Graphical Models: Assignment 2

Antonio Remiro, I-Horng Huang, Tomas Jakab, Jesús Herrera, Nicholas Williams

21 November 2016

Contributions

jesus.herrera.16@ucl.ac.uk: 5.10, 5.13
antonio.remiro.16@ucl.ac.uk: 4.15, 6,12, 6.7
nicholas.williams.16@ucl.ac.uk: 5.11, 5.12
tomas.jakab.16@ucl.ac.uk: 5.8, 6.9, 6.10
i-horng.huang.16@ucl.ac.uk: 5.9, 6.7

Exercise 4.15

(1.) We assume that the values of the variables are enumerated in the following manner: $y_1 = q_{12}(x_1 = 0, x_2 = 0)$, $y_2 = q_{12}(x_1 = 0, x_2 = 1)$, $y_3 = q_{12}(x_1 = 1, x_2 = 0)$, $y_4 = q_{12}(x_1 = 1, x_2 = 1)$, $y_5 = q_{13}(x_1 = 0, x_3 = 0)$, $y_6 = q_{13}(x_1 = 0, x_3 = 1)$, ... , $y_9 = q_{23}(x_2 = 0, x_3 = 0)$, ..., $y_{12} = q_{23}(x_2 = 1, x_3 = 1)$.

To represent these variables in a linear system $\mathbf{My} = \mathbf{c}$, where $\mathbf{y} = \begin{pmatrix} y_1 \\ \vdots \\ y_{12} \end{pmatrix}$, and $\mathbf{M}$ is a 9×12 matrix, 9 constraints must be imposed on $\mathbf{y}$ (and on the q distributions). Most obviously, 3 constraints - one for each distribution - correspond to enforcing the mutual exclusivity within a distribution (each q must sum to 1):

$$\sum_{x_i, x_j = 0,1} q_{ij}(x_i, x_j) = 1 \rightarrow \begin{cases} y_1 + y_2 + y_3 + y_4 = 1 \\ y_5 + y_6 + y_7 + y_8 = 1 \\ y_9 + y_{10} + y_{11} + y_{12} = 1. \end{cases}$$

Slightly subtler are the other 6 constraints, which correspond to enforcing cross-distribution consistency. If two distributions share a variable assignment, we need to ensure that such is consistent in those two distributions. This is equivalent to enforcing "marginal consistency" in the question: We have:

$$\sum_{x_2} q_{12}(x_1, x_2) = \sum_{x_3} q_{13}(x_1, x_3) \rightarrow \begin{cases} y_1 + y_2 - y_5 - y_6 = 0 \\ y_3 + y_4 - y_7 - y_8 = 0, \end{cases}$$

where the first constraint ensures consistency when $x_1 = 0$ and the second when $x_1 = 1$. Similarly,

$$\sum_{x_1} q_{12}(x_1, x_2) = \sum_{x_3} q_{23}(x_2, x_3) \rightarrow \begin{cases} y_1 + y_3 - y_9 - y_{10} = 0 \\ y_2 + y_4 - y_{11} - y_{12} = 0, \end{cases}$$

where the first constraint ensures consistency when $x_2 = 0$ and the second when $x_2 = 1$, and:

$$\sum_{x_1} q_{13}(x_1, x_3) = \sum_{x_2} q_{23}(x_2, x_3) \rightarrow \begin{cases} y_5 + y_7 - y_9 - y_{11} = 0 \\ y_6 + y_8 - y_{10} - y_{12} = 0, \end{cases}$$

where the first constraint ensures consistency when $x_3 = 0$ and the second when $x_3 = 1$. We now have a linear system with:

$$\mathbf{M} = \begin{pmatrix} 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & -1 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & -1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & -1 & 0 & -1 \end{pmatrix}, \mathbf{c} = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}.$$

Sets of marginally consistent distributions can be found by inspecting the simultaneous equations. For example in constraint 1, setting $y_1 + y_2 = 1 \to y_3 + y_4 = 0$ since $y_3, y_4 \geq 0$, which implies $y_7 + y_8 = 0$ in constraint 5. Similarly $y_7 + y_8 = 0 \to y_5 + y_6 = 1$ in constraint 2, $y_3 + y_4 = 0 \to y_3 = 0 \to y_1 = y_9 + y_{10}$ in constraint 6 and $y_3 + y_4 = 0 \to y_4 = 0 \to y_2 = y_{11} + y_{12}$ in constraint 7. Also, $y_5 = y_9 + y_{11}$ in constraint 8 and $y_6 = y_{10} + y_{12}$ in constraint 9. For these simplified constraints, it is simple to obtain a solution to the linear system e.g. $\mathbf{y} = (0.75, 0.25, 0, 0, 0.7, 0.3, 0, 0, 0.5, 0.25, 0.2, 0.05)^T$, which corresponds to $q_{12}(x_1 = 0, x_2 = 0) = 0.75$, $q_{12}(x_1 = 0, x_2 = 1) = 0.25$, $q_{13}(x_1 = 0, x_3 = 0) = 0.7$, $q_{13}(x_1 = 0, x_3 = 1) = 0.3$, $q_{23}(x_2 = 0, x_3 = 0) = 0.5$, $q_{23}(x_2 = 0, x_3 = 1) = 0.25$, $q_{23}(x_2 = 1, x_3 = 0) = 0.2$, $q_{23}(x_2 = 1, x_3 = 1) = 0.05$ and all other instances of the distributions set to zero. Other sets of distributions can be found setting $y_3 + y_4 = 1$, for example.

(2.) We write the 8 states of p as $z_1 = p(x_1 = 0, x_2 = 0, x_3 = 0)$, $z_2 = p(x_1 = 0, x_2 = 0, x_3 = 1)$, $z_3 = p(x_1 = 0, x_2 = 1, x_3 = 0)$, $z_4 = p(x_1 = 0, x_2 = 1, x_3 = 1)$,$z_5 = p(x_1 = 1, x_2 = 0, x_3 = 0)$, $z_6 = p(x_1 = 1, x_2 = 0, x_3 = 1)$, $z_7 = p(x_1 = 1, x_2 = 1, x_3 = 0)$, $z_8 = p(x_1 = 1, x_2 = 1, x_3 = 1)$.

To represent these variables in a linear system $\mathbf{Az} = \mathbf{y}$, where $\mathbf{z} = \begin{pmatrix} z_1 \\ \vdots \\ z_8 \end{pmatrix}$, and $\mathbf{M}$ is a 12×8 matrix, 12 constraints must be imposed on $\mathbf{z}$. Mutual exclusivity within distribution p is already imposed in the question, where $\sum_i z_i = 1$. The 12 constraints correspond to enforcing cross-distribution consistency. If p and q_{ij} share a variable assignment, we need to ensure that such is consistent in those two distributions (enforcing marginal consistency). We have:

$$\sum_{x_3} p(x_1, x_2, x_3) = q_{12}(x_1, x_2) \to \begin{cases} z_1 + z_2 = y_1 \\ z_3 + z_4 = y_2 \\ z_5 + z_6 = y_3 \\ z_7 + z_8 = y_4, \end{cases}$$

where the first constraint ensures consistency when $x_1 = 0, x_2 = 0$, the second when $x_1 = 0, x_2 = 1$, the third when $x_1 = 1, x_2 = 0$ and the fourth when $x_1 = 1, x_2 = 1$. Similarly,

$$\sum_{x_2} p(x_1, x_2, x_3) = q_{13}(x_1, x_3) \to \begin{cases} z_1 + z_3 = y_5 \\ z_2 + z_4 = y_6 \\ z_5 + z_7 = y_7 \\ z_6 + z_8 = y_8, \end{cases}$$

where the first constraint ensures consistency when $x_1 = 0, x_3 = 0$, the second when $x_1 = 0, x_3 = 1$, the third when $x_1 = 1, x_3 = 0$ and the fourth when $x_1 = 1, x_3 = 1$. Also:

$$\sum_{x_1} p(x_1, x_2, x_3) = q_{23}(x_2, x_3) \to \begin{cases} z_1 + z_5 = y_9 \\ z_2 + z_6 = y_{10} \\ z_3 + z_7 = y_{11} \\ z_4 + z_8 = y_{12}. \end{cases}$$

where the first constraint ensures consistency when $x_2 = 0, x_3 = 0$, the second when $x_2 = 0, x_3 = 1$, the third when $x_2 = 1, x_3 = 0$ and the fourth when $x_2 = 1, x_3 = 1$. We now have a linear system:

$$\mathbf{A} = \begin{pmatrix} 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \end{pmatrix}, \mathbf{y} = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \\ y_5 \\ y_6 \\ y_7 \\ y_8 \\ y_9 \\ y_{10} \\ y_{11} \\ y_{12} \end{pmatrix}.$$

Proof by counterexample: Lets introduce a marginally consistent $\mathbf{y}$ into the linear system. Consider a possible solution to (1.) $\mathbf{y} = (0.5, 0, 0, 0.5, 0, 0.5, 0.5, 0, 0.25, 0.25, 0.25, 0.25)^T$. $\mathbf{A}$ and $\mathbf{y}$ can be written as an augmented matrix $(\mathbf{A}|\mathbf{y})$. This system can now be simplified using the MATLAB function `rref`; which performs row reduction on the matrix to obtain a simpler one. We obtain the row reduced echelon form of the augmented matrix,

$$(\mathbf{A}|\mathbf{y}) = \begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & | & 0.25 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -1 & | & 0.25 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & -1 & | & -0.25 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & | & 0.25 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & -1 & | & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 & | & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & | & 0.5 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & | & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & | & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & | & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & | & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & | & 0 \end{pmatrix}.$$

Note how this matrix gives $y_5 - y_8 = 0$ in its fifth row. This implies $y_5 = y_8 = 0$, since both values represent probabilities and cannot take a negative value. If this were the case, we would have $y_3 - y_8 = -0.25 \to y_3 = -0.25$ in the third row, a negative probability. This counterexample is sufficient to prove that it is not always possible to find a distribution $\mathbf{z}$ which gives rise to marginally consistent $\mathbf{y}$.

Exercise 5.8

We use code available on the website `http://web4.cs.ucl.ac.uk/staff/D.Barber/pmwiki/pmwiki.php?n=Main.Software` in this solution. This was explicitly allowed as stated in the forum post `https://moodle.ucl.ac.uk/mod/forum/discuss.php?d=725099#p1225101`.

The problem can be formalized as a Hidden Markov Model:

$$p(v_{1:T}, h_{1:T}) = p(h_1)p(v_1|h_1)\prod_{t=2}^{T} p(v_t|h_t)p(h_t|h_{t-1}).$$

The hidden variables $h_{1:T}$ represent original string. The variables $v_{1:T}$ represent observed corrupted string.

We use 4 general hidden states:

(1) before-firstname

(2) firstname

(3) before-surname

(4) surname

The general states (2) and (4) consist of 5 respectively 7 pattern states that correspond to the names:

(2.1) david

(2.2) anton

(2.3) fred

(2.4) jim

(2.5) barry

(4.1) barber

(4.2) ilsung

(4.3) fox

(4.4) chain

(4.5) fitzwilliam

(4.6) quinceadams

(4.7) grafvonunterhosen

In addition, each of the pattern states consist of letter states that correspond to letters in a particular name. For example pattern state (2.1) consist of the following letter states:

(2.1.1) d

(2.1.2) a

(2.1.3) v

(2.1.4) i

(2.1.5) d

We define the transition probabilities $p(h_t|h_{t-1})$ for hidden states using transition diagrams. The figure 1 shows transition diagram for general and pattern hidden states. The figure 2 illustrates transition diagram for letter states of the (2.1) pattern state. Transition diagrams for letter states of other pattern states (2.*) and (4.*) are analogous. The prior probability $p(h_t)$ of the state (1) is 1. In the code, the transition probabilities for general states and pattern states are computed in the `tran` matrix. The final transitional matrix `phghm` is then computed by `patternsearchsetup` function.

The emission probability $p(v_t|h_t)$ is uniform over all possible letters for general states (1) and (3). This is specified in the variable `pvgh_general`. For the letter states, the emission probability is $p(v_t = u|h_t = s) = 0.7/25$ iff $u \neq s$ and $p(v_t = u|h_t = s) = 0.3$ iff $u = s$. This is specified in the `CorruptionProb` matrix. The final emission matrix `pvgh` is also computed by `patternsearchsetup` function.

Given the observed sequence, prior over states, transition matrix `phghm`, and the emission matrix `pvgh`, we can compute the most probable explanation, i.e.:

$$h^*_{1:T} = arg \max_{h_{1:T}} p(h_{1:T}|v_{1:T}).$$

This is done by the `HMMviterbi` function from the BRML toolbox. Finally, the matrix containing counts of firstname and surname pairs is shown at the end of this exercise.

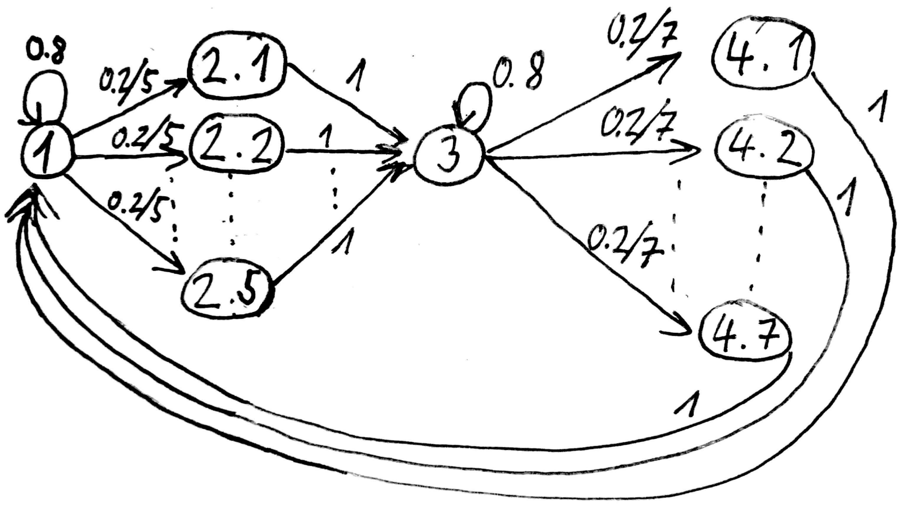


Figure 1: Transition diagram for general and pattern hidden states.

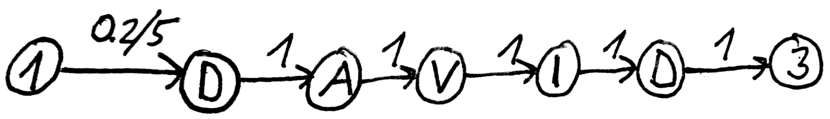


Figure 2: Transition diagram for letter states of the (2.1) pattern state.

Code:

```
% Search for pairs of *firstname*surname*
% where * represents a random sequence
%
% This code is based on the code available on the website
% http://web4.cs.ucl.ac.uk/staff/D.Barber/pmwiki/pmwiki.php?n=Main.Software

clear all ;
run BRMLtoolkit/setup.m ;
import brml.*

addpath patternsearch ;

cpattern{1}='david';
cpattern{2}='anton';
cpattern{3}='fred';
cpattern{4}='jim';
cpattern{5}='barry';

cpattern{6}='barber';
cpattern{7}='ilsung';
cpattern{8}='fox';
cpattern{9}='chain';
cpattern{10}='fitzwilliam';
cpattern{11}='quinceadams';
cpattern{12}='grafvonunterhosen';

for p=1:12
  pattern{p} = name2num(cpattern{p});
end

firstname = 1:5; % 5 first names
surname = 6:12; % 7 surnames

patternstate = 1:12; % pattern states (1 per pattern)
```

```
generalstate = 13:14; % general states

nstates = generalstate(end); % total number of pattern+general states

% make the transistion matrix:
tran = zeros(nstates,nstates);
tran(generalstate(1),generalstate(1)) = 0.8; % not started firstname
for p=firstname
    tran(patternstate(p),generalstate(1)) = 0.2/length(firstname); % into a firstname
    tran(generalstate(2),patternstate(p)) = 1; % out of firstname
end
tran(generalstate(2),generalstate(2)) = 0.8; % not start surname
for p=surname
    tran(patternstate(p),generalstate(2)) = 0.2/length(surname);
    tran(generalstate(1),patternstate(p)) = 1; % out of surname, go back to not start ...
        firstname
end

% make the emission matrix:
a = 0.3; % 30 percent probability of emitting the correct letter
CorruptionProb = ones(26,26)*(1-a)/25;
for i=1:26
    CorruptionProb(i,i) = a;
end

for g=1:length(generalstate)
    pvgh_general(:,g) = ones(26,1)/26; % uniform emission for general states
end
prior.generalstate = [1 0]'; % start in general state 1 with certainty

[phghm,pvgh,ph1,startpatternIDX,endpatternIDX,generalstateIDX] = ...
  patternsearchsetup(pattern,patternstate,generalstate,tran,CorruptionProb,pvgh_general,prior);

load noisystring ;
v = name2num(noisystring) ;

 % threshold for approximation. Set to zero for exact computation.
thresh = 0 ;
printtime = 1;

[alpha,beta,gamma,∆,loglik,loglikpvhstar]=patternsearch(v,phghm,pvgh,ph1,thresh,printtime);

 % just get which patterns these correspond to.
[pnum gnum]=getpattern(∆,startpatternIDX,generalstateIDX);

% construct matrix m(i,j) with the counts of the number of occurrences
%  of the pair (firstname(i), surname(j)
firstnameIDX = startpatternIDX(firstname) ;
surnameIDX = startpatternIDX(surname) ;
firstnames = pnum(find(ismember(∆, firstnameIDX))) ;
surnames = pnum(find(ismember(∆, surnameIDX))) - numel(firstname) ;
% firstname, surname
m = zeros(numel(firstname), numel(surname)) ;
for i=1:numel(surnames)
  m(firstnames(i), surnames(i)) = m(firstnames(i), surnames(i)) + 1 ;
end

disp('Matrix m(i,j) with the counts of the number of occurrences of the pair ...
    (firstname(i), surname(j)') ;
disp(m) ;
```

Output:

```
HMM marginal inference takes 38.240000 seconds
HMM viterbi takes 26.880000 seconds

Matrix m(i,j) with the counts of the number
of occurrences of the pair (firstname(i), surname(j)):
     6     4     8    11    14    10    11
     8     6     4     9    20    11    13
     6     5     9    10    12     6    19
     8     9     7    18    20    12    19
    10     5    10    10     9     8    16
```

Exercise 5.9

We are given a 50×50 grid with 500 people. One grid square (node) can have a value of 0 (no one standing on it) or 1 (one or more people standing on it). From the provided code (see `drunkmover.m`):

- All people can only move diagonally; $(x \pm 1, y \pm 1)$ or $(x \pm 2, y \pm 2)$.
- A non-drunk person has a probability of 0.9801 to move in the same direction as the previous step and 0.063 of moving in each of the other 3 directions.
- The drunk person has equal probability of 0.25 of moving in each of the 4 directions on size-2 steps. Note that the first step taken by the drunk is predetermined in the code.

Proposed solution 1: The grid can be modelled using a Markov network with diagonal edges as shown in Figure 1a. We know a non-drunk person occupying a node (node set to 1) can only move in ± 1 steps. We can devise an area corresponding to the possible nodes such person can move to in time $t + 1$, as seen in Figure 1b.

- we can first assume all nodes represents normal people and deduce a map of all possible walk/location for the next time step.

- with this map, and using the node location of the next time step, we can compare them and look at the locations where there is a 1 in node that are not in the possible walk from the map.

- this set of nodes are the possible nodes of drunk person, OR people who walked off he grid and been reset back to a random position.

- given enough time step, it is possible to retrieve a set of nodes that are possible drunk person candidate, how ever, it is not guaranteed (very unlikely) to find the drunk person.

proposed solution2: - model this for all the time steps (here $T = 100$)

- for each time step, model each grid as 50×50 nodes with no connections between them

- for each node with 1 on time step t, create connections between the node and 4 diagonally neighbouring node on time step $t + 1$, as shown in Figure 2a. This is the possible step taken by the normal people, when we don't know what was the previous step taken from the normal person, thus initially assign probability of 0.25 to each weight equally.

- for each node at time t, where $t > 1$: find out the direction where the normal person has came from. This can be done by checking if at time $t - 1$, if it is the only node that can step on the node on time t (i.e. if it is the only node in the 3×3 grid centre at the node at time t on time $t - 1$). If that is the case, note the 'direction' of walk from time $t - 1$ to time t.

- then modify the probability of the edge between time t and $t + 1$ that belongs to the same direction as the previous step to be 0.9801. Additionally, modify the probability of the remaining edge connected to the node from time t to t+1 to be 0.063.

- remove the edge between node in time t and $t + 1$ if both of the node does not contain at least a person (i.e. 1 in the grid, as oppose to 0)

- run the max flow algorithm on the graph, using the probability on edge to be the weight/cost. Initialize the flow with $f = 1$ and increases to $f_{max} = 499 (= N - 1)$ (excluding the drunk mover), from time $t = 1$ to $t = T$.

- there is no constraint on individual nodes that limit the maximum flow that can flow through each node (i.e. people standing can overlap)

- after we found the Max-flow of $f = 499$ we can look at the remaining nodes with a person (1) for all t that is not part of any flows f from the max flow algorithm. Not all t will contain a free person (1) due to the fact that drunk mover can also be in the same node as the normal person. However, we can deduce a set of nodes and run possible configurations of walks (using the rule given in drunkmover.m, where the drunk person will move 2 steps diagonally in random direction), thus resulting in a set of possible paths for the drunk person.

- if, given $T \to \infty$, it is guaranteed that all of the normal people will have the modified the edge probabilities

- however, note that this method won't work with the fact that the people move off the grid will be re-initialised randomly within the grid. As this will result in the flow not complete from on end $t = 1$ to another $t = T$, and we lose track of the people who move off the grid. This would means we have to be able to start flow from anywhere between $t = 1$ and $t = T$, and thus running max flow algorithm for $f = N - 1$ would not work because we dont know how much flow we would need to assign to account for all the re-initialsed people.

- but this method does give a nice bonus of tracking all the normal people as well ⌣̈

- cite: Zhang, L., Li, Y. and Nevatia, R., 2008, June. Global data association for multi-object tracking using network flows. In Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on (pp. 1-8). IEEE. Vancouver

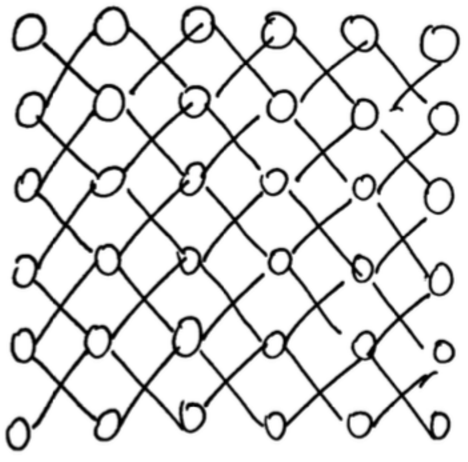


Figure 1a

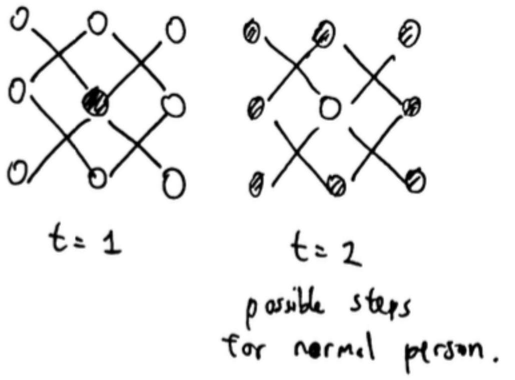
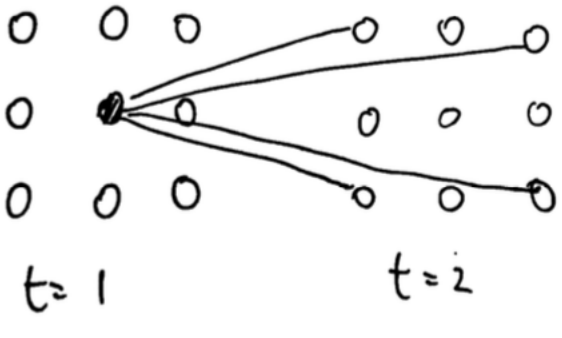
Figure 1b


Figure 2a


Exercise 5.10

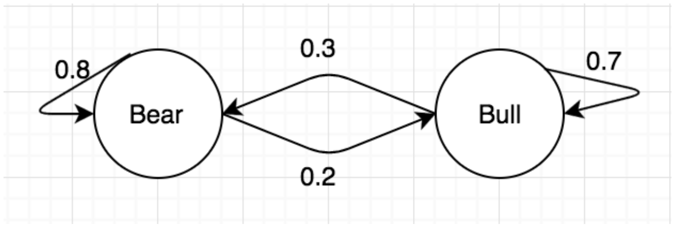
For this exercise we use a Hidden Markov Model with the following variables:

- $H = 2$, amount of hidden states,
- $V = 100$, number of visible states,
- $T = 200$, timepoints

We are given the following matrices to work with:

- p, price of the asset through $T = 200$ timepoints,
- $pbear$, transition matrix for the pbear state,
- $pbull$, transition matrix for the pbull state

Also, the transition matrix between hidden states (bear and bull) is represented by the graph below:



In this particular problem, the price at any given timepoint is dependent not only on the hidden variable (or state). It is also dependent on the price of the asset in timepoint $t-1$ as seen in the two transition matrices given ($pbear$ and $pbull$). Practically, this gives us a concrete emission probability that we can find in the original matrices. Therefore, for any given T_i the emission probability will be the column in $pbear$ and $pbull$ that corresponds to the price of T_{i-1} (one exception is the first price which has a uniform probability $\frac{1}{V}$).

```matlab
import brml.*

H = 2; % number of Hidden states
V = 100; % number of Visible states
T = 200; % length of the time-series

% setup the HMM
% bear is first column
phghm = [0.8 0.3;0.2 0.7]; % transition distribution p(h(t)|h(t-1))
ph1 = [0.5;0.5]; % initial p(h)

v = p;

% initialize the transition matrix.
for j=1:T
    phghm_3(:,:,j) = phghm;
end

% Initialize the emission matrix.
for i=1:T+1
    if i == 1
        pvgh_4(:,1,i) = repmat(1/V,1,V);
        pvgh_4(:,2,i) = repmat(1/V,1,V);
    else
        pvgh_4(:,1,i) = pbear(:,v(i-1));
        pvgh_4(:,2,i) = pbull(:,v(i-1));
    end
end

```

```
[alpha,loglik]=HMMforwardTimeDependent(v,phghm_3,ph1,pvgh_4);
% create final emission probability at timepoint T+1
pvgh_5(:,1,1) = pbear(:,v(T));
pvgh_5(:,2,1) = pbull(:,v(T));

% find the value of the gain
pvgh_final = pvgh_5(:,:,1)*phghm_3(:,:,T)*alpha(:,T);
result = find(pvgh_final == max(pvgh_final)) - v(T)

%standard deviation of the gain
sum = zeros(1, T-1);
for y = 2:T-1
    sum(1,y) = v(y) - v(y-1);
end
% add the result to the array
sum = [sum result]
% calculate the standard deviation
sd = sqrt((result-mean(sum))^2/(V))
```

We have used the HMM forward algorithm to complete this task; deciding to use its Time-Dependent variant. The reason for this choice is the following: a given price can be observed more than once and we risk overwriting the probability in the main array when this happens. This would lead to wrong calculations. Below we can find the result of the calculations made.

The final value of the gain and the standard deviation for this specific gain are:

```
gain = 2
sd = 0.20
```

Exercise 5.11

(1.) We use the two equations we have relating distance, velocity and acceleration:

$$x_{t+1} = x_t + \delta v_t \quad \text{and} \quad v_{t+1} = v_t + \delta a_t$$

Then, by substitution, we can write:

$$\begin{aligned} x_{t+1} &= x_t + \delta\left(v_{t-1} + \delta a_{t-1}\right) \\ &= x_t + \delta v_{t-1} + \delta^2 a_{t-1} \\ &= x_t + \delta\left(x_t - x_{t-1}\right)/\delta + \delta^2 a_{t-1} \\ &= x_t + x_t - x_{t-1} + \delta^2 a_{t-1} \\ &= 2x_t - x_{t-1} + \delta^2 a_{t-1} \end{aligned}$$

We will eventually show that

$$x_i(102) = \delta^2\left(100a_i(1) + 99a_i(2) + ... + a_i(100)\right)$$

but for now, we claim that:

$$x_{102} = 101x_2 - 100x_1 + \delta^2\left(100a_1 + 99a_2 + ... + a_{100}\right) = 101x_2 - 100x_1 + \delta^2\sum_{i=1}^{100}(101-i)a_i$$

or in more a more general setting:

Claim:
For $n \geq 1$ we have:

$$x_{n+2} = (n+1)x_2 - nx_1 + \delta^2\sum_{i=1}^{n}(n+1-i)a_i$$

This will be a proof by induction:

Base case:

For $n = 1$, we observe that this evaluates to:

$$x_3 = 2x_2 - 1x_1 + \delta^2 a_1$$

which is consistent with the first formula we obtained. Base case is established.

Inductive hypothesis:
Assume true for $n \leq k-1$. In particular, we have that:

$$x_{k+1} = kx_2 - (k-1)x_1 + \delta^2 \sum_{i=1}^{k-1} (k-i)a_i$$

and

$$x_k = (k-1)x_2 - (k-2)x_1 + \delta^2 \sum_{i=1}^{k-2} (k-1-i)a_i$$

Inductive step:
Here we prove for $n = k$:

$$\begin{aligned}
x_{(k)+2} = x_{k+2} &= 2x_{k+1} - x_k + \delta^2 a_k \\
&= 2\left(kx_2 - (k-1)x_1 + \delta^2 \sum_{i=1}^{k-1}(k-i)a_i\right) - \left((k-1)x_2 - (k-2)x_1 + \delta^2 \sum_{i=1}^{k-2}(k-1-i)a_i\right) + \delta^2 a_k \\
&= 2kx_2 - 2(k-1)x_1 - (k-1)x_2 + (k-2)x_1 + 2\delta^2 \sum_{i=1}^{k-1}(k-i)a_i - \delta^2 \sum_{i=1}^{k-2}(k-1-i)a_i + \delta^2 a_k \\
&= (2k - (k-1))\, x_2 + ((k-2) - 2(k-1))\, x_1 + \delta^2 \left(\sum_{i=1}^{k-1} 2(k-i)a_i - \sum_{i=1}^{k-2}(k-1-i)a_i + a_k\right) \\
&= (k+1)\, x_2 + (-k)\, x_1 + \delta^2 \left(\sum_{i=1}^{k-1} 2(k-i)a_i - \sum_{i=1}^{k-2}(k-1-i)a_i + a_k\right) \\
&= (k+1)\, x_2 - kx_1 + \delta^2 \left(\sum_{i=1}^{k-2} a_i\left(2(k-i) - (k-1-i)\right) + 2a_{k-1} + a_k\right) \\
&= (k+1)\, x_2 - kx_1 + \delta^2 \left(\sum_{i=1}^{k-2} a_i\left(k+1-i\right) + 2a_{k-1} + a_k\right) \\
&= (k+1)\, x_2 - kx_1 + \delta^2 \sum_{i=1}^{k} \left(k+1-i\right) a_i
\end{aligned}$$

□

Now, we have proved our claim for all $n \geq 1$. Accordingly, we can write:

$$\begin{aligned}
x_{102} &= 101x_2 - 100x_1 + \delta^2 \sum_{i=1}^{100}(101-i)a_i \\
&= 101x_2 - 100x_1 + \delta^2 \left(100a_1 + 99a_2 + ... + a_{100}\right)
\end{aligned}$$

or, by introducing the 3 dimensions where $i \in \{1, 2, 3\}$:

$$\begin{aligned}
x_i(102) &= 101x_i(2) - 100x_i(1) + \delta^2 \sum_{j=1}^{100}(101-j)a_i(j) \\
&= \delta^2 \sum_{j=1}^{100}(101-j)a_i(j) \\
&= \frac{1}{100}\left(100a_i(1) + 99a_i(2) + ... + a_i(100)\right)
\end{aligned}$$

as $x_i(1) = x_i(2) = 0$ for all i since $x_1 = v_1 = (0,0,0)^T$ and $x_2 = x_1 + \delta v_1$, and as $\delta = 0.1$ and so $\delta^2 = 1/100$.

$$\begin{aligned}
4.71 &= \frac{1}{100}\left(100a_1(1) + 99a_1(2) + ... + a_1(100)\right)\\
-6.97 &= \frac{1}{100}\left(100a_2(1) + 99a_2(2) + ... + a_2(100)\right)\\
8.59 &= \frac{1}{100}\left(100a_3(1) + 99a_3(2) + ... + a_3(100)\right)
\end{aligned}$$

(2.) From part 1, we have that:

$$\begin{aligned}
471 &= 100a_1(1) + 99a_1(2) + ... + a_1(100)\\
-697 &= 100a_2(1) + 99a_2(2) + ... + a_2(100)\\
859 &= 100a_3(1) + 99a_3(2) + ... + a_3(100)
\end{aligned}$$

where $a_i(t) \in \{-1, 0, 1\}$.
Fuel consumption, denoted by F, is equal to:

$$\begin{aligned}
F &= \sum_{t=1}^{100}\sum_{i=1}^{3} |a_i(t)|\\
&= \sum_{t=1}^{100} \left(|a_1(t)| + |a_2(t)| + |a_1(t)|\right)
\end{aligned}$$

where $|a_i(t)| \in \{0, 1\}$.
To minimize fuel consumption, we need as few $a_i(t)$ being non-zero as possible.
From the system of equations above we can easily determine this:

$$\begin{aligned}
471 &= 100(1) + 99(1) + 98(1) + 97(1) + 77(1)\\
-697 &= 100(-1) + 99(-1) + 98(-1) + 97(-1) + 96(-1) + 95(-1) + 94(-1) + 18(-1)\\
859 &= 100(1) + 99(1) + 98(1) + 97(1) + 96(1) + 95(1) + 94(1) + 93(1) + 87(1)
\end{aligned}$$

Therefore, clearly, we have that:

$$\begin{aligned}
F &= 5 + |-8| + 9\\
&= 5 + 8 + 9\\
&= 22
\end{aligned}$$

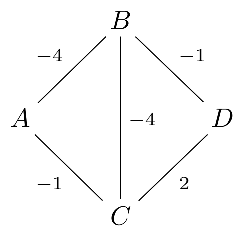
22 units of fuel is the minimum possible to consume.

(3.) If there is a given sequence $\mathbf{a}_1, \mathbf{a}_2, ..., \mathbf{a}_{100}$ then the values for $a_i(t)$ are already determined. The individual components of these vectors need summing and then these results need summing over all vectors. In pseudocode:

```
$F := 0$;
for loop from $i = 1$ to 100:
$F := F + (a_1(i) + a_2(i) + a_3(i))$
end;
return $F$;
```

Exercise 5.12

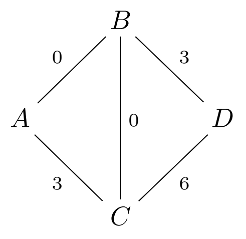
Consider the following simple graph with a number of negative weights. This will illustrate why it is not possible to do this.



Let us apply Dijkstra's algorithm to the graph, beginning at node A and ending at node D.

Visited Nodes	A	B	C	D
	∞	∞	∞	∞
A	0	-4	-1	∞
A,B	0	-4	-8	-5
A,B,C	0	-4	-8	-6

We find a minimum weighted path of weight -6 which travels $A \to B \to C \to D$.
Now, if we apply the suggested technique of subtracting $u^* = \min\{\text{edge weights}\} = -4$ from each edge weight, we have obtained a new graph with non-negative weights as follows:



Let us apply Dijkstra's algorithm to this graph, again beginning at node A and ending at node D.

Visited Nodes	A	B	C	D
	∞	∞	∞	∞
A	0	0	3	∞
A,B	0	0	0	3
A,B,C	0	0	0	3

We find a minimum weighted path of weight 3 and it is a different path from that found above, namely $A \to B \to D$.

Ultimately the reason why Dijkstra's algorithm and the suggested technique fail in general is because the algorithm works on the assumption that paths don't become lighter in weight and only get heavier. In our example, the first graph presents a route with a negative weighted cycle, namely $B \to C \to D$, that makes it cheaper to go via node C before hitting D. If you like. it is cycles of negative weight that cause this failure in general.

Exercise 5.13

In this exercise we are asked to provide the minimum cost of travelling from planet 1 to planet 1725 given some constraints in calculating the weights of edges in the graph:

- Simo Hurtta does not collect any taxes from planet 1.
- The distance between a given planet A and a given planet B is the euclidean distance between A and B minus the tax collected at planet B.

We are given three different matrices:

- A, which is an adjacency matrix and indicates the connections between planets.
- x, which contains the position multi-dimensional vectors representing the position of each planet.

- t, which represents the amount of taxes collected at a given planet once visited.

The first task involves transforming the original adjacency matrix by assigning the weights of going from a given planet A to a given planet B. We traverse the matrix checking for links between planets. When a link is found, we calculate the distance with the equation given $\sqrt{(x^i - x^j)^2} - t^j$, where t^j represents the tax collected at planet j. This is done from line 7 to line 18 in our code.

Once we have a graph with the correct weights, we need to calculate the path with minimum cost. One important note is that the graph contains negative weights, which is why an algorithm like Dijkstra's would not perform well. We tried a variation given by the Bellman-Ford algorithm. However, this didn't provide enough flexibility in order to solve for *negative cycles*.

We then turned to a probabilistic approach, using the `mostprobablepath` method provided in the *BRML toolbox*, highlighted in the solution you can find below.

```
1  import brml.*
2
3  %prepare the data
4  sa = size(A);
5  dist = inf(2000);
6
7  % create new matrix with weighted edges based on the constraints
8  for i=1:sa(1)
9      for j=1:sa(1)
10         if i == j
11             continue
12         else
13             if A(i,j) > 0
14                 dist(i,j) = sqrt(sum((x(:,i) - x(:,j)) .^ 2)) - t(1,j);
15             end
16         end
17     end
18 end
19
20 % calculate the distance
21 [a b]=assign([1 1725]);
22 [optpath pathweight]=mostprobablepath(-dist',a,b);
23 fprintf('shortest path has weight %g\n',-pathweight); optpath
```

This procedure gave the following result:

```
>> exec_5_13
shortest path has weight -209.65
```

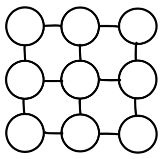
Exercise 6.7

For this Markov random field , we have $p = Z^{-1}\prod_{i<j}\phi(x_i, x_j)$ defined on an $n \times n$ lattice. We need to sum over all possible states of $\phi(x_i, x_j)$ to compute the normalisation constant Z.

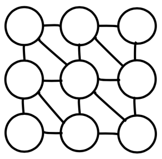
$$Z = \sum_{i,j}\prod_{i<j}\phi(x_i, x_j)$$

In this exercise, we stack all the variables in the ith column into a cluster variable. Since our initial variables are binary, taking the product of the potentials over each column adds a complexity of $O(2^n)$, where n is the number of atoms per column. The complexity involved taking the product of the cluster variables X_t should scale with $n-1$ (the number of links between the cluster variables.) The complexity of calculating the normalisation constant based on this representation is then $(n-1) \times 2^n$.

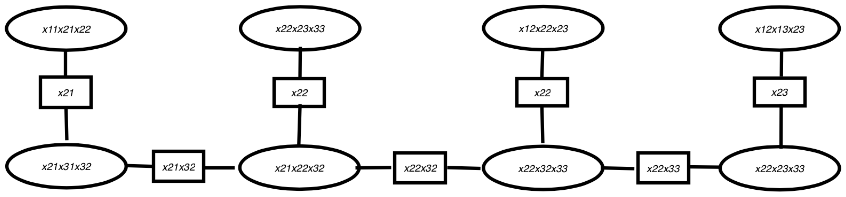
Consider a binary variable Markov field on the following lattice (let $n = 3$ for illustrative purposes):



The normalisation constant for the system can be found more efficiently using the Junction Tree Algorithm. **Moralisation**: not required since the distribution is undirected. **Triangulation**: we introduce a chord into every loop of length four or more (see below). We now have sets of size-3 cliques. Small clique sizes are convenient because the complexity of the normalisation constant scales with clique sizes in the triangulated graph (absorption requires computing tables on the cliques).



Junction tree: We can now form a junction tree from cliques of the triangulated graph, removing any unnecessary links in a loop on the cluster graph (see below).



Potential assignment: To assign potentials to junction tree cliques we can use the `jtassignpot.m` code in the *BRML Toolbox.* This function lists all the potentials of a function and orders the junction tree cliques arbitrarily. It then searches through the cliques, for each potential, until the first is encountered for which the potential variables are a subset of the JT clique variables. Subsequently, the potential on each JT clique is taken as the product of all clique potentials assigned to the JT clique. Finally, all separators are assigned to unity.

Message propagation: We are now able to calculate $logZ$ by the function `absorption.m` in *BRML Toolbox.* This code forms a full absorption schedule
Bare in mind that using the JTA,

$$p(X) = \frac{1}{Z}\frac{\prod_c \phi(X_c)}{\prod_s \phi(X_s)} \to Z = \sum_X \frac{\prod_c \phi(X_c)}{\prod_s \phi(X_s)},$$

where c are the indices of the cliques (ovals) in our figure above and s those of the separators (rectangles). For a consistent JT, marginal cliques cancel with their corresponding separators to give a unity term. We are left with:

$$Z = \sum_{X_c} \phi(X_c)$$

Exercise 6.9

1. Please see the code "Part 1" and output at the end of this exercise.

2. Let D be the set of diseases and S the set of symptoms. Let us define a set of quadruples

   $T = \{(s, d_i, d_j, d_k)|s, d_i, d_j, d_k$ iff a symptom s connects with parent diseases $d_i, d_j, d_k; s \in S; d_i, d_j, d_k \in D\}$.

The joint distribution for the belief network is

$$p(s_1, \dots, s_{40}, d_1, \dots, d_{20}) = \Big(\prod_{d \in D} p(d_i) \Big) \Big(\prod_{(s,d_i,d_j,d_k) \in T} p(s|d_i, d_j, d_k) \Big).$$

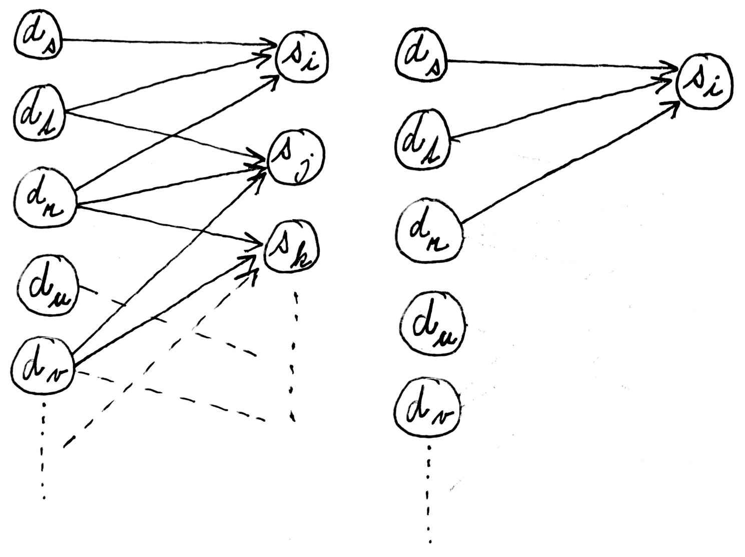
Before we compute $p(s_i, d_s, d_t, d_r)$ where $(s_i, d_s, d_t, d_r) \in T$, let us define $T^* = T \setminus (s_i, d_s, d_t, d_r)$, $S^* = S \setminus s_i$, and $D^* = D \setminus d_s, d_t, d_r$. Then

$$\begin{aligned}
p(s_i, d_s, d_t, d_r) &= \sum_{D^*} \sum_{S^*} \prod_{d \in D} p(d) \prod_{(s,d_i,d_j,d_k) \in T} p(s|d_i, d_j, d_k) \\
&= \sum_{D^*} \prod_{d \in D} p(d) \sum_{S^*} \prod_{(s,d_i,d_j,d_k) \in T} p(s|d_i, d_j, d_k) \\
&= p(d_s)p(d_t)p(d_r)p(s_i|d_s, d_t, d_r) \sum_{D^*} \prod_{d \in D^*} p(d) \sum_{S^*} \prod_{(s,d_i,d_j,d_k) \in T^*} p(s|d_i, d_j, d_k) \\
&= p(d_s)p(d_t)p(d_r)p(s_i|d_s, d_t, d_r) \sum_{D^*} \prod_{d \in D^*} p(d) \prod_{(s,d_i,d_j,d_k) \in T^*} \underbrace{\sum_{S^*} p(s|d_i, d_j, d_k)}_{1} \\
&= p(d_s)p(d_t)p(d_r)p(s_i|d_s, d_t, d_r) \sum_{D^*} \prod_{d \in D^*} p(d) \\
&= p(d_s)p(d_t)p(d_r)p(s_i|d_s, d_t, d_r) \prod_{d \in D^*} \underbrace{\sum_{D^*} p(d)}_{1} \\
&= p(d_s)p(d_t)p(d_r)p(s_i|d_s, d_t, d_r).
\end{aligned}$$

Finally, we just marginalize over three diseases to get $p(s_i)$:

$$p(s_i) = \sum_{d_s,d_t,d_r} p(s_i, d_s, d_t, d_r) = \sum_{d_s,d_t,d_r} p(d_s)p(d_t)p(d_r)p(s_i|d_s, d_t, d_r).$$

As we can see, in order to obtain the marginal probability of a particular symptom, we only need to consider prior probabilities of the three parent diseases and the conditional probability of the symptom given the three parent diseases.



(a) (b)

Figure 3: (a) Illustration of the belief network of the given distribution. (b) Belief network of the given distribution after marginalization over all the symptoms except for the s_i.

We can arrive to the same result by reasoning over the graph of the belief network and using Definition 3.3 (properties of belief networks) in the BRML book. Considering a particular symptom

s_i, marginalizing over other symptoms disconnects all the diseases from the symptom s_i except for the parents of the symptom s_i. Therefore, we get a graph where only parents of the symptom s_i are connected to the symptom s_i. Moreover, using again Definition 3.3, we can see that all the parent diseases are independent from each other. This is illustrated in the figure 3. We can then compute $p(s_i, d_s, d_t, d_r)$ as

$$p(s_i, d_s, d_t, d_r) = p(d_s, d_t, d_r)p(s_i|d_s, d_t, d_r) = p(d_s)p(d_t)p(d_r)p(s_i|d_s, d_t, d_r).$$

Using this simplified computation, we are able to get the same results in shorter time. The computation using JTA took 1.90 seconds and only 0.25 seconds when using our method.

3. Please see the code "Part 3" and output at the end of this exercise.

Code:

```
clear all
run BRMLtoolkit/setup.m
import brml.*

load diseaseNet.mat

YES = 1 ;
NO = 2 ;

disp('##########################################################################') ;
disp('Part 1') ;
disp('##########################################################################') ;
% setup the Junction Tree
[jtpot jtsep infostruct]=jtree(pot);

% do full round of absorption
jtpot=absorption(jtpot,jtsep,infostruct);

% p(s_i=1)
 % find a single JT potential that contains s_i
for i=21:60
  jtpotnum = whichpot(jtpot,i,1);
  % sum over everything but s_i
  margpot = sumpot(jtpot(jtpotnum),i,0);
  disp(['p(s_' num2str(i-20) ' =1 ) = ' num2str(margpot.table(YES)./sum(margpot.table))]);
end

disp(' ') ;
disp('##########################################################################') ;
disp('Part 2') ;
disp('##########################################################################') ;
% multiply p(s_i|d_s,d_t,d_r)p(d_s)p(d_t)p(d_r)
% and sum over everything but s_i
for i=21:60
  d = setdiff(pot{i}.variables, i) ;
  multpot = multpots([pot(i), pot(d)]) ;
  margpot = sumpot(multpot, i, 0) ;
  disp(['p(s_' num2str(i-20) ' = 1) = ' num2str(margpot.table(YES)./sum(margpot.table))]);
end

disp(' ') ;
disp('##########################################################################') ;
disp('Part 3') ;
disp('##########################################################################') ;
% Clamp variables to the given states
newpot = setpot(pot,[21:30],[repmat(YES, 1, 5) repmat(NO, 1, 5)]);
% To use the JT we must have no missing variables, so squeeze the potential
[newpot newvars oldvars]=squeezepots(newpot);
% setup the Junction Tree
[jtpot2 jtsep2 infostruct2]=jtree(newpot);
% do full round of absorption
[jtpot2 jtsep2 logZ2]=absorption(jtpot2,jtsep2,infostruct2);
% After absorbing, transform back to original variables
jtpot2=changevar(jtpot2,newvars,oldvars);

```

```matlab
for i=1:20
  % find a single JT potential that contains d_i
  jtpotnum = whichpot(jtpot2,i,1);
  % sum over everything but d_i
  margpot=sumpot(jtpot2(jtpotnum),i,0);
  % normalise to get conditional probability
  disp(['p(d_' num2str(i) ' = 1|s_1:10) = ' ...
      num2str(margpot.table(YES)./sum(margpot.table))]) ;
end
```

Output:

```
##########################################################################
Part 1
##########################################################################
p(s_1 = 1) = 0.44183
p(s_2 = 1) = 0.45668
p(s_3 = 1) = 0.44141
p(s_4 = 1) = 0.49127
p(s_5 = 1) = 0.49389
p(s_6 = 1) = 0.65748
p(s_7 = 1) = 0.50456
p(s_8 = 1) = 0.26869
p(s_9 = 1) = 0.64908
p(s_10 = 1) = 0.49074
p(s_11 = 1) = 0.42255
p(s_12 = 1) = 0.4291
p(s_13 = 1) = 0.54502
p(s_14 = 1) = 0.63296
p(s_15 = 1) = 0.42954
p(s_16 = 1) = 0.45879
p(s_17 = 1) = 0.42756
p(s_18 = 1) = 0.40426
p(s_19 = 1) = 0.58209
p(s_20 = 1) = 0.58959
p(s_21 = 1) = 0.76127
p(s_22 = 1) = 0.69559
p(s_23 = 1) = 0.5087
p(s_24 = 1) = 0.41996
p(s_25 = 1) = 0.35194
p(s_26 = 1) = 0.38961
p(s_27 = 1) = 0.32597
p(s_28 = 1) = 0.46962
p(s_29 = 1) = 0.52287
p(s_30 = 1) = 0.71731
p(s_31 = 1) = 0.5242
p(s_32 = 1) = 0.3537
p(s_33 = 1) = 0.51268
p(s_34 = 1) = 0.5294
p(s_35 = 1) = 0.38575
p(s_36 = 1) = 0.48909
p(s_37 = 1) = 0.6336
p(s_38 = 1) = 0.5896
p(s_39 = 1) = 0.42316
p(s_40 = 1) = 0.52823

##########################################################################
Part 2
##########################################################################
p(s_1 = 1) = 0.44183
p(s_2 = 1) = 0.45668
```

```
p(s_3 = 1) = 0.44141
p(s_4 = 1) = 0.49127
p(s_5 = 1) = 0.49389
p(s_6 = 1) = 0.65748
p(s_7 = 1) = 0.50456
p(s_8 = 1) = 0.26869
p(s_9 = 1) = 0.64908
p(s_10 = 1) = 0.49074
p(s_11 = 1) = 0.42255
p(s_12 = 1) = 0.4291
p(s_13 = 1) = 0.54502
p(s_14 = 1) = 0.63296
p(s_15 = 1) = 0.42954
p(s_16 = 1) = 0.45879
p(s_17 = 1) = 0.42756
p(s_18 = 1) = 0.40426
p(s_19 = 1) = 0.58209
p(s_20 = 1) = 0.58959
p(s_21 = 1) = 0.76127
p(s_22 = 1) = 0.69559
p(s_23 = 1) = 0.5087
p(s_24 = 1) = 0.41996
p(s_25 = 1) = 0.35194
p(s_26 = 1) = 0.38961
p(s_27 = 1) = 0.32597
p(s_28 = 1) = 0.46962
p(s_29 = 1) = 0.52287
p(s_30 = 1) = 0.71731
p(s_31 = 1) = 0.5242
p(s_32 = 1) = 0.3537
p(s_33 = 1) = 0.51268
p(s_34 = 1) = 0.5294
p(s_35 = 1) = 0.38575
p(s_36 = 1) = 0.48909
p(s_37 = 1) = 0.6336
p(s_38 = 1) = 0.5896
p(s_39 = 1) = 0.42316
p(s_40 = 1) = 0.52823

#########################################################################
Part 3
#########################################################################
p(d_1 = 1|s_1:10) = 0.029776
p(d_2 = 1|s_1:10) = 0.38176
p(d_3 = 1|s_1:10) = 0.95423
p(d_4 = 1|s_1:10) = 0.39664
p(d_5 = 1|s_1:10) = 0.49647
p(d_6 = 1|s_1:10) = 0.43515
p(d_7 = 1|s_1:10) = 0.18749
p(d_8 = 1|s_1:10) = 0.70118
p(d_9 = 1|s_1:10) = 0.043127
p(d_10 = 1|s_1:10) = 0.61031
p(d_11 = 1|s_1:10) = 0.28732
p(d_12 = 1|s_1:10) = 0.48983
p(d_13 = 1|s_1:10) = 0.8996
p(d_14 = 1|s_1:10) = 0.61956
p(d_15 = 1|s_1:10) = 0.92048
p(d_16 = 1|s_1:10) = 0.7061
```

```
p(d_17 = 1|s_1:10) = 0.20125
p(d_18 = 1|s_1:10) = 0.90849
p(d_19 = 1|s_1:10) = 0.86497
p(d_20 = 1|s_1:10) = 0.88393
```

Exercise 6.10

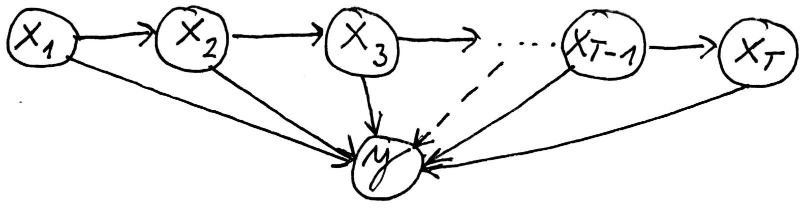


Figure 4: Belief network of the given distribution.

1. The belief network can be seen in the figure 4. The junction tree of this distribution is:

$$\begin{tikzcd}
x_1x_2y \arrow[r, dash] & \boxed{x_2y} \arrow[r, dash] & x_2x_3y \arrow[r, dash] & \boxed{x_3y} \arrow[r, dash] & \cdots \arrow[r, dash] & \boxed{x_{T-1}y} \arrow[r, dash] & x_{T-1}x_Ty
\end{tikzcd}$$

   JTA complexity is exponential with tree width. The tree width is linear with T as indicated in the graph of the junction tree. Therefore, the complexity of computing $p(x_{T-1}, x_T, y)$ is exponential with T. We marginalize over x_{T-1} and y to get $p(x_T)$. Since all the variables are binary, this requires summation of $2 \cdot 2 = 4$ values, i.e. a constant number of operations. Therefore, the overall complexity is still exponential with T.

2. We can compute the marginal as follows:

$$\begin{aligned}
p(x_T) &= \sum_{\{y, x_1, \ldots, x_{T-1}\}} p(y, x_1, \ldots, x_T) \\
&= \sum_{\{y, x_1, \ldots, x_{T-1}\}} p(y|x_1, \ldots, x_T) p(x_1) p(x_2|x_1) \cdots p(x_{T-1}|x_{T-2}) p(x_T|x_{T-1}) \\
&= \sum_{\{x_1, \ldots, x_{T-1}\}} p(x_1) p(x_2|x_1) \cdots p(x_{T-1}|x_{T-2}) p(x_T|x_{T-1}) \underbrace{\sum_y p(y|x_1, \ldots, x_T)}_{1} \\
&= \sum_{x_1} p(x_1) \underbrace{\sum_{x_2} p(x_2|x_1) \cdots \underbrace{\sum_{x_{T-2}} p(x_{T-2}|x_{T-3}) \underbrace{\sum_{x_{T-1}} p(x_{T-1}|x_{T-2}) p(x_T|x_{T-1})}_{\gamma_{T-2}(x_{T-2}, x_T)}}_{\gamma_{T-3}(x_{T-3}, x_T)}}_{\gamma_1(x_1, x_T)} \\
&= \sum_{x_1} p(x_1) \gamma_1(x_1, x_T).
\end{aligned}$$

   For a given $\gamma_{t+1}(x_{t+1}, x_T)$, the computation of $\gamma_t(x_t, x_T) = \sum_{x_{t+1}} p(x_{t+1}|x_t) \gamma_{t+1}(x_{t+1}, x_T)$ for fixed x_t and x_T requires 3 operations (2x multiplication and 1x summation); for all possible values of x_t and x_T it takes $4 \cdot 3 = 12$ operations. The computation of $\gamma_1(x_1, x_T)$ for all possible values of x_1 and x_T takes $12(T-2)$ operations. Finally, in order to compute $p(x_T) = \sum_{x_1} p(x_1)\gamma_1(x_1, x_T)$ for all x_T, we need $12(T-2) + 2(2+1)$ operations. Therefore, the overall complexity is $O\big(12(T-2) + 2(2+1)\big) = O(T)$, i.e. linear with T.

Exercise 6.12

(1.) Removing the neighbour of clique 1 from the product of higher order cliques, we have:

$$p(\mathcal{X}) = \frac{\phi_1(\mathcal{X}_1)\prod_{c\geq 2}\phi_c(\mathcal{X}_c)}{\psi_1(S_1)\prod_{s\geq 2}\psi_s(\mathcal{S}_s)} = \frac{\phi_1(\mathcal{X}_1)\phi_d(\mathcal{X}_d)\prod_{c\geq 2, c\neq d}\phi_c(\mathcal{X}_c)}{\psi_1(S_1)\prod_{s\geq 2}\psi_s(\mathcal{S}_s)}.$$

In analogy to the absorption procedure, clique d 'absorbs' information from neighbouring clique 1, summing over all variables in clique 1 that are not in separator S_1. For this we define a new separator,

$$\psi_1'(\mathcal{S}_1) = \sum_{\mathcal{X}_1\backslash\mathcal{S}_1}\phi_1(\mathcal{X}_1),$$

and refine the potential for clique 1's neighbours, $\phi_d(X_d)$, using:

$$\phi_d'(\mathcal{X}_d) = \phi_d(\mathcal{X}_d)\frac{\psi_1'(\mathcal{S}_1)}{\psi_1(\mathcal{S}_1)} = \phi_d(\mathcal{X}_d)\frac{\sum_{\mathcal{X}_1\backslash S_1}\phi_1(\mathcal{X}_1)}{\psi_1(\mathcal{S}_1)}.$$

We notice that under absorption, the potential of clique d divided by that of its separator is invariant:

$$\frac{\phi_d'(\mathcal{X}_d)}{\psi_1'(\mathcal{S}_1)} = \frac{\phi_d(\mathcal{X}_d)\frac{\psi_1'(\mathcal{S}_1)}{\psi_1(\mathcal{S}_1)}}{\psi_1'(\mathcal{S}_1)} = \frac{\phi_d(\mathcal{X}_d)}{\psi_1(\mathcal{S}_1)}.$$

This implies,

$$p(\mathcal{X}) = \frac{\phi_1(\mathcal{X}_1)\phi_d'(\mathcal{X}_d)\prod_{c\geq 2, c\neq d}\phi_c(\mathcal{X}_c)}{\psi_1'(S_1)\prod_{s\geq 2}\psi_s(\mathcal{S}_s)}.$$

Now clique d has calibrated to clique 1, receiving information about 1's marginalisation; what it 'thinks' clique 1's marginal probability should be. We can advance to clique 2 with a new JT of the form:

$$p_2(\mathcal{X}) = \frac{\phi_d'(\mathcal{X}_d)\prod_{c\geq 2, c\neq d}\phi_c(\mathcal{X}_c)}{\prod_{s\geq 2}\psi_s(\mathcal{S}_s)}, \tag{1}$$

where $p_2(\mathcal{X})$ refers to the junction tree on cliques $c \geq 2$ and $s \geq 2$. Clique 1 and separator 1 are not (graphically) present in this junction tree and are hence eliminated from the expression.

(2.) If clique 2 is a neighbour of 1, let $\phi_d'(\mathcal{X}_d) = \phi_2'(\mathcal{X}_2)$ in equation (1). Then, denoting the potential corresponding to clique 2's neighbour as $\phi_d(\mathcal{X}_d)$:

$$p_2(\mathcal{X}) = \frac{\phi_2'(\mathcal{X}_2)\prod_{c\geq 3}\phi_c(\mathcal{X}_c)}{\psi_2(\mathcal{S}_2)\prod_{s\geq 3}\psi_s(\mathcal{S}_s)} = \frac{\phi_2'(\mathcal{X}_2)\phi_d(\mathcal{X}_d)\prod_{c\geq 3, c\neq d}\phi_c(\mathcal{X}_c)}{\psi_2(\mathcal{S}_2)\prod_{s\geq 3}\psi_s(\mathcal{S}_s)}.$$

To eliminate clique 2, we define a new separator,

$$\psi_2'(\mathcal{S}_2) = \sum_{\mathcal{X}_2\backslash\mathcal{S}_2}\phi_2(\mathcal{X}_2),$$

and refine the potential of clique 2's neighbours using:

$$\phi_d'(\mathcal{X}_d) = \phi_d(\mathcal{X}_d)\frac{\psi_2'(\mathcal{S}_2)}{\psi_2(\mathcal{S}_2)} = \phi_d(\mathcal{X}_d)\frac{\sum_{\mathcal{X}_2\backslash S_2}\phi_2(\mathcal{X}_2)}{\psi_2(\mathcal{S}_2)}.$$

Since under absorption, the potential of clique d divided by that of its separator is invariant:

$$p_2(\mathcal{X}) = \frac{\phi_d'(\mathcal{X}_d)\phi_2'(\mathcal{X}_2)\prod_{c\geq 3, c\neq d}\phi_c(\mathcal{X}_c)}{\psi_2'(\mathcal{S}_2)\prod_{s\geq 3}\psi_s(\mathcal{S}_s)}.$$

Then clique d is absorbed by clique 2 and we can advance to clique 3 with a new JT of the form:

$$p_3(\mathcal{X}) = \frac{\phi_d'(\mathcal{X}_d)\prod_{c\geq 3, c\neq d}\phi_c(\mathcal{X}_c)}{\prod_{s\geq 3}\psi_s(\mathcal{S}_s)},$$

Note that we have assumed clique 2 is a neighbour of clique 1 for simplicity. If it were not a neighbour, utilising the same procedure:

$$p_2(\mathcal{X}) = \frac{\phi'_{d1}(\mathcal{X}_{d1})\prod_{c\geq 2, c\neq d1}\phi_c(\mathcal{X}_c)}{\prod_{s\geq 2}\psi_s(\mathcal{S}_s)} = \frac{\phi'_{d1}(\mathcal{X}_{d1})\phi_2(\mathcal{X}_2)\phi'_{d2}(\mathcal{X}_{d2})\prod_{c\geq 3, c\neq d1, c\neq d2}\phi_c(\mathcal{X}_c)}{\psi'_2(\mathcal{S}_2)\prod_{s\geq 3}\psi_s(\mathcal{S}_s)} \rightarrow$$

$$p_3(\mathcal{X}) = \frac{\phi'_{d1}(\mathcal{X}_{d1})\phi'_{d2}(\mathcal{X}_{d2})\prod_{c\geq 3, c\neq d1, c\neq d2}\phi_c(\mathcal{X}_c)}{\prod_{s\geq 3}\psi_s(\mathcal{S}_s)},$$

where in this case $\phi'_{d1}(\mathcal{X}_{d1})$ corresponds to the updated potential of clique 1's neighbours and $\phi'_{d2}(\mathcal{X}_{d2})$ to that of clique 2's neighbours. Continuing in this manner, once $n-2$ cliques have been eliminated and 2 neighbouring cliques are left in the JT[1]:

$$p_{n-1}(\mathcal{X}) = \frac{\phi'_{n-1}(\mathcal{X}_{n-1})\phi_n(\mathcal{X}_n)}{\psi_{n-1}(\mathcal{X}_{n-1})\psi_n(\mathcal{X}_n)},$$

For simplicity in the notation, we assume that clique n has not absorbed any information from its neighbours during the elimination process (although this is irrelevant for our final computation - just a note on notation). Defining a separator,

$$\psi'_{n-1}(\mathcal{S}_{n-1}) = \sum_{\mathcal{X}_{n-1}\backslash\mathcal{S}_{n-1}}\phi_{n-1}(\mathcal{X}_{n-1}),$$

we can explicitly work out an updated potential for clique $n-1$'s neighbour clique n as a function of the old potentials:

$$\phi'_n(\mathcal{X}_n) = \phi_n(\mathcal{X}_n)\frac{\psi'_{n-1}(\mathcal{S}_{n-1})}{\psi_{n-1}(\mathcal{S}_{n-1})} = \phi_n(\mathcal{X}_n)\frac{\sum_{\mathcal{X}_{n-1}\backslash S_{n-1}}\phi_{n-1}(\mathcal{X}_{n-1})}{\psi_{n-1}(\mathcal{S}_{n-1})},$$

The expression above is equivalent to computing the marginal $p(X_n)$, since,

$$p(\mathcal{X}_n) = \sum_{\mathcal{X}_{n-1}\backslash\mathcal{S}_{n-1}} p(\mathcal{X}_{n-1}\cup\mathcal{X}_n) = \sum_{\mathcal{X}_{n-1}\backslash\mathcal{S}_{n-1}}\frac{\phi_{n-1}(\mathcal{X}_{n-1})\phi_n(\mathcal{X}_n)}{\psi_{n-1}(\mathcal{S}_{n-1})} = \phi_n(\mathcal{X}_n)\frac{\sum_{\mathcal{X}_{n-1}\backslash S_{n-1}}\phi_{n-1}(\mathcal{X}_{n-1})}{\psi_{n-1}(\mathcal{S}_{n-1})} = \phi'_n(\mathcal{X}_n).$$

Hence, the updated last clique contains $p(X_n)$.

(3.) Lets now reverse the elimination schedule. We have:

$$p(\mathcal{X}) = \frac{\phi'_n(\mathcal{X}_n)\prod_{c\leq n-1}\phi'_c(\mathcal{X}_c)}{\psi'_{n-1}(S_{n-1})\prod_{s\leq n-2}\psi'_s(\mathcal{S}_s)} = \frac{\phi'_n(\mathcal{X}_n)\phi'_{n-1}(\mathcal{X}_{n-1})\prod_{c\leq n-2}\phi'_c(\mathcal{X}_c)}{\psi'_{n-1}(S_{n-1})\prod_{s\leq n-2}\psi'_s(\mathcal{S}_s)}.$$

Consider absorbing information from neighbouring X_n to X_{n-1} using the updated potentials $\phi'_n(X_n)$ and $\psi'_{n-1}(S_{n-1})$ (we know these from part (2.)). We define a new separator for this purpose,

$$\psi''_{n-1}(\mathcal{S}_{n-1}) = \sum_{\mathcal{X}_n\backslash\mathcal{S}_{n-1}}\phi'_n(\mathcal{X}_n) = \sum_{\mathcal{X}_n\backslash\mathcal{S}_{n-1}}\frac{\phi_n(\mathcal{X}_n)\psi'_{n-1}(S_{n-1})}{\psi_{n-1}(S_{n-1})},$$

where the separator is numbered by the lower of two values, $\min(n-1, n)$. We can then refine a new potential for clique n's neighbour $n-1$ given by:

$$\phi'_{n-1}(\mathcal{X}_{n-1}) = \phi_{n-1}(\mathcal{X}_{n-1})\frac{\psi''_{n-1}(\mathcal{S}_{n-1})}{\psi'_{n-1}(\mathcal{S}_{n-1})} = \frac{\phi_{n-1}(X_{n-1})\sum_{X_n\backslash S_{n-1}}\phi_n(\mathcal{X}_n)\psi'_{n-1}(S_{n-1})/\psi_{n-1}(S_{n-1})}{\psi'_{n-1}(S_{n-1})}$$

$$= \phi_{n-1}(\mathcal{X}_{n-1})\frac{\sum_{\mathcal{X}_n\backslash S_{n-1}}\phi_n(\mathcal{X}_n)}{\psi_{n-1}(\mathcal{S}_{n-1})}.$$

[1]In a junction tree, a given node can send exactly one message to one neighbour in a given elimination schedule. Also, this message may only be sent when such node has received a message from each of its other neighbours. The point is, each clique when eliminated will have to be extremal to not break the tree, passing a message to only one neighbour. Given a well defined update ordering, which clique is a neighbour of which should not matter in our computation (its only notation). The marginal information of each clique will eventually be transmitted to the last one.

The last expression is equivalent to computing the marginal $p(X_{n-1})$, since:

$$p(\mathcal{X}_{n-1}) = \sum_{\mathcal{X}_n \setminus \mathcal{S}_{n-1}} p(\mathcal{X}_{n-1} \cup \mathcal{X}_n) = \sum_{\mathcal{X}_n \setminus \mathcal{S}_{n-1}} \frac{\phi_{n-1}(\mathcal{X}_{n-1})\phi_n(\mathcal{X}_n)}{\psi_{n-1}(\mathcal{S}_{n-1})} = \phi_{n-1}(\mathcal{X}_{n-1}) \frac{\sum_{\mathcal{X}_n \setminus S_{n-1}} \phi_n(\mathcal{X}_n)}{\psi_{n-1}(\mathcal{S}_{n-1})} = \phi'_{n-1}(\mathcal{X}_{n-1})$$

We have seen that updated clique $\phi'_{n-1}(X_{n-1})$ contains the marginal $p(X_{n-1})$ after absorption from X_n to X_{n-1}. Iterating this procedure and eliminating cliques one by one, after absorption from a clique to a neighbour of lesser rank X_j, $\phi'_j(X_j)$ will contain the marginal $p(X_j)$.

(4.) Let (V, W) be an arbitrary link with separator S and corresponding potentials $\phi(V)$ and $\phi(W)$. Such link is defined to be consistent if,

$$\sum_{V \setminus S} \phi(V) = \sum_{W \setminus S} \phi(W).$$

Take for example the link between X_{n-1} and X_n illustrated in parts (2.) and (3.) of the question. This link is consistent, since after message passing between both nodes (firstly from X_{n-1} to X_n and then from X_n to X_{n-1}) we have separators,

$$\psi'_{n-1}(\mathcal{S}_{n-1}) = \sum_{\mathcal{X}_{n-1} \setminus \mathcal{S}_{n-1}} \phi'_{n-1}(\mathcal{X}_{n-1}) \quad and \quad \psi''_{n-1}(\mathcal{S}_{n-1}) = \sum_{\mathcal{X}_n \setminus \mathcal{S}_{n-1}} \phi'_n(\mathcal{X}_n),$$

respectively. Then,

$$\psi''_{n-1}(\mathcal{S}_{n-1}) = \sum_{\mathcal{X}_n \setminus \mathcal{S}_{n-1}} \phi'_n(\mathcal{X}_n) = \sum_{\mathcal{X}_n \setminus \mathcal{S}_{n-1}} \phi_n(\mathcal{X}_n) \frac{\psi'_{n-1}(\mathcal{S}_{n-1})}{\psi_{n-1}(\mathcal{S}_{n-1})} = \frac{\psi'_{n-1}(\mathcal{S}_{n-1})}{\psi_{n-1}(\mathcal{S}_{n-1})} \sum_{X_n \setminus S_{n-1}} \phi_n(X_n),$$

From the definition of a separator in this case, $\sum_{X_n \setminus S_{n-1}} \phi_n(X_n) = \psi_{n-1}(S_{n-1}) \rightarrow$

$$\frac{\psi'_{n-1}(\mathcal{S}_{n-1})}{\psi_{n-1}(\mathcal{S}_{n-1})} \sum_{X_n \setminus S_{n-1}} \phi_n(X_n) = \frac{\psi'_{n-1}(S_{n-1})\psi_{n-1}(S_{n-1})}{\psi_{n-1}(S_{n-1})} = \psi'_{n-1}(S_{n-1}) = \sum_{\mathcal{X}_{n-1} \setminus \mathcal{S}_{n-1}} \phi'_{n-1}(\mathcal{X}_{n-1}).$$

Hence, $\sum_{\mathcal{X}_n \setminus \mathcal{S}_{n-1}} \phi'_n(\mathcal{X}_n) = \sum_{\mathcal{X}_{n-1} \setminus \mathcal{S}_{n-1}} \phi'_{n-1}(\mathcal{X}_{n-1})$, and the link is consistent. This procedure can be extended to all of the elimination process in the junction tree, with,

$$\sum_{\mathcal{X}_j \setminus \mathcal{S}_i} \phi'_j(\mathcal{X}_j) = \sum_{\mathcal{X}_i \setminus \mathcal{S}_i} \phi'_i(\mathcal{X}_i). \tag{2}$$

Then, after updating all cliques according to the forward and backward elimination schedules, any link between two cliques is locally consistent, following (2). Therefore, the junction tree is globally consistent. Additionally, we know the requirement for the propagation of local to global consistency in a clique tree is that it is a junction tree, as the question provides. A junction tree structure ensures that if a variable occurs in two cliques, it must be present in all cliques on any path connecting these cliques.