CS 188, Fall 1997, Introduction to Artificial Intelligence Assignment 4, due Oct 24 and 29, total value 8% of grade

Programming assignment, to be done in pairs.
IF POSSIBLE, FIND A DIFFERENT PARTNER FOR THIS ASSIGNMENT

This project deals with representation and inference in probabilistic (belief) networks. You will use the code provided with AIMA to build and exercise a simple network. You will also write code to implement the logic sampling and likelihood weighting algorithms and measure their performance. This will give you a good feel for approximation algorithms in general and for stochastic simulation methods in particular.

Turn in Part 1 (Questions 1 and 2) by 10/24, and Part 2 (Questions 3 to 7) by 10/29.

Part 1: Building a simple network

The first thing you need to do is load the AIMA code in the usual way (see Assignment 0), including the `uncertainty` package. The AIMA code includes a general facility for defining and usingbelief nets. You should look in particular at the following files:

- `code/uncertainty/algorithms/belief-nets.lisp`
- `code/uncertainty/algorithms/enumerate.lisp`
- `code/uncertainty/algorithms/belief-net-io.lisp`

These files do not appear in the standard WWW AIMA code distribution, so make sure you use the versions from `~cs188/code` (or `~cs188/code-dev`).

A belief net can be constructed using the interactive function `create-belief-net`. For example, the following transcript shows the construction of a two-node network:

```
>> (setq ab (create-belief-net))
*****************Creating New Node*******************
What is the node's name (nil if none)?  a
Enter list of values  (true false)
*****************Creating New Node*******************
What is the node's name (nil if none)?  b
Enter list of values  (true false)
*****************Creating New Node*******************
What is the node's name (nil if none)?  nil
************Creating belief net arcs*****************
Enter list of parent node names for B  (a)
Enter list of parent node names for A  nil
*******Creating distribution for node A*******
   Enter probabilities for A = (TRUE FALSE)
   (0.6 0.4)
```

```
*******Creating distribution for node B*******
 Given A = TRUE
     Enter probabilities for B = (TRUE FALSE)
     (0.9 0.1)
 Given A = FALSE
     Enter probabilities for B = (TRUE FALSE)
     (0.1 0.9)

#S(BN :NODES ... etc.
```

The interactive entry process is somewhat error-prone, so you may choose to use some of the functions called by `create-belief-net` to create and save intermediate stages in the process.

Once you have a belief net, you can create evidence for it by creating a data structure called an *event*:

```
>> (setq e1 (create-event ab))
Name of node to set (nil if none)?  b
Value to set it to?  true
Name of node to set (nil if none)?  nil

#(NIL 0)
```

Notice that an event is actually a vector of integers (or NILs) representing the values of all the nodes. Once you have evidence, you can ask for the probability of a variable given the evidence. The simplest exact inference algorithm is `(enumerate-ask X E bn)`, which takes a variable name X and an event E and returns a distribution over X:

```
>> (enumerate-ask 'A e1 ab)
((TRUE . 0.9310344816955176d0) (FALSE . 0.06896551830448243d0))
```

A belief net for CS classes

Let *Pass* be a variable which has value *true* if a given student passes a given class, and *false* otherwise. Let us consider some of the factors that might be relevant to this question, as seen from the point of view of the professor. The observable variables are the *GPA* (*high*, *medium*, or *low*), whether or not the student has taken the *PreReqs* (*true* or *false*), and whether or not the student is *Asleep* in class (*true* or *false*). Some unobserved variables include whether the student is *Smart*, *Studious*, and pulls *AllNiter*, and the student's *WorkLoad*. (It is up to you to choose the possible values of these unobserved variables.)

Question 1 (20 pts). Begin by using the procedure in Chapter 15 to design the topology of the network. You will need to choose an ordering of the variables. Then use the function `create-belief-net` to create the belief net itself. Use the function `display-belief-net` to display it readably. Explain your choice of topology. (You may also want to use `save-bn` to save the network to a file.)

Question 2 (10 pts). Write a function `(enumerate-ask-all-cases X evidence-vars bn)` which takes a variable X, a list of variable names, and a belief net, and prints out a list of all possible events using evidence-vars together with the distribution for the query variable given each event. For example:

```
>> (enumerate-ask-all-cases 'A '(B) ab)

((B . TRUE)): A is ((TRUE . 0.9310344816955176d0)
                    (FALSE . 0.06896551830448243d0))
```

```
((B . FALSE)): A is ((TRUE . 0.1428571509666184 7d0)
                     (FALSE . 0.8571428490333816d0))
```

(Hint: you might want to look at the definition of `display-cpt` to see how to enumerate over possible cases recursively.) Use your function to find out the probability of *Pass* given all twelve possible cases for the observable variables in your network. Check to see that these values look reasonable. If not, you may want to alter some of your conditional probabilities. You can do this using `edit-cpt-row` for each row you want to change.

Part 2: Stochastic simulation algorithms

Begin by reading the brief descriptions of both logic sampling and likelihood weighting in Chapter 15 of AIMA. Then read the detailed analysis (postscript file) of the likelihood weighting (LW) algorithm. A description of the enumeration algorithm is also available.

Question 3 (20 pts). Implement logic sampling. Do this bottom-up, using the following steps:

- Write a function `(random-from-discrete d)` that takes a discrete probability distribution `d`, represented as a vector of probabilities, and returns an integer corresponding to a randomly selected value from the distribution. For example, if `d` is `#(0.9 0.1)`, then with probability 0.9 the function returns 0 (the index of the first value), and with probability 0.1 it returns 1.
- Now write a function `(sample-node bnode event)` which generates a sample value for `bnode` given the values of its parents as specified in `event`. For example:

```
>> (setq e2 (make-event '((a . true)) ab))
#(0 NIL)
>> (setq b (bnode-by-name 'b ab))
>> (sample-node b e2)
0
>> (sample-node b e2)
0
>> (sample-node b e2)
1  etc...
```

- Now write `(ls-ask Xname E bn)`, which should take an additional, optional argument specifying the number of iterations. The skeleton can be similar to that of `enumerate-ask` and `enumerate-infer`, but the body of `ls-infer` should of course use logic sampling.

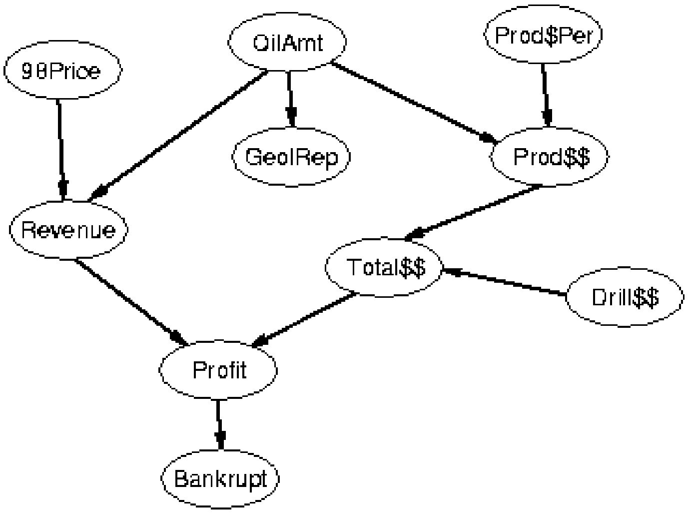
Now write `ls-ask-all-cases`, and show the results for your CS class net using 50 and 500 iterations.

Question 4 (20 pts). Write `lw-ask` to use likelihood weighting, as well as `lw-ask-all-cases`. Run the same tests. Comment on your results.

A belief net for oil drilling

We will now do some more thorough testing on a (slightly) more realistic example. (Using a single example also allows everyone to compare results.)

Imagine you are an oil industry analyst trying to predict whether LilOilCo will go bankrupt. Shown here is a belief network for assessing the financial situation of the company. LilOilCo has one prospect with an unknown amount of oil, *OilAmt*. There is a geologist's report *GeolRep* that estimates the amount of oil. Whether the company goes *Bankrupt* depends on the *Profit* it makes, which in turn depends on its *Revenue* and the *Total$$* it spends producing the oil. *Revenue* depends on the amount of oil and its price in 1998. Total cost depends on the cost to drill and the production cost; the latter depends on the amount of oil and the production cost per barrel. You may want to do `display-belief-net`.



Question 5 (5 pts). Load the oil network using

```
(setq oil (load-bn "~cs188/code/uncertainty/domains/oil.bn"))
```

Compute the exact probability of bankruptcy given that `GeolRep=low`. Save the results for later use.

Question 6 (10 pts). Suppose that in addition to the geologist's estimate, we can also look at a Wall Street Journal article assessing the current (1997) progress of the Middle East peace talks. This might be useful because if there is a war in 1998 that will dramatically affect the 1998 oil price. To take this into account, we will need three new variables: `WSJRep` (which can be `good` or `bad`), `97Talks` -- the *actual* state of the talks (which can be `good` or `bad`), `98War` (which can be `true` or `false`). Explain where these nodes would go in the network, what arcs you would need, and what new and modified CPTs are needed (propose some reasonable values for these).

Question 7 (15 pts). Now `ls-ask` and `lw-ask` must be ‘‘instrumented’’ so that we can measure the error with respect to the true probability distribution as a function of the number of samples. Write modified versions called

`(recording-ls-ask Xname E bn iterations interval true-distribution file)` and
`(recording-lw-ask Xname E bn iterations interval true-distribution file).`

The idea is that after every `interval` iterations (but not after 0 iterations) the current distribution for X is compared to the true distribution. Accumulate an association list, each element of which is a cons pair of the number of iterations and the squared error (see `belief-nets.lisp` for `squared-error`). At the end, call `plot-alist` to write the data to a file. Do as many iterations as you can reasonably manage. Use your functions to plot the squared error for `Bankrupt` given `GeolRep=low`, writing the output to `ls-error.data` and `lw-error.data` respectively. These files will be in a format suitable to be processed by gnuplot. There is a gnuplot command file called `error.gnuplot`. Copy this to your directory, and run it using the Unix command

```
UNIX PROMPT>> /usr/sww/bin/gnuplot < error.gnuplot
```

The output should appear on the screen and then will be written as `error.ps`, which should be included in your turnin after you have checked to make sure it looks right.

Question 8 (20 pts extra credit). Investigate the relative performance of LS and LW as you 1) increase the number of evidence variables, and 2) have evidence on nodes earlier or later in the node ordering.