Homework Assignment 3

Due: February 23, 2017 at 11am

Total points: 100

Deliverables: `hw3.pdf` containing typeset solutions to Problems 1-3.
Source code containing your implementation for Problem 4.

Guidelines: All files must be submitted by Dropbox. You can brainstorm with others, but please solve the problems and write up the answers and code by yourself. You may use textbooks (Koller & Friedman, Russell & Norvig, etc.), lecture notes, and standard programming references (e.g., online Java API documentation). Please do NOT use any other resources or references (e.g., example code, online problem solutions, etc.) without asking.

1 Message Passing on a Tree

1. Consider the Bayesian network shown in Figure 1 which represents a fictitious biological model. Each G_i represents a genotype of a person: $G_i = 1$ if they have a healthy gene and $G_i = 2$ if they have an unhealthy gene. G_2 and G_3 may inherit the unhealthy gene from their parent G_1. $X_i \in \mathbb{R}$ is a continuous measure of blood pressure, which is low if you are healthy and high if you are unhealthy. We define the CPDs as follows

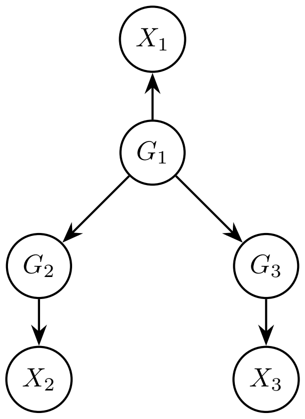

Figure 1

$$P(G_1) = [1/2, 1/2]$$
$$P(G_i = G_1 | G_1) = 0.9, \ i \in \{2, 3\}$$
$$P(G_i \neq G_1 | G_1) = 0.1, \ i \in \{2, 3\}$$
$$P(X_i | G_i = 1) = \mathcal{N}(X_i | \mu = 50, \sigma^2 = 10)$$
$$P(X_i | G_i = 2) = \mathcal{N}(X_i | \mu = 60, \sigma^2 = 10)$$

(a) Suppose that you observe $X_2 = 50$, and X_1 is unobserved. What is the posterior belief on G_1, i.e., $P(G_1 | X_2 = 50)$?

(b) Suppose you observe $X_2 = 50$ and $X_3 = 50$. What is $P(G_1 | X_2, X_3)$? Explain your answer intuitively.

(c) Suppose $X_2 = 60$ and $X_3 = 60$. What is $P(G_1 | X_2, X_3)$? Explain your answer intuitively.

(d) Suppose $X_2 = 50$ and $X_3 = 60$. What is $P(G_1 | X_2, X_3)$? Explain your answer intuitively.

2 Clique Trees

2. Assume that we have constructed a clique tree $\mathcal{T}$ for a given Bayesian network graph $\mathcal{G}$, and that each of the cliques in $\mathcal{T}$ contains at most k nodes. Now, the user decides to add a single edge to the Bayesian

network, resulting in a network $\mathcal{G}'$. (The edge can be added between any pair of nodes in the network, so log as it maintains acyclicity.) What is the tightest bound you can provide on the maximum clique size in a clique tree $\mathcal{T}'$ for $\mathcal{G}'$? Justify your response by explaining how to construct such a clique tree. (Note: You do not need to provide the optimal clique tree $\mathcal{T}'$. The question asks for the tightest clique tree that you can construct, using only the fact that $\mathcal{T}$ is a clique tree for $\mathbb{G}$.

3. Consider the problem of eliminating extraneous variables from a clique tree. More precisely, given a calibrated clique tree $\mathcal{T}$ over $\mathcal{X}$, we want to generate a (calibrated) clique tree $\mathcal{T}'$ whose scope is some subset $\boldsymbol{Y} \subset \mathcal{X}$. Clearly, we want to make $\mathcal{T}'$ as small as possible (in terms of the size of the resulting cliques). However, we do not want to construct and calibrate a clique tree from scratch; rather, we want to reuse our previous computation.

   (a) Suppose $\mathcal{T}$ consists of two cliques $\boldsymbol{C}_1$ and $\boldsymbol{C}_2$ over variables A, B, C and C, D respectively. What is the resulting $\mathcal{T}'$ if $\boldsymbol{Y} = \{B, C\}$?

   (b) For the clique tree $\mathcal{T}$ defined in part 1, what is $\mathcal{T}'$ if $\boldsymbol{Y} = \{B, D\}$?

   (c) Now consider an arbitrary clique tree $\mathcal{T}$ over $\mathcal{X}$ and an arbitrary $\boldsymbol{Y} \subseteq \mathcal{X}$. Provide an algorithm to transform a calibrated tree $\mathcal{T}$ into a calibrated tree $\mathcal{T}'$ over $\boldsymbol{Y}$. Your algorithm should *not* resort to manipulating the underlying network or factors; all operations should be performed directly on the clique tree.

   (d) Give an example where the resulting tree $\mathcal{T}'$ is larger than the original clique tree $\mathcal{T}$.

3 Programming: Belief Propagation

4. The bulk of the assignment is implementing belief propagation. You may write your implementation of belief propagation in C, C++, Java, Python or MATLAB.

   Your implementation must meet the following input/output specifications in order to keep grading tractable:

   - You must include two scripts along with your code: one to compile the code on tricycle (if necessary), and one to run the code on tricycle once it has been compiled.
   - The compile script should run in a directory consisting of just your submitted files (including any additional libraries your code needs to compile/run).
   - The run script must accept one command-line argument, the name of the Bayesian network in .bif (Bayesian Interchange Format) file, and must write all marginal probabilities to the file "result.txt" in the current directory. (You are free to print any diagnostic output you wish to the standard output or other files).
   - The "result.txt" file produced must contain a list of variables and their marginal distributions. Variables must appear in the order in which they were introduced in the .bif file. Example:

```
A 0.9 0.1
B 0.72 0.28
C 0.001 0.999
D 1.0 0
```

   The belief propagation algorithm you implement shoudl follow the description given in class, and also described here: http://www.comm.utoronto.ca/~frank/papers/KFL01.pdf.

   NOTE: In order to have comparable answer, please use a simple message passing schedule in which all variable messages are sent and then all factor messages are sent in each iteration. (Mixing up the order of everything can lead to interesting, but different, results).

In order to make this assignment less onerous, we provide a skeleton in C++ that handles some of the input and output. You are free to use any portions of that skeleton that you find helpful, or discard it entirely and come up with something else. We also provide several example input/output pairs to help you check your implementation.

Use your implementation of belief propagation as you answer the following questions.

(a) *Briefly* describe how you implemented belief propagation (e.g., data structures, code organization, etc.).

(b) Run you algorithm on the Sprinkler network (sprinkler.bif). What is the marginal probability of WetGrass according to belief propagation? What is the true marginal probability of WetGrass (you can easily compute this by hand)? Explain why these number are different.

(c) Suppose X is a variable with k states and f neighboring factors. Explain how to compute all outgoing belief propagation messages with complexity $\mathcal{O}(kf)$. Does your method work even when the incoming message have zeros? (NOTE: You are free to use a less efficient method in your actual implementation).

Appendix

We recommend you use the VFML library, which provides routines for reading .bif files, and provides data structures and functions for traversing the graph, retrieving values from conditional probability tables, etc. The documentation for the library may be found at `http://www.cs.washington.edu/dm/vfml/`. The "Getting Started" link describes how to download the library. It is also included in the `hw3-dist.zip` download with the skeleton code.

Here are some suggestions if you do end up using VFML. Documentation for the Bayesian network routines is found by following Modules → "Belief Net Section" → BeliefNet.h. The **ExamplePtr** class is used for holding values of the nodes. Spend a little time understanding Examples. You can create an **ExamplePtr** by using *BNGetExampleSpec* to get an **ExampleSpecPtr**, which you give as a parameter to **ExampleNew**. Probably the best routine for retrieving CPT values from the nodes will be *BNNodeGetCP*, which takes an **ExamplePtr** that defines the value of the parents of the node and the state of the node that you want to ask the probability of. Note that the ID of a node (from *BNNodeGetID*) is the same as the attribute number in the Example (for instance, when using *ExampleSetDiscreteAttributeValue*). Similarly, to find out the number of nodes in the network and names of the nodes, etc., you use the **ExampleSpecPtr** associated with the network, and routines like *ExampleSpecGetNumAttributes* and *ExampleSpecLookupAttributeName*.

It could also be useful to look at the code for *BNLikelihoodSampleNTimes* (in BeliefNet.c) to get an idea about traversing the graph, retrieving CPT values, etc.