CSC 261 **Lab 9: Bayesian Network Inference** **Fall 2011**

Assigned: Tuesday 1 November 2011

Due: Monday 7 November 2011, 11:59 pm

Objective: Understand bayesian network construction and message-passing based inference

Collaboration: This homework assignment will be completed in pairs assigned by the instructor

Introduction

We have studied how representing a probability distribution over many variables may be made very compact with the use of a Bayesian network. Now you will complete an implementation of the inference algorithm for Bayesian networks and use it to verify some probability calculations.

Code

You can copy the starter code for this assignment from the MathLan:
`/home/weinman/courses/CSC261/code/bayesnet`

Bayesian networks

The file `bayesnet.scm` contains procedures for creating a Bayesian network and accessing various aspects of the network, including the parents and children of a node, how many values a variable can take on, as well as the CPT for a particular variable. There is an example of using the Bayesian network data type to create a simple network in `example.scm`. More details on this file are given below.

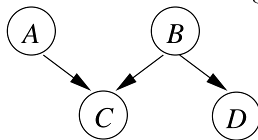
Belief propagation

The file `belief-propagation.scm` contains documentation for three of the five procedures for implementing a complete inference package for the Bayesian network data type. The procedure `compute-belief` makes the calculation of $\mathbf{P}(X \mid \mathbf{e})$, while `mesg-from-children` computes $\boldsymbol{\lambda}_X\left(X\right)$ and `mesg-to-child` computes $\boldsymbol{\pi}_{U\to X}\left(U\right)$. You will implement these three methods, while two others are implemented for you. The completed procedure `message-from-parents` computes $\boldsymbol{\pi}_X\left(X\right)$, while `message-to-parent` computes $\boldsymbol{\lambda}_{Y\to X}(X)$.

Example

The variables in our networks are simply indexed by integers, starting at zero. Thus, while we often wish to refer to them by name, in Scheme they will simply have numbers. Similarly, while the values are often meaningful (True/False, Red/Green/Orange/Brown), these too will be ordered and indexed by integers.

Consider the following simple network, where all nodes correspond to binary variables.



In this network, we give the nodes the following numbering:

$$\begin{aligned} A &= 0 \\ B &= 1 \\ C &= 2 \\ D &= 3. \end{aligned}$$

Creating a network

Conditional Probabilities

There is an example of using the Bayesian network data type to create the example network above in `example.scm`. The method for specifying the CPTs for each node is unlikely to be immediately intuitive, so we detail it here.

Lists are used to store probability table entries. The first entry in the list would correspond to the CPT entry for the first value of the variable, the second entry for the second value, etc. For example, the A node can take on two values, so this CPT is stored as a simple list

```
(define a-cpt (list 1/10 9/10))
```

where the first entry is for $P(\neg a)$ and the second entry is for $P(a)$. Thus, taking `(car a-cpt)` gives $P(\neg a)$.

When a node has one parent, as node D does, the table `d-cpt` is stored as a list of lists. The first entry `(car d-cpt)` still corresponds to the portion of the CPT for the first value . However, the entry is no longer a number, because the value of the probability $\mathbf{P}(\neg d \mid B)$ depends on the value of the parent node, B. Thus, the entry is also a list. The first entry of this inner list, `(car (car d-cpt))` now corresponds to the portion of the CPT for the first value of the *parent* node, $P(\neg d \mid \neg b)$. The second entry `(cadr (car d-cpt)` corresponds to the second value of the parent node, $P(d \mid b)$etc. In order to specify the probability for $D = d$ with both values of B we need a list

```
(list 18/100 73/100)
```

which gives the values for $P(\neg d \mid \neg b) = 0.18$ and $P(\neg d \mid b) = 0.73$.

What about the probabilities for $P(d \mid \neg b)$ and $P(d \mid b)$? These are not strictly necessary, because we know that a probability distribution must sum to one. However, in order to make the implementation of our algorithms more straightforward, we explicitly encode these numbers, i.e.,

```
(list 82/100 27/100)
```

which gives the values for $P(d \mid \neg b) = 0.82$ and $P(d \mid b) = 0.27$. To represent the entire CPT for D given B, we simply put these two lists together, with the first entry being for the case $D = \neg d$,and the second entry for $D = d$

```
(define d-cpt (list (list 18/100 73/100)
                    (list 82/100 27/100)))
```

Thus, taking `(car d-cpt)` yields $\mathbf{P}(\neg d \mid B)$—both values of B are represented for the case when d is false.

When a node has more parents, we simply repeat the nesting of lists to accommodate the additional values that must be accounted for by more nodes. For instance, C has two parents so we need three nested lists to account for the values of the two parents plus the possible values for C.

```
(define c-cpt (list (list (list 42/100 29/100)
                          (list 92/100 5/1000))
                    (list (list 58/100 71/100)
                          (list 8/100 995/1000))))
```

Thus, taking `(car c-cpt` gives $\mathbf{P}(\neg c \mid A, B)$–all four possibilities for A and B are represented for the case when c is false. Furthermore, taking `(caar c-cpt)` gives $\mathbf{P}(\neg c \mid \neg a, B)$), leaving the two possibilities for B. We could also take `(caaar c-cpt)`to find $P(\neg c \mid \neg a, \neg b)$.

Defining the topology

As mentioned above, the nodes are each given a number. In order to specify the topology of the network, we simply indicate the parents of each node. This is done as a list (one entry for each node) of lists (containing the parents of the node). The nodes A and B have no parents, so these lists are null, while C has two parents and D has B as its parent. This is represented with the following list.

```
(define example-topology (list null null (list 0 1) (list 2)))
```

Defining the network

The last element we need is to explicitly specify the "arity" of each of the nodes. That is, how many values they may take on. While this is implicitly found in the conditional probability tables, we prefer an explicit representation. Each of the nodes in the simple network are binary, so we specify this with a list of 2s.

```
(define example-node-sizes (list 2 2 2 2))
```

We can finally construct a network by passing in the node sizes, topology, a list of all the CPTs given in the same order as the node numbering

```
(define example-network
  (make-bayes-net example-node-sizes
                  example-topology
                  (list a-cpt
                        b-cpt
                        c-cpt
                        d-cpt)))
```

Performing inference

Specifying evidence

Adding evidence to our query means we have fixed specific values for certain nodes. Scheme allows us to easily represent this with an association list. Every key in the association list is a particular node number (index), and the value associated with that is the index of the value the observed variable. For instance, if we observe d, we indicate this by saying variable 3 has value 1 (because 0 represents false, while 1 represents true. We can directly quote such an evidence list as follows

```
'((3 . 1))
```

If we wanted to incorporate the fact $\neg c$ into the list, we simply add another pair

```
(define evidence '((3 . 1) (2 . 0)))
```

This is equivalent to a list of cons cells,

```
(define evidence (list (cons 3 1) (cons 2 0)))
```

Note that order is not strictly important in an association list. If we had no evidence at all, our association list would simply be `null`.

How then can we query the association list? Scheme provides the procedure `(assoc` *`key alist`*`)`. If it returns `#f`, the `key` was not found. Otherwise, it returns the complete matching entry. For example

```
> (assoc 0 evidence)
#f
> (assoc 3 evidence)
(3 . 1)
```

Computing beliefs

The procedure `compute-belief` takes a network, a variable (index number), and a list of evidence, returning the marginal conditional probability for the query variable given the evidence. For example, to find $\mathbf{P}(E \mid \neg j, m)$,

```
> (compute-belief example-network 1 '((3 . 1)))
(119760/131297 11537/131297)
```

The first entry in the list corresponds to $P(\neg b \mid d)$ and the second entry is $P(b \mid d)$. The CPTs were entered as exact rational numbers, so we can verify the calculations are correct exactly. However, these ratios are not directly enlightening. If we map `exact->inexact` onto the list we find the values are

```
(0.9121305132638217 0.08786948673617828)
```

so the odds seem to be against b about ten to one.

Assignment

Your tasks are to complete the belief propagation implementation, as specified in `belief-propagation.scm`, build the earthquake network, and use it to perform some queries.

Problem 1: Earthquake!

Using the examples above as a guide, create the earthquake network detailed in AIMA Figure 14.2 using the following steps in a file `earthquake.scm`.

a. Detail (i.e., in a comment) the node ordering you have chosen.

b. Define CPTs for each node in Scheme; use meaningful names.

c. Define `earthquake-topology`, the parent-child relations for all nodes in the network.

d. Define `earthquake-node-sizes`, the cardinality of each variable in the network.

e. Define `earthquake-network` using `make-bayes-net` and the values defined above.

Problem 2: `mesg-from-children`

Implement the procedure `mesg-from-children` as specified by MESSAGE-FROM-CHILDREN in the belief propagation reading. Some important things to note:

- You may get a list of all children of a *node* for a `bayes-net` with the procedure `(bayes-net-children` *`bayes-net node`*`)` from `bayesnet.scm`.

  Hint: use `map` to calculate the messages from all the children to the node.

- The procedure `pointwise-product` is implemented in `general.scm`. This procedure takes a list of lists of numbers and returns a list whose first entry is the product of the first entries of all the lists, the second entry is the product of the second entries of all the lists, etc.

- `(normalize` *`numbers`*`)` is also implemented in `general.scm`. It takes a list of numbers and returns a new list that sums to one; in other words, it takes a list of relative magnitudes and transforms them into a probability distribution.

- The procedure `(ones` *`len`*`)` in `general.scm` creates a list of ones of the given length.

- You may find the "arity" of a variable in the network with the procedure `(bayes-net-node-size` *`bayes-net node`*`)` from `bayesnet.scm`.

- Finally, the procedure `(point-distribution` *`index len`*`)` found in `general.scm` creates a list of length *`len`* with a 1 at *`index`* and 0 at every other position.

Problem 3: `mesg-to-child`

Implement the (very similar) procedure `mesg-to-child`.

Some (additional) important things to note:

- The procedure `(delete num lst)` from `general.scm` takes a list of numbers *lst* and returns a new list with *num* removed.

  Hint: Use `map` to calculate the messages from all the (other) children of the node
- You may wish to handle separately the case that U only has no children aside from X.

Problem 4: `compute-belief`

Implement the procedure `compute-belief`.

Problem 5: Testing your work

Using enumeration or another method, write down the expression for calculating the following probabilities from the earthquake network, and give the resulting values:

Diagnostic: $P(e \mid j)$

Causal: $P(\neg m \mid \neg b)$

Both: $P(a \mid e, j)$

Finally, use your `compute-beliefs` procedure and your `earthquake-network` to verify agreement between your calculation and your code.

Problem 6: Querying your network

One often interesting question to ask is: how does the probability of an earthquake change as we incorporate more evidence? How does our belief that an earthquake occured change from when we know John hasn't called to when we find that Mary has called (and John has not)? Compute the ratio (typically called the odds) of these two probabilities. Comment on the numeric value and the interpretability of this ratio, especially as compared with the those of the raw probabilities themselves.

What to turn in

Your submission should include the following

- Your completed `earthquake.scm` and `belief-propagation.scm` files
- A short driver program that evalutes the queries of Problems 5 and 6
- A single PDF containing (merged)
  - Your Scheme files
  - A transcript of your test driver program's output
  - Your manual calculation of the probabilities in Problems 5
  - Your brief commentary on Problem 6

Copyright ©2011 Jerod Weinman. This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States License.