP}|\text{ANAPHYLAXIS} = \text{TRUE}, \text{LV FAILURE} = \text{TRUE})
\]

Note that the Burglary network has variables that take boolean values (ComparableBoolean.TRUE and ComparableBoolean.FALSE), whereas the other three networks have variables, the values of which are strings, such as “FALSE” or “NORMAL”.

2. (a) In the Neighbor network, assume you know that Tom’s cousin is not visiting him and there was not lots of traffic on 5th street last year. Consider the probability that Hank moves away sometime in the next 5 years as \( P(\text{HM}|\text{CV} = \text{FALSE}, \text{HT} = \text{FALSE}) \). And there are two more events (Molly gets elected mayor (\( \text{MM} \)). Tom is currently fairly wealthy (\( \text{TW} \)) that could happen. From the structure of the network Figure 1, which of the two events would you expect could better convince Hank to stay (not move away)? Calculate and explain why Hank changes his mind for the following queries:

\[
\begin{align*}
P(\text{HM} \mid \text{CV} = \text{FALSE}, \text{HT} = \text{FALSE}, \text{MM} = \text{TRUE}) \\
P(\text{HM} \mid \text{CV} = \text{FALSE}, \text{HT} = \text{FALSE}, \text{TW} = \text{TRUE})
\end{align*}
\]

(b) Consider the following probability distribution in the Alarm network.

\[
P(\text{BP} \mid \text{ANAPHYLAXIS} = \text{FALSE}, \text{INTUBATION} = \text{NORMAL}, \text{MINVOLSET} = \text{HIGH})
\]

From Figure 2, how would you expect the results to change if it were also known that the HR (heart rate) of this patient is normal? What if you knew that this patient does not have a history of left ventricular failure? Calculate and explain why the values of the probabilities changed in that manner for the following queries:

\[
\begin{align*}
P(\text{BP} \mid \text{ANAPHYLAXIS} = \text{FALSE}, \text{HR} = \text{NORMAL}, \text{INTUBATION} = \text{NORMAL}, \text{MINVOLSET} = \text{HIGH}) \\
P(\text{BP} \mid \text{ANAPHYLAXIS} = \text{FALSE}, \text{HISTORY} = \text{FALSE}, \text{INTUBATION} = \text{NORMAL}, \text{MINVOLSET} = \text{HIGH})
\end{align*}
\]
3. (a) For each of the four queries in task 2 try at least 10 (but more will give you clearer results) random elimination orders, and show a histogram of the time taken. In the eliminations, it is possible for a factor to be so large that an out-of-memory error can happen, in which case you can just treat it as a really long run time. Are the histograms from the different queries or problems interestingly different? Why or why not?

(b) One heuristic for choosing an elimination order is to be greedy: on each iteration of variable elimination, choose to eliminate the variable that will result in the smallest factor being created. Implement this heuristic and apply it to the cases studied in the task 2. How long does it take to run in each case? Explain how this result relates to those you obtained in (a).

4. Consider the Carpo network. Since this network was imported from an external source it should be noted that the variable values are the strings “0”, “1”, etc. Calculate the following probabilities with both random elimination orders and the greedy elimination order:

\[ P(N104|N41 = "2", N84 = "1", N116 = "0") \]

and

\[ P(N73|N152 = "1", N116 = "0", N43 = "1") \]

How long does it take to run in each case? Explain the influence of elimination order to the Carpo network compared to the other two networks in task 3.

5. Now we’ll consider two approximate inference algorithms, likelihood weighting and Gibbs sampling. Since we will not get the exact answer, we’ll have to measure performance in terms of running time versus quality of the result.

Implement likelihood weighting and Gibbs sampling. Test them on the queries from tasks 1 and 2.

6. For Gibbs sampling, experiment with throwing away an initial prefix of the examples (before the process has converged to the stationary distribution). How does the length of the initial prefix that is thrown away affect your estimates?

7. How do we measure the quality of an approximate inference algorithm? Ultimately, we’re computing a probability distribution over some variable given evidence, \( \hat{P}(X|e) \). If you know the true probability distribution \( P \), then the Kullback-Leibler divergence between the two distributions, \( D(P, \hat{P}) \) is given by

\[
\sum_{x \in X} P(x|e) \log \left( \frac{P(x|e)}{\hat{P}(x|e)} \right).
\]

Use the true answer distributions to the problems in task 2. In each of those problems, run likelihood weighting and Gibbs sampling (using the strategy you think is best based on your experiments in task 6). Plot number of samples on the \( x \) axis versus KL divergence of your answer from the true answer on the \( y \) axis. Be sure to choose a maximum value of \( x \) so that you get a nearly optimal answer.
Do this 10 times on each problem, and plot all 10 curves on the same set of axes. (So your answer for each algorithm should be 4 graphs, each of which has 10 curves). Also provide graphs with the mean KL divergences measured against the number of samples for each algorithm for the different problems.

Find four interesting things to say about these graphs.

At what number of samples does the computation time equal the amount of computation you spent on these problems in task 2? Which method would these results lead you to prefer for these problems? (May be different answers for the different problems).

8. **Optional** Try the variation of Gibbs sampling in which you choose the variables to flip uniformly at random, rather than in systematic passes. Compare the performance of this method with that of the systematic method.
Figure 1: The Neighbor Network
Figure 2: The Alarm Network
Figure 3: The Carpo network.
import edu.mit.six825.bn.inputs.*;
import edu.mit.six825.bn.bayesnet.*;
import edu.mit.six825.bn.functiontable.*;

BayesNet bn = Nets.getBurglary();
FunctionVariable[] vars = new FunctionVariable[2];
Comparable[] vals = new Comparable[2];

vars[0] = bn.nodes.getNode("JohnCalls").var;
vars[1] = bn.nodes.getNode("MaryCalls").var;
vals[0] = ComparableBoolean.TRUE;
vals[1] = ComparableBoolean.TRUE;

Assignment evidence = new Assignment(new FunctionVariableSet(vars), vals);

Solver enumSolver = new EnumerationSolver(bn);
// ...GibbsSamplerSolver(bn);
// ...LikelihoodWeightingSolver(bn);
// ...VariableEliminationSolver(bn);

enumSolver.setEvidence(evidence);

System.out.println(enumSolver.query(bn.nodes.getNode("Burglary")));

Figure 4: Querying the net