6.825 Project 2

Due: Thursday, Nov. 4 at 5PM
Submit to: box outside of Tomas' office (32-D542)

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 the TAs (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 three weeks in total for this assignment. One good strategy might be to allocate one week to implementing and testing variable elimination, a week to implement the sampling algorithms, and one week to running the experiments and doing the write-up.

1 What to Submit

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.

2 Given Networks

You will be working with three different Bayesian networks; one is trivial and the other two 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., $P(Burglary|JohnCalls = true, MaryCalls = true)$. For the CPTs in the network, $P(Burglary|JohnCalls = true, MaryCalls = true) = 0.2841718$.

The second network is the *Insurance* network, which is for evaluating car insurance risks. This network, shown in Figure 1, was extracted from `http://www.cs.huji.ac.il/labs/compbio/Repository/networks.html`; it has 27 nodes. Typically only 15 of the nodes are observable in the network and there are over 1400 parameters. The network allows us to compute the probability of the property cost (of both cars for accidents or owned car for thefts) given information about the driver. For example, given an adolescent driving a car with about 50,000 miles without anti-lock brakes, we can estimate the property cost. Here, $P(PropCost|Age = Adolescent, Antilock = False, Mileage = FiftyThou)$ is the distribution shown in table 1.

Property Cost	Probability
Thousand	0.4572275
Ten Thousand	0.3427002
Hundred Thousand	0.1729787
Million	0.0270935

Table 1: Distribution over property cost for an adolescent driving a car with 50,000 miles and without anti-lock brakes.

The third network provided is the *Carpo* network, which is designed for diagnosing carpal tunnel syndrome, shown in Figure 2. 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, this network was *sanitized* before being provided; i.e., for privacy reasons, the network variable names and their values were all converted to numbers.

3 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. FunctionEntrys 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.

- 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 3).

- 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, Insurance, 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 3. 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.

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 VE and then make sure that your implementation is correct by comparing the results in the Burglary network on the following queries:

$$P(Burglary|JohnCalls = true, MaryCalls = true)$$

$$P(Earthquake|JohnCalls = true, Burglary = true)$$

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

$$P(PropCost|Age = Adolescent, Antilock = False, Mileage = FiftyThou)$$

Note that the Burglary network has variables that take boolean values (ComparableBoolean.TRUE and ComparableBoolean.FALSE), whereas the other two networks have variables, the values of which are strings, such as "False" or "FiftyThou".

2. (a) Consider $P(PropCost|Age = Adolescent, Antilock = False, Mileage = FiftyThou)$. From Figure 1, how would you expect the results to change if it were also known that the car was sporty? What if you knew that the driver was a good student?

   Calculate and explain why the values of the probabilties changed in the manner for the following queries:

$$\begin{aligned} P(PropCost \quad | \quad & Age = Adolescent, Antilock = False, Mileage = FiftyThou, \\ & MakeModel = SportsCar) \end{aligned}$$

   and

$$\begin{aligned} P(PropCost \quad | \quad & Age = Adolescent, Antilock = False, Mileage = FiftyThou, \\ & GoodStudent = True) \end{aligned}$$

   (b) 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:

$$P(N112|N64 = \text{“3”}, N113 = \text{“1”}, N116 = \text{“0”})$$

   and

$$P(N143|N146 = \text{“1”}, N116 = \text{“0”}, N121 = \text{“1”})\ .$$

3. 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?

4. 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 part 2. How long does it take to run in each case? Explain how this result relates to those you obtained in part 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

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

   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.

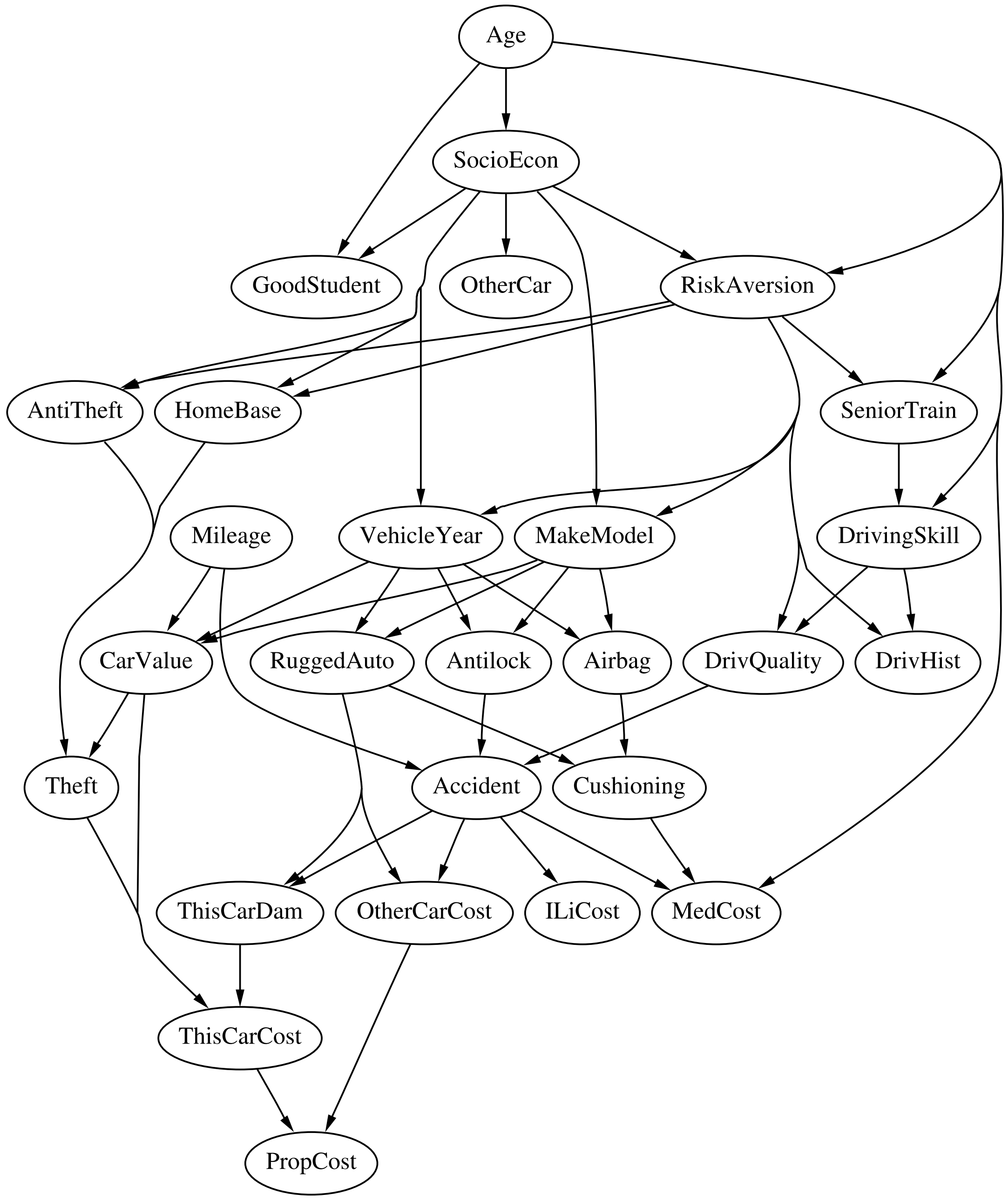

Figure 1: The Insurance Network

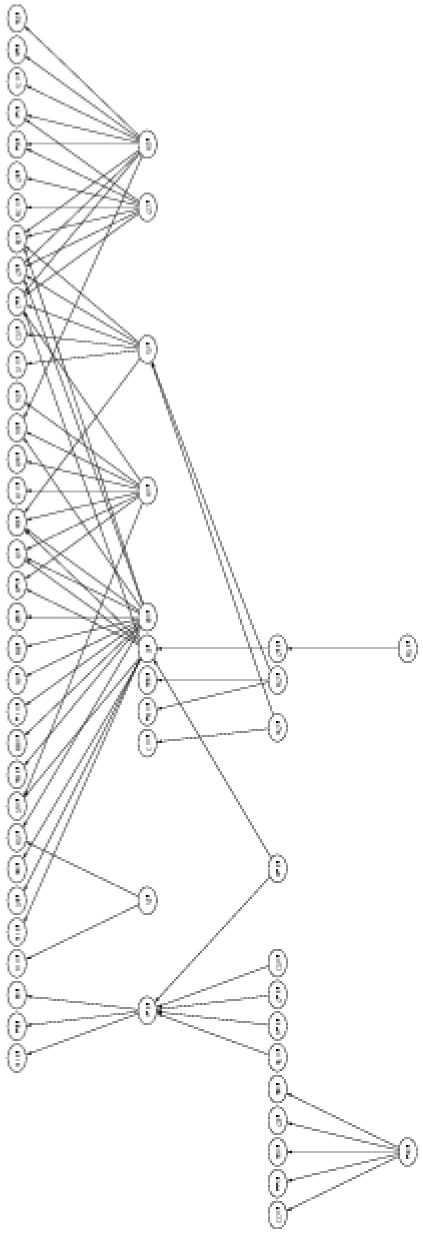

Figure 2: The *Carpo* network.

```
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("MaryCalls").var;
vars[1] = bn.nodes.getNode("JohnCalls").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 3: Querying the net