CSE 515: Statistical Methods in Computer Science
Homework #3

Due at noon on February 25th

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.) and lecture notes from class. 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 email to Chloé Kiddon (chloe@cs). Attachments should include: A PDF containing written answers and your complete implementation for the programming portion. Typed answers are highly preferred, but if this is a hardship, then handwritten answers are fine as long as they are completely legible.

1. Let $\mathcal{B}$ be a Bayesian network over a set of n random variables $\mathcal{X}$ and $\mathcal{F}$ be the set of factors corresponding to the CPDs of $\mathcal{B}$. Suppose that we then construct a clique tree $\mathcal{T}$ by choosing a variable elimination ordering $\prec$ and finding the maximal cliques in the induced graph $\mathcal{I}_{\mathcal{F},\prec}$. Our task is to optimize the message passing that takes place as we calibrate $\mathcal{T}$. For simplicity you may assume $|Val(X)| = d$ for all $X \in \mathcal{X}$ for the following question.

   Recall that the message passing operation works by multiplying the incoming messages (except for one) with the product of the factors assigned to the clique, and then eliminating all the variables not in the scope of the outgoing message. For example, in the figure below, suppose the pictured clique (called $\mathbf{C}_2$) contains initial factors $\phi_1(A, B)$, $\phi_2(B, C)$, and $\phi_3(C, D)$ and it receives message $\delta_{1\to2}(A, D)$ from $\mathbf{C_1}$. Then an outgoing message $\delta_{2\to3}(B, D)$ is generated by multiplying all initial factors and incoming messages, and then summing out over all variables except B and D.

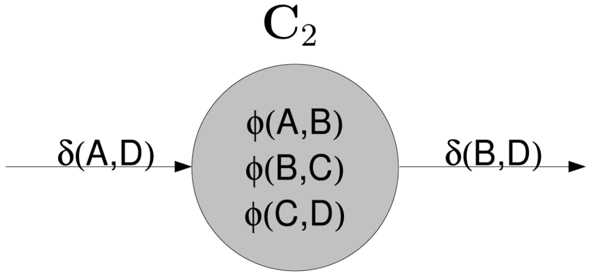


Figure 1: Message Passing Example

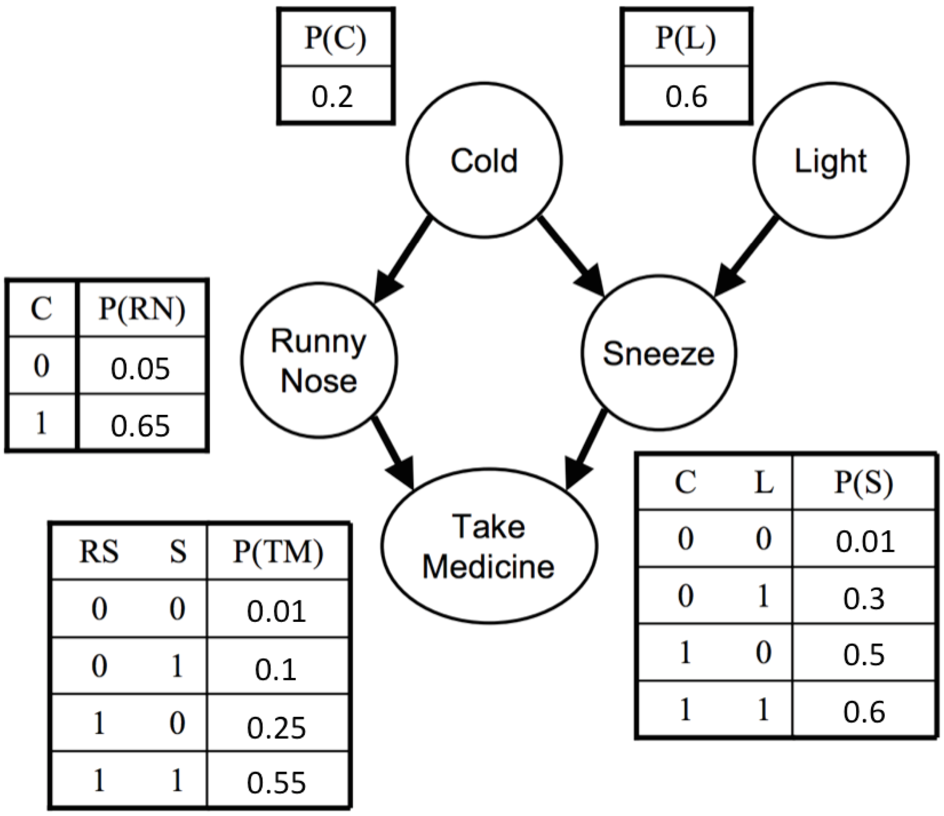
Now we wish to improve upon the efficiency of our message passing, but at the cost of the exactness of our algorithm. We would like to force the incoming message into the following factored form: $\tilde{\delta}_{1\to 2}(A, D) = \tilde{\delta}^1_{1\to 2}(A)\tilde{\delta}^2_{1\to 2}(D)$. Similarly, we would like to force the outgoing message into the following form: $\tilde{\delta}_{2\to 3}(B, D) = \tilde{\delta}^1_{2\to 3}(B)\tilde{\delta}^2_{2\to 3}(D)$.

(a) Show how the outgoing message $\tilde{\delta}_{2\to 3}(B, D)$ can be computed more efficiently using this representation. Specifically, you should show precisely how the factored messages $\tilde{\delta}^1_{1\to 2}(A)$ and $\tilde{\delta}^2_{1\to 2}(D)$ are used along with the initial factors in $\mathbf{C}_2$ to compute $\tilde{\delta}_{2\to 3}(B, D)$. Also show how this is asymptotically more efficient than standard message passing.

(b) Briefly but precisely describe the algorithm underlying this example (no need for pseudo-code), assuming that incoming messages may be forced into the following factored form $\tilde{\delta}_{i\to j}(\mathbf{S}_{i,j}) = \prod_k \tilde{\delta}^k_{i\to j}(\mathbf{A}^k_{i,j})$, where $\mathbf{A}^k_{i,j}$ is a *disjoint* partition of $\mathbf{S}_{i,j}$. Describe how factored messages are computed and how they are used to compute outgoing messages. Briefly explain why the factored messages are not exact.

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, as long 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. (Node: 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 $\mathcal{G}$. Hint: Construct an example.)

3. Apply the variable elimination algorithm to the network below to compute the probability of Cold given Sneeze and TakeMedicine (i.e., P(C = true | S = true, T = true)). (Please show the order in which you eliminate variables and some sample intermediate calculations, but you don't need to include every single number from every step.)



4. 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.)

Programming Project: Belief Propagation

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.

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.