Homework 3

Due at noon on February 24, 2016

GUIDELINES: You can brainstorm with others, but please solve the problems and write up the answers by yourself. You may use textbooks (Koller & Friedman, Russel & 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.

SUBMISSION INSTRUCTIONS: Submit this assignment by Dropbox. You submission should include: A PDF containing written answers and all your code.

1 FUNDAMENTALS [25 POINTS]

Consider the Bayesian network shown in Figure 1, which illustrates the influence of genetics and environment on an individual's overall health. Specifically, this model encodes a set of independence relationships among the following variables: Genetics (G), Environment (E), Health (H), Disease1 (D_1), Disease2 (D_2), Disease3 (D_3), Symptom1 (S_1), Symptom2 (S_2). Use this model to answer the questions in Parts 1 and 2 below.

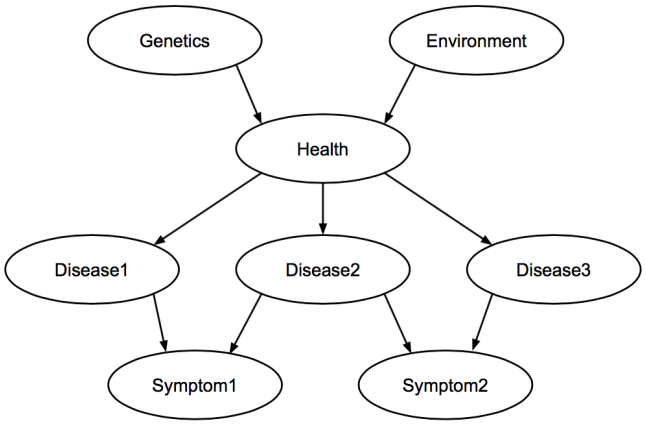

Figure 1: A Bayesian network that represents a joint distribution over the variables Genetics, Environment, Health, Disease1, Disease2, Disease3, Symptom1, and Symptom2.

Part 1: Variable Elimination in Bayesian Networks [13 points]

1. Write down an expression for $P(S_1)$ in terms of the un-factorized joint distribution.
2. Starting with the expression you gave in the last question, write down an expression for $P(S_1)$ in terms of the factorized joint distribution and simplify it by pushing summations into the product of terms whenever possible.
3. What variable elimination ordering does the resulting expression correspond to?
4. Walk through an execution of the variable elimination algorithm for evaluating $P(S_1)$ using the ordering you specified in the last question. Start by writing out the initial set of factors and drawing the moralized graph. Then go through each step of the elimination algorithm and (1) specify the intermediate factors that are generated by the product and sum operations, in that order, (2) write down the new set of factors that remain after the entire step has been executed, and (3) draw the new graph induced by eliminating the variable. For clarity, please use $\phi(\cdot)$ to denote initial factors, $\psi(\cdot)$ to denote intermediate factors generated by a product operation, and $\tau(\cdot)$ to denote intermediate factors generated by a sum operation.
5. Assuming each variable can take m values, what is the size of the largest intermediate factor generated during this execution of the variable elimination algorithm, i.e. how many entries does this factor have?
6. What is the computational complexity (in terms of the number of possible assignments, m, and the number of variables in the network, n) of this execution of variable elimination?
7. Can you do better than this? Specifically, is there another elimination ordering that yields a lower-complexity algorithm? If so, give the best such ordering and evaluate its computational complexity.

Part 2: Variable Elimination with Clique Trees [6 points]

A *clique tree*, also known as a *junction tree*, is a very useful data structure that serves as a "graphical flowchart" of the factor manipulation process for the variable elimination algorithm (and, as we will later see, for the message passing algorithm). We define a clique tree $\mathcal{T}$ for a set of factors Φ over variables $\mathcal{X}$ as an undirected graph whose notes i are each associated with a cluster $\boldsymbol{C}_i \subseteq \mathcal{X}$ and whose edges ij are each associated with a sepset $\boldsymbol{S}_{ij} = \boldsymbol{C}_i \cap \boldsymbol{C}_j$. Clique trees must satisfy two important properties: *family preservation* and *running intersection*. Before doing this question, please carefully read Section 10.1 of Koller and Friedman so that you obtain a good understanding of what a clique tree is and how it plays a role in variable elimination.

1. Draw the best clique tree for the Bayesian network of Figure 1, i.e. the one induced by the variable elimination ordering with the lowest computational complexity. What is the size of the largest intermediate factor generated in the corresponding execution of variable elimination? What is the computational complexity of this variable elimination run?
2. Draw the worst clique tree for the Bayesian network of Figure 1, i.e. the one induced by the variable elimination ordering with the highest computational complexity. What is the size of the largest intermediate factor generated in the corresponding execution of variable elimination? What is the computational complexity of this variable elimination run?

Part 3: From Variable Elimination to Message Passing [6 points]

Before doing this question, please review the lecture notes and/or read Section 10.2 of Koller and Friedman, paying careful attention to Section 10.2.2, so that you understand how to perform sum-product message passing using clique trees. Note that the sum-product algorithm described in the textbook is equivalent to the junction tree algorithm with Shafer-Shenoy updates introduced in class. From now on, we will simply refer to this as sum-product message passing.

Suppose you design a clique tree $\mathcal{T}$ for the Bayesian network of Figure 1 and run sum-product message passing to calibrate $\mathcal{T}$. Denote the calibrated beliefs over each cluster as $\beta_i(\boldsymbol{C}_i)$ and the calibrated beliefs over each sepset as $\mu_{ij}(\boldsymbol{S}_{ij})$.

1. What exactly is a "message" in the sum-product message passing algorithm?
2. Assume there are N cliques in $\mathcal{T}$. How many messages must be sent in order to fully calibrate the clique tree?
3. Give an expression for the full joint distribution in terms of the cluster and/or sepset beliefs.
4. Given an expression for $P(S_1)$ in terms of the cluster and/or sepset beliefs.

2 BELIEF PROPAGATION [75 POINTS]

The bulk of this 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 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 should 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 answers, 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 your 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 numbers 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 $O(kf)$. Does your method work even when the incoming messages 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.