Graphical Models Assignment 2

Team Members:
- Marika Paraskevopoulou
- Eoghan Flanagan
- Jonathan Smith
- Klaudia Ludwisiak

Work Breakdown:
We approached the questions in groups of two or three, and then came together as a team to brainstorm ideas and compare answers where needed.
The following table shows in green which people took primary responsibility for which questions. It should be emphasised that for the harder questions the entire team contributed ideas.

	Eoghan	Klaudia	Maria	Jonathan
4.15				
5.8				
5.9	partial solution	partial solution		
5.10				
5.11				
5.12				
5.13				
6.7				
6.9				
6.10				
6.12				

Exercise 4.15

1. Write $q_{12}(X_1=0, X_2=0) = y_1$

$q_{12}(X_1=0, X_2=1) = y_2$

$q_{12}(X_1=1, X_2=0) = y_3$

$q_{12}(X_1=1, X_2=1) = y_4$

Since q_{12} is a distribution we must have

$$y_1+y_2+y_3+y_4 = 1 \qquad (*)$$

Write $q_{13}(X_1=0, X_3=0) = w_1$ etc.

and $q_{23}(X_2=0, X_3=0) = z_1$ etc.

Then

$$\sum_{X_2} q_{12}(X_1, X_2) = \sum_{X_3} q_{13}(X_1, X_3)$$

$\Leftrightarrow$

$$y_1+y_2 = w_1+w_2 \qquad (1)$$

$$y_3+y_4 = w_3+w_4 \qquad (2)$$

and similarly

$$y_1+y_3 = z_1+z_2 \qquad (3)$$

$$y_2+y_4 = z_3+z_4 \qquad (4)$$

$$w_1+w_3 = z_1+z_3 \qquad (5)$$

$$w_2+w_4 = z_2+z_4 \qquad (6)$$

There are three conditions of the form (*) and 6 conditions above on twelve variables.

⇒ the equations are of linear $My = c$ form with M 9×12 matrix

Consider $y_1 = w_1 = z_1 = 1$, all other variables equal to 0. This clearly satisfies the conditions

2.

Exercise 4.15 (cont)

Consider the three (seperate) two variable distributions defined by

1) $y_1 = 0, y_2 = 1/2, y_3 = 1/2, y_4 = 0$

2) $w_1 = 0, w_2 = 1/2 \; w_3 = 1/2, w_4 = 0$

3) $z_1 = 0, z_2 = 1/2, z_3 = 1/2, z_4 = 0$

(using the same notation as in the previous section)

It can easily be seen that the one variable marginals are $x_1 = 1/2, x_2 = 1/2$ for both q_{12} and q_{13} and by symmetry all other marginals are consistent

However 1) $\Rightarrow P(X_1 = X_2) = 0$

2) $\Rightarrow P(X_2 = X_3) = 0$

Since X_1, X_2, X_3 are binary this implies $P(X_1 = X_3) = 1$. But 3) above implies $P(X_1 = X_3) = 0$

$\Rightarrow$ there is no joint 3 variable dist. which can simultaneously satisfy the marginals

EXERCISE 5.8

1 Description of the problem and the data

In this particular exercise we deal with the problem of pattern recognition in noisy sequences. Particularly we are asked to decode a 10.000 noisy characters sequence, given two specific dictionary of patterns and find the most likely *clean* sequence. One way to deal with this problem is to use a Hidden Markov Model, where the h_t; *hidden* variables represent the clean sequence and the v_t; *visible* variables represent the observed noisy characters sequence. Below we can see a brief description of the data.

- Observed noisy characters sequence v_t size 10.000, with characters from a-z.
- Two dictionary of patterns (5 names, 7 surnames) see Table 1.

Name	Surname
david (5)	barber (6)
anton (5)	ilsung (6)
fred (4)	fox (3)
jim (3)	chain (5)
barry (5)	fitzwilliam (11)
	quinceadams (11)
	grafvonunterhosen (15)

Table 1: Dictionarry of the two patterns (number of characters of each name/surname).

- Total number of characters in the patterns 81.
- Pr(not started a name)=0.8, Pr(not started a surname)=0.8, Pr(being into a name)=$\frac{1}{25}$=0.04, Pr(being into a surname)=$\frac{1}{35}$=0.0286, Pr(the character we want to generate is chosen correctly)=0.3

2 The procedure

- Define the general states and the total states: number of general states = 2; Not started a name w.p 0.8 and not started a surname w.p 0.8 respectively. Number of total states= 12 patterns + 2 general states.

- Define the hidden states: number of hidden states h_t= 12 patterns + 2 general states (13: not started a name, 14: not started a surname) ; 81 characters + 2 general states.

- Define the transition matrix, $T_{14\times 14}$, which describes the transition of the hidden variables. See Table 2.

0	0	0	0	0	0	0	0	0	0	0	0	0.040	0
0	0	0	0	0	0	0	0	0	0	0	0	0.040	0
0	0	0	0	0	0	0	0	0	0	0	0	0.040	0
0	0	0	0	0	0	0	0	0	0	0	0	0.040	0
0	0	0	0	0	0	0	0	0	0	0	0	0.040	0
0	0	0	0	0	0	0	0	0	0	0	0	0	0.029
0	0	0	0	0	0	0	0	0	0	0	0	0	0.029
0	0	0	0	0	0	0	0	0	0	0	0	0	0.029
0	0	0	0	0	0	0	0	0	0	0	0	0	0.029
0	0	0	0	0	0	0	0	0	0	0	0	0	0.029
0	0	0	0	0	0	0	0	0	0	0	0	0	0.029
0	0	0	0	0	1	1	1	1	1	1	1	0.800	0
1	1	1	1	1	0	0	0	0	0	0	0	0	0.800

Table 2: Transition matrix, T.

The transition matrix simple means that if we are not in a surname state we are in the name state w.p 1 and as we can see the columns add to one.

- Define the emission matrix, $B_{26\times 83}$, which indicates that if for example we are in the state 82: not started a name, any of the different characters of the alphabet have equal probability to be chosen $\frac{1}{26}$ and we are in a pattern state w.p 0.3 (since the correct character we wanted to generate is chosen correctly). The emission matrix can be generated in different ways, and is not unique, but it is up to as which procedure we will follow. As for step two here, define the emission probabilities for the general states 2×28 with -$[i,j]^{th}$ element 0.0385.

- Define the first hidden state as being not started a name.

- Use the patternserchsetup.m to get the transition amd emission distributions; the forward passing hidden markov routine to get the $\log[Pr(h_t, v_1, \ldots, v_T)]$ and the sequence of the log- likelihood $\log[Pr(v_1, \ldots, v_T)]$; and the backward message passing to get the $\log[Pr(v_{t+1}, \ldots, v_T|h_t)]$; then use the HMMviterbi.m routine to get the most likely joint hidden state and the logarithm of of the most likely hidden sequence.

- As final step, after having compute all the above the routine getpattern.m will return us for each input (not started a name, not started a surname) which most likely joint hidden state (clean) sequence correspond.

3 Results

	barber	ilsung	fox	chain	fitzwilliam	quinceadams	grafvonunterhosen
david	6	4	8	11	14	10	11
anton	8	6	4	9	20	11	13
fred	6	5	9	10	12	6	19
jim	8	9	7	18	20	12	19
barry	10	5	10	10	9	8	16

Table 3: Matrix, $M_{i \times j=5 \times 7}$, with the counts of the number of occurrences of the pair (first-name(i), surname(j)) in the final clean sequence.

Acknowledgments

For solving this problem we based; `http://web4.cs.ucl.ac.uk/staff/D.Barber/publications/patternsearch.pdf` (and the corresponding code), and `http://web4.cs.ucl.ac.uk/staff/D.Barber/pmwiki/pmwiki.php?n=Brml.Software` .

observation: [patternstate generalstate]

k:	[0	1]	
j:	[0	1]	
r:	[0	1]	
s:	[0	1]	
e:	[0	1]	
k:	[0	1]	
y:	[0	1]	
d:	[1	0]	david
h:	[1	0]	david
v:	[1	0]	david
i:	[1	0]	david
y:	[1	0]	david
u:	[0	2]	
l:	[0	2]	
c:	[0	2]	
f:	[0	2]	
k:	[0	2]	
q:	[0	2]	
f:	[0	2]	
k:	[0	2]	
y:	[11	0]	quinceadams
g:	[11	0]	quinceadams
j:	[11	0]	quinceadams
b:	[11	0]	quinceadams
c:	[11	0]	quinceadams
e:	[11	0]	quinceadams
g:	[11	0]	quinceadams
p:	[11	0]	quinceadams
a:	[11	0]	quinceadams
m:	[11	0]	quinceadams
f:	[11	0]	quinceadams
o:	[0	1]	
i:	[0	1]	
w:	[0	1]	
w:	[0	1]	
b:	[1	0]	david
g:	[1	0]	david
m:	[1	0]	david
i:	[1	0]	david
d:	[1	0]	david
u:	[0	2]	
r:	[0	2]	
s:	[0	2]	
j:	[0	2]	
g:	[0	2]	
q:	[11	0]	quinceadams
u:	[11	0]	quinceadams
z:	[11	0]	quinceadams
d:	[11	0]	quinceadams
c:	[11	0]	quinceadams
d:	[11	0]	quinceadams
n:	[11	0]	quinceadams
e:	[11	0]	quinceadams

```
function [gamma]=patternsearch(v,phghm,pvgh,ph1,thresh,printime,alpha,
    logklik,beta,delta,logpvhstar)
% gamma: p(h_t|v_{1:T})
if nargin==6
    printtime=varargin{1};
else
    printtime=0;
end
t=cputime;

gamma=(alpha).*(beta);
gamma = bsxfun(@rdivide, gamma, sum(gamma));
if printtime; fprintf(1,'\nHMM marginal inference takes %f seconds',cputime-
    t); end
t=cputime;
if printtime; fprintf(1,'\nHMM viterbi takes %f seconds\n',cputime-t); end
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%EXERCISE%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%5.8%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%BASED ON%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%http://web4.cs.ucl.ac.uk/staff/D.Barber/pmwiki/pmwiki.php?n=Main.Software%
%%%%%%%%%%%%%Searching for patterns in a noisy string/sequence%%%%%%%%

clear all;

import brml.*

% load the data; noisystring
% we generate letters by choosing randomly from a to z
data=load('noisystring.mat');
% data=struct with fields, to view the names;
names = data.noisystring(1:10000);
% 5 different names
% 7 different surnames
% two general states, not start a first name,
% not start a surname

% set the names in the paterns
cpattern{1}='david';
cpattern{2}='anton';
cpattern{3}='fred';
cpattern{4}='jim';
cpattern{5}='barry';

% set the surnames in the paterns
cpattern{6}='barber';
cpattern{7}='ilsung';
cpattern{8}='fox';
cpattern{9}='chain';
cpattern{10}='fitzwilliam';
cpattern{11}='quinceadams';
cpattern{12}='grafvonunterhosen';

% for loop for all names and surnames (12)
% for converting the desirsble names to numders
for p=1:12
    pattern{p}=name2num(cpattern{p});
end

% define the pattern states 1:12, for 1:5->first names
% for 6:12-> surname
% 13->14 and the general states,names,surnames,we
% don't define third state, because after the algorithm
% found the first name+surname if we define a third state it
% will stop

firstname=1:5;
surname=6:12;
patternstate=1:12;
generalstate=13:14;
nstates=generalstate(end);

% make the transistion matrix:tran
% tran will be an 15x15 matrix with tran13x13=0.8
% and tran14x14=0.8 defining the probability of not started
% first name and surname respectivelly
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);
    % a firstname started
    tran(generalstate(2),patternstate(p))=1;
    % a first name ended, move to search for surname
end
tran(generalstate(2),generalstate(2))=0.8;
% not started surname
for p=surname
    tran(patternstate(p),generalstate(2))=0.2/length(surname);
    % a surname started
    tran(generalstate(1),patternstate(p))=1;
    % a surname started, go back and start searching
    % for name again
end

% define the probability the character we want to generate
% is generating correctly (remember the string variable
% consists on 26 letters from a to z)
a=0.3;
CorruptionProb=ones(26,26)*(1-a)/25;
for i=1:26
    CorruptionProb(i,i)=0.3;
end
for g=1:length(generalstate)
    pvgh_general(:,g)=ones(26,1)/26;
    % uniformly at random for general states
end
prior.generalstate=[1 0]';
% set starting prior in general state 1 with certainty

% given now starting prior, general states, transition matrix etc
% use them to fing starting and ending points
% transition and emission distributions etc
[phghm,pvgh,ph1,startpatternIDX,endpatternIDX,generalstateIDX]=
    patternsearchsetup(pattern,patternstate,generalstate,tran,CorruptionProb
    ,pvgh_general,prior);


% our original data are letters generated from a to z (26 letters)
v=char(names)-char('a');
% transform from characters to numbers
% to get the visable observation sequence (v) as a vector
v=double(v)+1;
% for the next step we are going to treat this data as
% a generated hidden markov sample

% we set threshold equal to zero,
% because we want to perform exact computation
thresh=0;

printtime=1;


% delta: most likely state
% loglik: log p(v_{1:T})
% logpvhstar: log p(v_{1:T},h^*_{1:T})  -- log of most likely explanation
```

```
% use the hmm forward message passing to get the probability of hidden
% state at time step t given all the visable until 1:t
% and the siquence of log likelihood fuction of all visable states

[alpha loglik]=HMMforward(v,phghm,ph1,pvgh,thresh);

% use hmm backward for the probability: p(h_t|v_{t+1:T})
beta=HMMbackward(v,phghm,pvgh,thresh);

% use the viterbi algorithm for calculating
% most likely joint hidden state sequence and the logged
% loged probability of the most likely hidden sequence/
% most likely explanation
[delta,logpvhstar]=HMMviterbi(v,phghm,ph1,pvgh,thresh);

% use the patternsearch function for calculating the probability of a hidden
% state at time t given all the visable
[gamma]=patternsearch(v,phghm,pvgh,ph1,thresh,printtime,alpha,loglik,beta,
    delta,logpvhstar);

% finally use getpattern to learn in wich pattern the inputs correspond
[pnum gnum]=getpattern(delta,startpatternIDX,generalstateIDX);


fprintf(1,'\nViterbi sequence corresponds to:\n\n')
fprintf(1,'observation: [patternstate generalstate]')
for i=1:length(v)
    fprintf(1,'\n          %s:         [%d        %d]',num2name(v(i)),pnum(i
        ),gnum(i));
    if pnum(i)>0
        fprintf(1,' %s',cpattern{pnum(i)});
    end
end
fprintf(1,'\n\nlikelihood log p(v_{1:T})=%f\n',loglik);
fprintf(1,'likelihood log p(v_{1:T},h^*_{1:T})=%f\n',logpvhstar);

%find the frequencies
%take out all the zeros
pnum(pnum==0)=[];

sol=zeros(1,length(pnum));
sol(1,1)=pnum(1,1);

%for loop for removing all the doublicated elements in
%the pnum, ie:22222225555<-keep only the 2 and 5
for i=1:length(pnum)-1
    if pnum(1,i+1)~=pnum(1,i)
        sol(1,i)=pnum(1,i+1);
    end
end
 sol(sol==0)=[];

%take out the zeros again
 solM=zeros(round((length(sol))/2),3);

for i=1:2:length(sol)-1
      solM(round(i/2),:)=[sol(i),sol(i+1),0];
end

%count how many times each pear of
```

```
%(name surname has occured)
for i=1:round((length(sol))/2)
    c=0;
    for j=i:round((length(sol))/2)
        if solM(i,:)==solM(j,:)
        c=c+1;
        end
    end
    if  solM(i,3)==0;
    solM(i,3)=c;
    end
end
```

Exercise 5.9

(Discussion is here, Matlab code and output follows after)

Key structural elements of problem

This problem has significant structure, as follows:

- The drunk moves at every timestep, one of the following directions with equal probability:
    - North-East
    - South-East
    - North-West
    - South-West
- The drunk and any number of non-drunks can occupy the same point in the matrix simultaneously.
- If the drunk or a non-drunk move off the edge of the matrix they reappear at a random (uniform) point in the matrix, but not within two rows of the edge.
- Non-drunks continue to move in the same direction with probability 0.99 in each dimension.

Approach

The idea here is to examine the position of the people in the matrix, and compare subsequent timesteps. Due to the difference in movement characteristic between the drunk and the others, as time progresses it should be possible to isolate where the drunk is.

The approach taken was as follows:

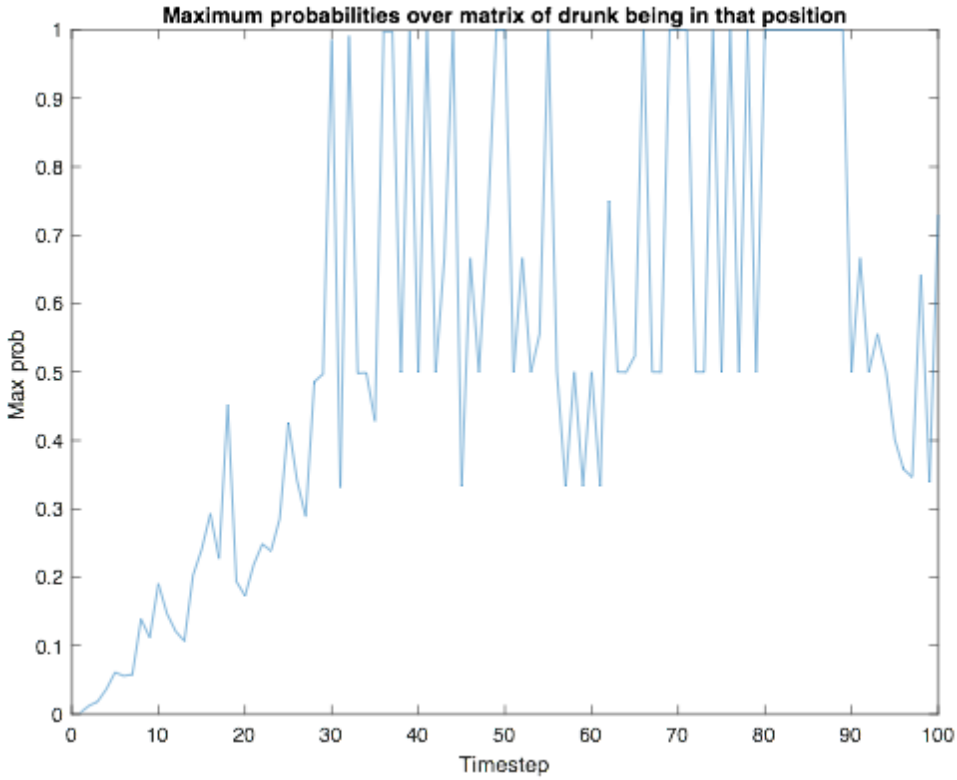
1. At each timestep, assign each occupied square in the matrix a probability of the drunk being in that square.
2. At the first timestep we have no information about where the drunk is, so distribute the probability evenly across the occupied squares.
3. Proceed forwards through time, updating the probabilities based on the occupancy of nearby squares from the previous timestep.
4. Care needs to be taken in the following regions:
    a. 2x2 corner regions
        i. The drunk is only able to enter this region from one (diagonal) direction and has a probability of ¾ of moving out of the matrix in the next time step.
    b. Non corner edge regions
        i. The drunk is only able to enter this region from two diagonal directions and has probability of ½ of moving out of the matrix in the next timestep.
    c. Central region

i. The drunk can move into any given square from four possible squares in the previous timestep.
ii. With every timestep there is a non-zero probability that a given occupied square is occupied due to the random creation of a new person in that square due to someone moving off the edge of the grid.

The Matlab code written deals with each of these regions in turn in the update procedure.

Discussion of Results

The following graph shows, for each timestep, the maximum probability over all squares of the matrix. As expected, after the start (random initialisation of locations) we build confidence of the drunks position.



The sequence of jumps from 1 to 0.5 and back again is where there are suddenly two plausible moves the drunk could have made, so the confidence drops from 1 to 0.5.

If the drunk moves off the matrix I would expect the confidence in his position to be very low. Interestingly that is not the case here, so it seems that the drunk did not move off the edge of the matrix.

The calculated most likely path of the drunk at this point does have some implausible jumps (i.e. more than 2 squares!) in, especially at the start. This must be wrong. This is consistent with the low confidence in the drunk's position at the start due to the random initialisation.

Further Improvements and Ideas

- I tried to then run the same calculation backwards, initialising using the high-confidence results from the forward pass where possible. Interestingly that did not seem to improve the results very much, and in some cases made them worse.

- I have not used the information about the predictability of the movement of the non-drunks. This seems a very significant piece of structure so plausibly would improve the accuracy of the calculation.

- We discussed using the Viterbi algorithm for this question. We concluded that it would arrive at the correct answer, but with an unfeasibly large number of calculations, so did not use it.

Table of Contents

Environment setup 1
Initialisation 1
Looping forwards through data, updating probabilities 1
Identification of most likely path after forward calculation path 4
Output likely path of drunk 5

Environment setup

```
close all
clear all
import brml.*
load drunkproblemX.mat
```

Initialisation

```
% Setup size of probability matrix
Gx = 50;
Gy = 50;
prob_matrix{1} = zeros(Gy,Gx);

% Initialise prob_matrix with a uniform probability of every square
% containing the drunk person
num_cells_occupied = sum(sum(X{1}));

for x = 1:Gx
    for y = 1:Gy
        if X{1}(x,y) == 1
            prob_matrix{1}(x,y) = 1/num_cells_occupied;
        end
    end
end


% Check that probabilities sum to 1
epsilon = 1e-14;
assert(sum(sum(prob_matrix{1})) - 1 < epsilon)
```

Looping forwards through data, updating probabilities

```
for t = 2:100
    % Init size of prob_matrix{t)
    prob_matrix{t} = zeros(Gy,Gx);

    % North-West corner region
```

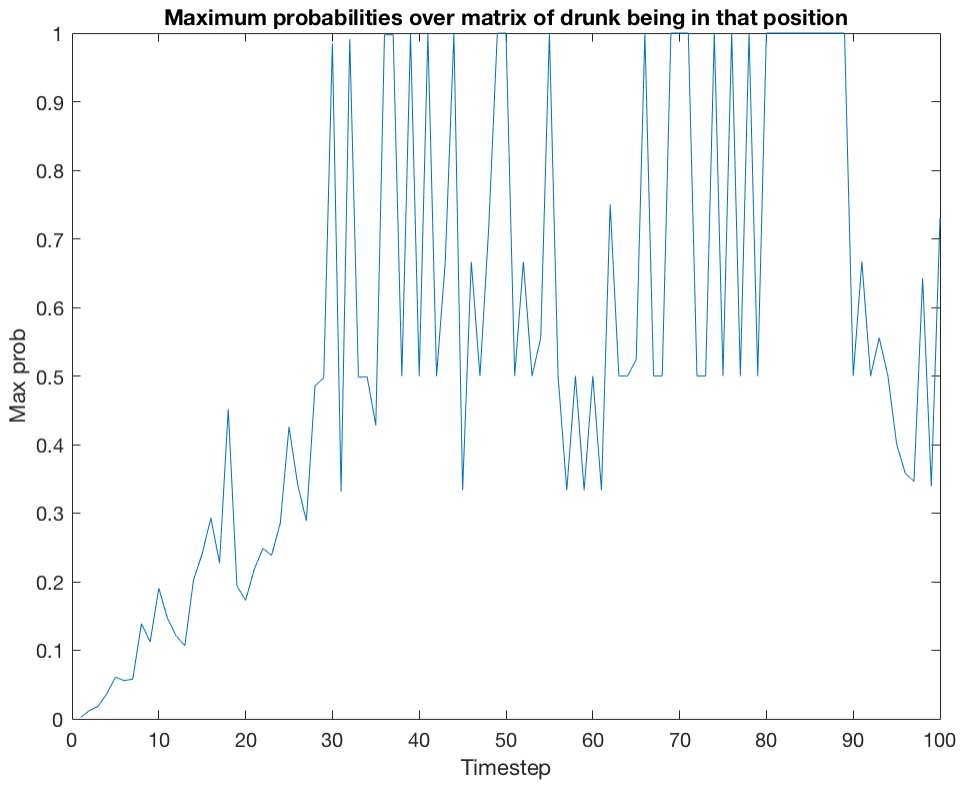
```
for x = 1:2
   for y = 1:2
       if X{t}(x,y) == 1
           prob_matrix{t}(x,y) = prob_matrix{t-1}(x+2,y+2)*0.25;
       end
   end
end

% North-East corner region
for x = 49:50
   for y = 1:2
       if X{t}(x,y) == 1
           prob_matrix{t}(x,y) = prob_matrix{t-1}(x-2,y+2)*0.25;
       end
   end
end

% South-West corner region
for x = 1:2
   for y = 49:50
       if X{t}(x,y) == 1
           prob_matrix{t}(x,y) = prob_matrix{t-1}(x+2,y-2)*0.25;
       end
   end
end


% South-East corner region
for x = 49:50
   for y = 49:50
       if X{t}(x,y) == 1
           prob_matrix{t}(x,y) = prob_matrix{t-1}(x-2,y-2)*0.25;
       end
   end
end


% North edge region
for x = 3:48
   for y = 1:2
       if X{t}(x,y) == 1
           prob_matrix{t}(x,y) = ...
               prob_matrix{t-1}(x+2,y+2)*0.25 ...
             + prob_matrix{t-1}(x-2,y+2)*0.25;
       end
   end
end


% East edge region
for x = 49:50
   for y = 3:48
       if X{t}(x,y) == 1
           prob_matrix{t}(x,y) = ...
```

```
                prob_matrix{t-1}(x-2,y+2)*0.25 ...
              + prob_matrix{t-1}(x-2,y-2)*0.25;
        end
    end
end

% South edge region
for x = 3:48
    for y = 49:50
        if X{t}(x,y) == 1
            prob_matrix{t}(x,y) = ...
                prob_matrix{t-1}(x+2,y-2)*0.25 ...
              + prob_matrix{t-1}(x-2,y-2)*0.25;
        end
    end
end

% West edge region
for x = 1:2
    for y = 3:48
        if X{t}(x,y) == 1
            prob_matrix{t}(x,y) = ...
                prob_matrix{t-1}(x+2,y+2)*0.25 ...
              + prob_matrix{t-1}(x+2,y-2)*0.25;
        end
    end
end

% Calculate the probability that the drunk moved off the edge
% of the matrix in the last time step
% First sum the probabilities that he was in each of the regions
% at the previous timestep
prob_corners = sum(sum(prob_matrix{t-1}(1:2,1:2))) ...
             + sum(sum(prob_matrix{t-1}(49:50,1:2))) ...
             + sum(sum(prob_matrix{t-1}(1:2,49:50))) ...
             + sum(sum(prob_matrix{t-1}(49:50,49:50)));

prob_edges = sum(sum(prob_matrix{t-1}(3:48,1:2))) ...
           + sum(sum(prob_matrix{t-1}(3:48,49:50))) ...
           + sum(sum(prob_matrix{t-1}(1:2,3:48))) ...
           + sum(sum(prob_matrix{t-1}(49:50,3:48)));

% Calculate the probability that the drunk moved off the edge of
% the matrix at the last timestep
prob_jumped = prob_corners * 0.25 + prob_edges * 0.5;

% Calculate the number of squares in the central region that are
% filled in the current timestep
num_central = sum(sum(X{t}(3:48,3:48)));

% Central region
for x = 3:48
    for y = 3:48
        if X{t}(x,y) == 1
```

```
                prob_matrix{t}(x,y) = ...
                    prob_matrix{t-1}(x+2,y+2)*0.25 ...
                  + prob_matrix{t-1}(x+2,y-2)*0.25 ...
                  + prob_matrix{t-1}(x-2,y+2)*0.25 ...
                  + prob_matrix{t-1}(x-2,y-2)*0.25 ...
                  + prob_jumped/num_central;
            end
        end
    end


    % Normalise the prob_matrix to make sure it still sums to 1
    prob_matrix{t} = prob_matrix{t} / sum(sum(prob_matrix{t}));

end
```

Identification of most likely path after forward calculation path

```
for t = 1:100
    highest_prob = -1;
    highest_prob_x = -1;
    highest_prob_y = -1;

    for x = 1:50
        for y = 1:50
            if prob_matrix{t}(x,y) > highest_prob
                % New best probability found.  Update record.
                highest_prob = prob_matrix{t}(x,y);
                highest_prob_x = x;
                highest_prob_y = y;
            end
        end
    end

    % store the highest value probability and associated x,y
    maxprobs_forwards(t) = highest_prob;
    maxprobs_forwards_x(t) = highest_prob_x;
    maxprobs_forwards_y(t) = highest_prob_y;
end

plot(maxprobs_forwards)
title('Maximum probabilities over matrix of drunk being in that
 position')
xlabel('Timestep') % x-axis label
ylabel('Max prob') % y-axis label
```



Output likely path of drunk

```
disp('Calculated likliest path of drunk is:')
for t = 1:100
    disp(['t = ' num2str(t) '  x = '
 num2str(maxprobs_forwards_x(t)) ...
        '  y = ' num2str(maxprobs_forwards_y(t))])
end
```

```
Calculated likliest path of drunk is:
t = 1  x = 3  y = 4
t = 2  x = 34  y = 36
t = 3  x = 29  y = 23
t = 4  x = 27  y = 21
t = 5  x = 27  y = 25
t = 6  x = 28  y = 24
t = 7  x = 26  y = 30
t = 8  x = 24  y = 28
t = 9  x = 20  y = 28
t = 10  x = 22  y = 30
t = 11  x = 18  y = 26
t = 12  x = 16  y = 24
t = 13  x = 39  y = 24
t = 14  x = 16  y = 24
t = 15  x = 18  y = 22
t = 16  x = 16  y = 24
```

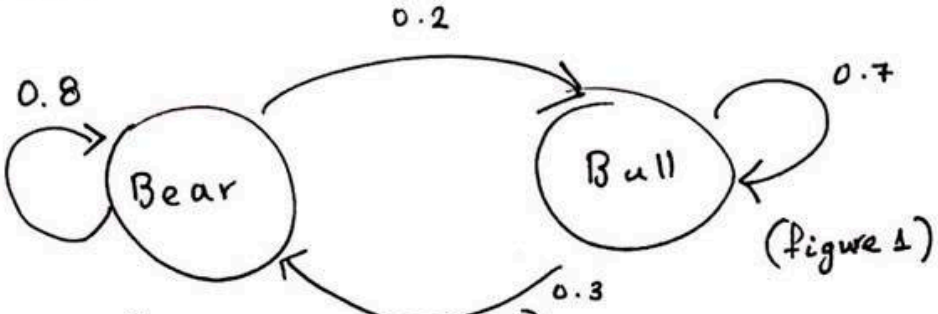
```
t = 17  x = 18  y = 26
t = 18  x = 20  y = 24
t = 19  x = 39  y = 24
t = 20  x = 37  y = 26
t = 21  x = 39  y = 28
t = 22  x = 37  y = 26
t = 23  x = 35  y = 28
t = 24  x = 37  y = 26
t = 25  x = 35  y = 24
t = 26  x = 37  y = 22
t = 27  x = 35  y = 32
t = 28  x = 37  y = 22
t = 29  x = 35  y = 24
t = 30  x = 41  y = 26
t = 31  x = 39  y = 28
t = 32  x = 41  y = 26
t = 33  x = 43  y = 24
t = 34  x = 41  y = 26
t = 35  x = 43  y = 24
t = 36  x = 37  y = 30
t = 37  x = 39  y = 32
t = 38  x = 41  y = 30
t = 39  x = 43  y = 32
t = 40  x = 41  y = 30
t = 41  x = 43  y = 32
t = 42  x = 41  y = 30
t = 43  x = 43  y = 32
t = 44  x = 41  y = 34
t = 45  x = 39  y = 32
t = 46  x = 37  y = 34
t = 47  x = 39  y = 32
t = 48  x = 41  y = 34
t = 49  x = 39  y = 32
t = 50  x = 41  y = 30
t = 51  x = 39  y = 28
t = 52  x = 41  y = 30
t = 53  x = 39  y = 32
t = 54  x = 41  y = 30
t = 55  x = 43  y = 36
t = 56  x = 45  y = 34
t = 57  x = 39  y = 36
t = 58  x = 45  y = 34
t = 59  x = 43  y = 36
t = 60  x = 41  y = 42
t = 61  x = 39  y = 44
t = 62  x = 37  y = 42
t = 63  x = 35  y = 40
t = 64  x = 37  y = 38
t = 65  x = 35  y = 36
t = 66  x = 33  y = 34
t = 67  x = 31  y = 36
t = 68  x = 33  y = 34
t = 69  x = 35  y = 36
t = 70  x = 37  y = 38
```

```
t = 71  x = 35  y = 36
t = 72  x = 37  y = 34
t = 73  x = 35  y = 40
t = 74  x = 33  y = 38
t = 75  x = 31  y = 36
t = 76  x = 33  y = 38
t = 77  x = 31  y = 40
t = 78  x = 29  y = 42
t = 79  x = 31  y = 44
t = 80  x = 29  y = 42
t = 81  x = 31  y = 40
t = 82  x = 33  y = 42
t = 83  x = 31  y = 44
t = 84  x = 29  y = 46
t = 85  x = 27  y = 48
t = 86  x = 25  y = 46
t = 87  x = 27  y = 48
t = 88  x = 25  y = 46
t = 89  x = 27  y = 44
t = 90  x = 25  y = 46
t = 91  x = 27  y = 48
t = 92  x = 29  y = 46
t = 93  x = 27  y = 48
t = 94  x = 25  y = 46
t = 95  x = 27  y = 48
t = 96  x = 25  y = 46
t = 97  x = 27  y = 48
t = 98  x = 29  y = 46
t = 99  x = 19  y = 44
t = 100  x = 21  y = 46
```

Published with MATLAB® R2016b

5.10 BearBull,
Outline of approach is given below and followed by annotated code and MATLAB output of final results

- Price of an asset through $T = 200$ timepoints
- $1 \leq$ price ≤ 100
- Market



(figure 1)

Bear state: $\text{Price}_{t-1} \to \text{Price}_t$ w.p $P(\text{bear})$

Bull state: $\text{Price}_{t-1} \to \text{Price}_t$ w.p $P(\text{Bull})$

- from (figure 1) we can define the stochastic matrix

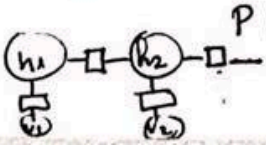
$$\begin{matrix} & \text{Bear} & \text{Bull} \\ \text{Bear} & 0.8 & 0.3 \\ \text{Bull} & 0.2 & 0.7 \end{matrix}, \quad \sum_i P_{ij} = 1. \quad \text{(simple transition matrix)}$$

- from $T = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix}$ we can find the probabilities $P_1 = P(\text{Bull})$ and $P_2 = P(\text{Bear})$ by marginalizing

hence
$$\left.\begin{aligned} 0.8\,P_1 + 0.3\,P_2 &= P_1 \\ 0.2\,P_1 + 0.7\,P_2 &= P_2 \end{aligned}\right\} \Rightarrow \begin{aligned} P_1 &= \frac{9}{15} \Rightarrow P(\text{Bull}) = \frac{9}{15} \\ P_2 &= \frac{6}{15} \Rightarrow P(\text{Bear}) = \frac{6}{15} \end{aligned}$$

- Suppose a hidden markov problem where in our case the visable variable are the ~~prices~~ and the hidden the Bear, Bull, states.
- Assume that we start from Bear $\Rightarrow P(h_1 = \text{Bear}) = \frac{9}{15}$
  hence $P(h_1, v_1) = P(v_1|h_1)\,P(h_1)$
- Thus the likelihood is then given by

$$P(v_1, \ldots, v_T; h_1, \ldots, h_T) = P(h_1)\,P(v_1|h_1) \prod_{t=2}^{200} P(v_t|h_t)\,P(h_t|h_{t-1})$$

- 

Define factors; $\psi(h_t, v_t) = P(v_t|h_t)$ and $\phi(h_t, h_{t+1}) = P(h_{t+1}|h_t)$

```
%BearBull

close all;
clear all;

%Exploratory
load('BearBullproblem.mat')
%pbear pbull cols sum to one
%thus, stochastic matrix where each col is a probability distribution

%occasionally price does not change from t to t+1
%this imples change of state

Mtrans=[0.8, 0.3; 0.2, 0.7]; %rows sum to one

%initialise empty array of conditional price probability
Prob=zeros(size(p)); %where P(1,1)=P(1), P(1,2) denotes P(2|1) and
 P(1,3) denotes P(3|2,1) etc.

%Assume that at tilmestep 1, the market is uniformly in either a bear
 or
%bull state and also that the price distribution at
Prob(1,1)=(9/15)*pbear(1,1);
%for bull: (6/15)*pbull(1,1) %Inspection and maths indicate that the
 state
%we initialise to has no impact on final solution

for t=2:length(p) %2:200
    if p(1,t)==p(1,t-1) %change state
       Prob(1,t)=Prob(1,t-1)*((Mtrans(1,2)*pbear(p(1,t),p(1,t-1)))...
           +(Mtrans(2,1)*pbull(p(1,t),p(1,t-1))));
    else  %stay in same state
        Prob(1,t)=Prob(1,t-1)*((Mtrans(1,1)*pbear(p(1,t),p(1,t-1)))...
            +(Mtrans(2,2)*pbull(p(1,t),p(1,t-1))));
    end
end

%FIND P(201|200,199....):
%setup:

%initialise vector to write probabilty distribution at T=201
Prob201=zeros(1,size(pbear,1));

Prob200=Prob(1,length(Prob)); %equiv. to P(200|199,198....)
p200=p(1,length(p)); %%price at last timestep t=200

%calculate:
for i=2:size(pbear,1)
   if p(1,length(p))==i
    Prob201(1,i)=Prob200*((Mtrans(2,1)*pbull(i,p200))...
            +(Mtrans(1,2)*pbear(i,p200)));
   else  %saty in same state
```

```
    Prob201(1,i)=Prob200*(Mtrans(2,2)*pbull(i,p200))...
            +(Mtrans(1,1)*pbear(i,p200));
   end
    %also assign proababilty for praice=1 (know proce at t=200 is 58
 so use
    %formula for when new price is not equal old price)
  Prob201(1,1)=Prob200*(Mtrans(2,2)*pbull(1,p200))...
        +(Mtrans(1,1)*pbear(1,p200));
end

%final output:

pricerange=linspace(1,100);

%expected price at time 201: P(pice(T+1)|price(T:1)) is the mean of
 the
%distribution on prices ranging from 1 to 100.
%normalise probability distribution:
normalisedProb201=(1/sum(Prob201))*Prob201;
distrib=(normalisedProb201.*pricerange);
%P(pice(T+1)=Xi|price(T:1)) where i ranges from 1 to 100
p201bar=sum(distrib);

%expected new price is aprox.52.7 units

%expected gain is the difference between expected price at T=201 and
%observed price at T=200:
ExpGain=p201bar-p200;

%expected gain is negative at aprox.-5.3 units

%find variance
v=sum((((distrib)-p201bar).^2)/200);

%find standard deviation
StandDev=sqrt(v);

%standard deviation is high at almost 37 units, implying uncertainty
 in
%the future price.  Below a check is performed where the historical
 price
%fluctuations are examined and it is found that they range as much as
 22
%units per day, therefore this standard deviation is not entirely
%unreasonable.

difference=[];
for i=2:200
difference(1,i)=p(1,i)-p(1,i-1);
end

%OUTPUT
disp('P(pice(T+1)=Xi|price(T:1)) where i ranges from 1 to 100, is
 given by the following: \n')
```

```
disp(distrib(1,:));
fprintf('The expected gain is negative and amounts to %.2f units
\n',ExpGain);
fprintf('The standard deviation in price at T=201 is %.2f units
\n',StandDev);
```

```
P(pice(T+1)=Xi|price(T:1)) where i ranges from 1 to 100, is given by
 the following: \n
  Columns 1 through 7

    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000

  Columns 8 through 14

    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000

  Columns 15 through 21

    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000

  Columns 22 through 28

    0.0001    0.0002    0.0003    0.0006    0.0010    0.0018    0.0032

  Columns 29 through 35

    0.0054    0.0089    0.0145    0.0230    0.0357    0.0543    0.0809

  Columns 36 through 42

    0.1181    0.1688    0.2364    0.3242    0.4356    0.5732    0.7391

  Columns 43 through 49

    0.9335    1.1551    1.4002    1.6630    1.9350    2.2060    2.4640

  Columns 50 through 56

    2.6966    2.8916    3.0381    3.1276    3.1549    3.1184    3.0202

  Columns 57 through 63

    2.8663    0.0000    2.4291    2.1691    1.8981    1.6276    1.3677

  Columns 64 through 70

    1.1262    0.9088    0.7187    0.5569    0.4229    0.3148    0.2296

  Columns 71 through 77

    0.1641    0.1149    0.0789    0.0531    0.0350    0.0226    0.0143

  Columns 78 through 84
```

```
    0.0089    0.0054    0.0032    0.0019    0.0011    0.0006    0.0003

  Columns 85 through 91

    0.0002    0.0001    0.0000    0.0000    0.0000    0.0000    0.0000

  Columns 92 through 98

    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000    0.0000

  Columns 99 through 100

    0.0000    0.0000

The expected gain is negative and amounts to -5.23 units
The standard deviation in price at T=201 is 36.95 units
```

Published with MATLAB® R2016b

5.11

1) The equations of motion are independent for each dimension so can be treated separately for this section.

Working in dimension 1:

$$x_{102} = x_{101} + \delta v_{101}$$

$$= (x_{100} + \delta v_{100}) + \delta v_{101}$$

where I have substituted using the given equation $x_{t+1} = x_t + \delta v_t$

Continuing in this way we reach

$$x_{102} = \delta \sum_{i=2}^{101} v_i$$

where i starts at 2 because $v_1 = 0$ from question.

Similarly, using the given equation $v_{t+1} = v_t + \delta a_t$ and performing successive substitutions:

$$v_i = \delta \sum_{j=1}^{i-1} a_j$$

Combining the above two equations gives

$$x_{102} = \delta^2 \sum_{i=2}^{101} \sum_{j=1}^{i-1} a_j$$

for each dimension $i \in \{1, 2, 3\}$

2) To start with, consider a single dimension. (combine the three dimensions later when calculating total fuel).

eg dimension 1. $x_{102} = 4.71$

because $\delta = 0.1$ we have

$$\frac{4.71}{\delta^2} = 471 = \sum_{i=2}^{101} \sum_{j=1}^{i-1} a_j$$

So the solutions for this dimension are the set of a_j that fulfil this equation.

First I will search for a solution where all the a_j are either 0 or 1, because (in this single dimension case) if a solution exists it will use less fuel than a solution that has some $a_j = -1$.

Visualising the summation:

as a table

$$471 = \sum_{i=2}^{101} \sum_{j=1}^{i-1} a_j$$

i \ j	1	2	3	4	...	100
2	a_1					
3	a_1	a_2				
4	a_1	a_2	a_3			
5	a_1	a_2	a_3	a_4		
...	...	...	...	...	⋱	
101	a_1	a_2	a_3	a_4	...	a_{100}

as expected, a_1 has the largest impact on x_{102} as it is the earliest acceleration.

in total there are $100 \times a_1$, $99 \times a_2$, ... , $1 \times a_{100}$.

471 can be written:

$$471 = \underset{\downarrow}{100} + \underset{\downarrow}{99} + \underset{\downarrow}{98} + \underset{\downarrow}{97} + \underset{\downarrow}{77}$$

$$a_1 \quad a_2 \quad a_3 \quad a_4 \quad a_{24}$$

So one solution is $a_1, a_2, a_3, a_4, a_{24} = 1$

all other $a_j = 0$.

This is not a unique solution. All sets of 5 a_j that sum to 471 are equivalent at this stage

Performing the same calculation in the other dimensions gives similar results, shown below:

Dimension	Total	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_{14}	a_{24}	a_{83}
1	471	1	1	1	1	0	0	0	0	0	1	0
2	-697	-1	-1	-1	-1	-1	-1	-1	0	0	0	-1
3	859	1	1	1	1	1	1	1	1	1	0	0
	↑ already divided by δ^2											

Now we should consider the 'overlap' between the dimensions. If we can we should do accelerations at the same time, as accelerations in 2 dimensions only have cost $\sqrt{2}$ and in 3 dimensions only have cost $\sqrt{3}$ (ass opposed to 2 and 3).

By inspection, by 'sliding' dimension 1's accelerations, the acceleration at time $t=24$ can be moved to occur at $t=14$ instead, where it will overlap with an acceleration in dimension 3, while maintaining existing overlaps.
By inspection no further overlaps are possible without introducing more accelerations (and increasing cost)

So the final set of accelerations is as follows:

Dimension	Total	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8	a_{14}	a_{24}	a_{83}
1	471	0	0	1	1	0	1	1	0	1	0	0
2	-697	-1	-1	-1	-1	-1	-1	-1	0	0	0	-1
3	859	1	1	1	1	1	1	1	1	1	0	0

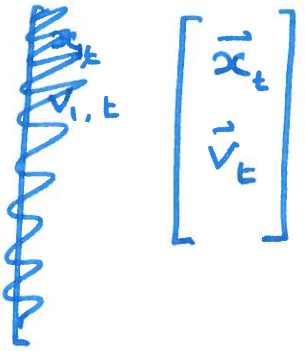
This gives a total cost of:

$$4 \times \sqrt{3} \quad (a_3, a_4, a_6, a_7)$$
$$+ 4 \times \sqrt{2} \quad (a_1, a_2, a_5, a_{14})$$
$$+ 2 \times 1 \quad (a_3, a_{83})$$
$$= \underline{14.59} \text{ units of fuel.}$$

---

3 This calculation can be approached with a "message passing" approach, with polynomial efficiency.

define state at time t to be $\vec{S}_t = \begin{bmatrix} \vec{x}_t \\ \vec{v}_t \end{bmatrix}$



$$\vec{S}_{\bullet 1} = \vec{\emptyset}$$

$$S_{102} = \begin{bmatrix} 4.71 \\ -6.97 \\ 8.59 \\ x \\ x \\ x \end{bmatrix}$$

where x = don't care.

The total cost can be expressed as the following:

$$\min_{S_2, \ldots, S_{102}} \left[C(S_2 \mid S_1 = \vec{\emptyset}) + C(S_3 \mid S_2) + \ldots + C(S_{102} \mid S_{101}) \right]$$

where $C(S_i, S_{i-1})$ is the fuel cost of transitioning from state $i-1$ to i.

You can then consider

$$\min_{s_2} \left[C(s_2 | s_1 = \vec{\emptyset}) + C(s_3 | s_2) \right] = \gamma_{s_2 \to s_3}(s_3)$$

ie a message to s_3, and proceed to the end.
effectively this is a "minimum running total" for each state as you proceed through time.

Computational efficiency

Without the message passing approach exponential computational steps are required.

With the ~~comp~~ message passing approach we have the following complexity for a given final state s_{102} (ie both $\vec{x}$ and $\vec{v}$ given).

- at timestep t there could be $\sim t^2$ different x values in each dimension reached.
- likewise at timestep t there could be $\sim t$ different v values reached.
- so for 3 dimensions this gives a bound of

$$(t^2 . t)^3 = t^9 \text{ states.}$$

So for final timestep T, $\sum_{t=1}^{T} t^9 \sim T^{10}$ computations.

this is for one final state with given $\vec{x}$ and $\vec{v}$ values.

In the question only the final $\vec{x}$ value is important,

so with this message-passing approach you would

need to either – check solution at all final speeds (additional T^3 time)

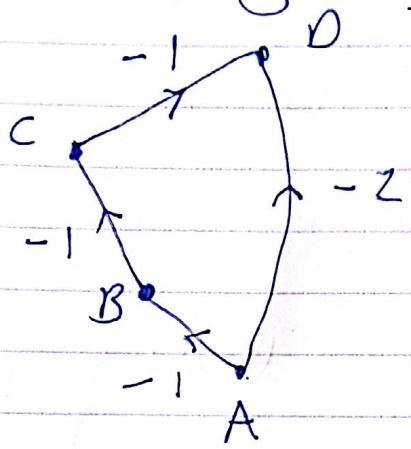
– Use a single-source, multiple sink variant.

In conclusion, a calculation in polynomial time is possible, eg T^{13}. Whether this is considered "efficient" is subjective. I would say no as 13 is a high power. Eg this question would need $100^{13} \simeq 10^{26}$ computations.

The reason more efficient computation is not possible is that you have to store position and velocity at all times, hugely increasing the number of possible states.

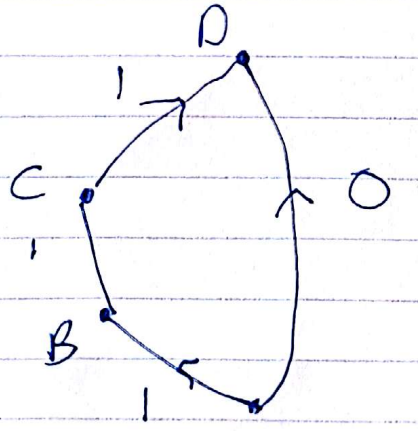
Exercise 5.12

Consider the following graph with associated (negative) edge weights



The path from A to D which minimizes the sum of edge weights is clearly A-B-C-D with a total score of -3

Following your friend's suggestion would result in the following set of edge weights



It can be seen that the shortest path would now be A-D, which is not the correct path for the orginal problem.

```
%%Question 5.13

%%APPROACH OUTLINE:
%
%1.Input and data exploration:
%(i)A is sparse logical matrix defining if travel is possible between
%   planets i and j. This is a triangular matrix with zeros on the
 diagonal
%   implying that  travel is only possible in one direction (i.e. from
%   node 1 can go anywhere, but from node 2 cannot go back to node 1)
%(ii)t is a vector of tax collectable on each planet (between 0 and 1)
%(iii)x contains x,y,z coordinates of every planet (between 0 and 1)
%
%2.Define Question:
%Provided with a cost function of the form: C=fun(distance,tax),where:
%distance- Euclidean distance (3 dimensions) between plant i and j
%tax- amount of tax collected.
%the objective is to minimise C whilst going from planet 1 to 1725
%this is a type of most probable path problem
%
%3.Solution:
%(i)Compute Cost matrix using inputs provided in SimoHurrta.mat
%   note that the cost function is defined in such a way that a
 negative
%   Cost(i,j) value means that more tax is collected than the cost of
%   traveling from between i and j, therefore the most negative value
 is
%   the one to be selected.
%(ii)use mostprobablepath.m to calculate the path which minimises cost
 of
%    travel adjusted over profits from collected tax.
%    function syntax:
%     [optpath
 logprob]=mostprobablepath(logtransition,startstate,endstate)
%       where:
%       logtransition: is the logarithm of the transition matrix (A)
%         However as done in demoShortestPath.m in this case it can be
%         taken as the negative of the inverse of the Cost matrix
 where
%         this needs to have zeros on diagonal. Taking the negative of
 the
%         cost matrix ensures a maximisation porblem is turned into a
%         minimisation problem.
%       startstate: is the index of the start point
%       endstate: is the index of the end point
%%Background on mostprobablepath.m
%messages are recursively passed O(N^2) times where N is vertical size
 A
%this is done for time steps of length(start to end state) (here 1725)
%this is equivalent to computing the min cost for going from planet 1
 to 1725
```

```
%in 1,2,...1725 steps and then picking the minimum from all these
 paths.
```

SOLUTION

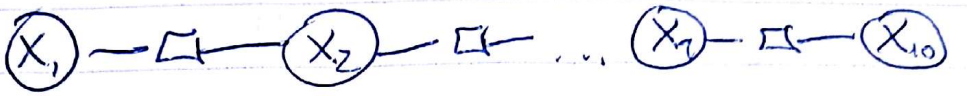
```
%SET-UP:

clear all;
close all;

load('SimoHurrta.mat')

Cost=zeros(size(A));

%Compute Cost matrix

for j= 1:size(x,2)%cols
    for i= 1:size(x,2)%rows
       if A(i,j)==1
        dist=sqrt(sum((x(:,i)-x(:,j)).^2)); %Euclidian distance
        Cost(i,j)=dist-t(1,j);              %subtract tax
       else
           Cost(i,j)=inf;
       end
    end
end

[optpath, weight]=mostprobablepath(-Cost',1,1725);

%RESULTS:
%optimum path is stored in vector optpath
%weight is the sum of the cost function when traveling on the optimum
 path,
%should take the nagative of this value as as we have used the
 negative of
%matrix C. Thus, teh result implies that the tax man makes money
%(i.e. tax collected is more than the travel expense on this trip)

fprintf('shortest path is via %i planets and has cost of %g implying
 the tax man makes money on his trip\n',length(optpath),-weight);
```

```
shortest path is via 707 planets and has cost of -209.65 implying the
 tax man makes money on his trip
```

Published with MATLAB® R2016b

Exercise 6.7

The cluster variables $X_1 \dots X_{10}$ are related by the factor graph shown



And we have

$$Z = \sum_{X_1 \dots X_{10}} \psi(X_1, X_2)\, \psi(X_2, X_3) \dots \psi(X_9, X_{10})$$

$$= \sum_{X_1 \dots X_9} \psi(X_1, X_2) \dots (X_8, X_9) \sum_{X_{10}} \psi(X_9, X_{10})$$

To evaluate this sum using message-passing we need to have messages

$$\mu_{10 \to 9} = \sum_{X_{10}} \psi(X_9, X_{10})$$

Each of the cluster variables has 2^0 states, therefore computing this message would require 2^{30} operations

Thus in total, the computation of Z would require

$$9 \times 2^{30} \text{ computations}$$

Exercise 6.7 cont.

$$Z = \sum_{\substack{\text{all binary} \\ \text{lattices}}} \prod_{\substack{\text{all elements} \\ \text{in lattice}}} e^{I(x_i - x_j)}$$

Note that because it's specified that $i > j$ each pair of ~~no~~ (neighbour) vertices contributes only once. So this is actually a sum over the edges of the (binary) lattices

$$Z = \sum_{\substack{\text{all binary} \\ \text{lattices}}} \prod_{\substack{\text{all edges} \\ \text{in lattice}}} e^{I(x_i - x_j)}$$

The edges of any binary-~~metric~~ valued lattice form two sets - those between equally-valued vertices and those between unequal vertices

$$\Rightarrow Z = \sum_{\substack{\text{all binary} \\ \text{lattices } B}} e^{K(B)}$$

where $K(B)$ is the number of edges in lattice B between distinct vertices

$K(B)$ must be even, and in our example ($n = 10$) we have $(10-1)^2 = 81$ edges

Therefore $Z = 2\left[1 + \binom{81}{2}\epsilon^2 + \binom{81}{4}\epsilon^4 + \ldots\right]$

$= 2\left[(1+\epsilon)^{81} + (1-\epsilon)^{81}\right]$

$\log Z \simeq \log 2 + 81 \log(1+\epsilon)$

since $(1-\epsilon)^{81} \ll (1+\epsilon)^{81}$

Exercise 6.9

Part 1:

See following Matlab output, section 'Part 1'. Junction tree is constructed in Matlab script and is too large to display in this report.

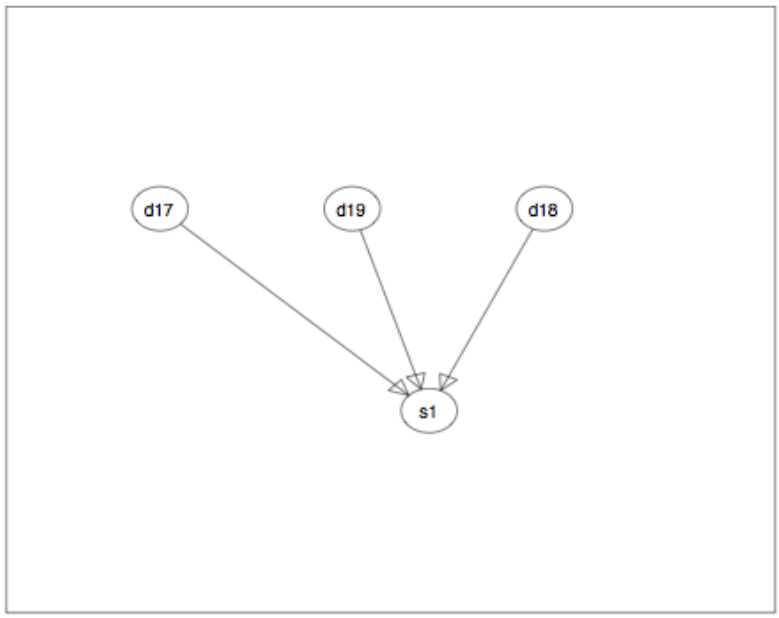
Part 2:

Because this is a valid belief network, with significant structure (all the diseases are only parents, all the symptoms are only children) we can use this structure to make the calculation of marginals $p(s_i = 1)$ more efficient, with the following steps.

Take, for example, calculating $p(s_1 = 1)$, where symptom s_1 only depends on diseases d_{17}, d_{18} and d_{19}.

- Marginalise over all other symptoms. Because they are only children and this is a valid belief network, they just 'disappear'.
- Marginalise over all other diseases. Because they are no longer connected to any of the variables of interest, they also just 'disappear'.

So you can calculate $p(s_1 = 1)$ by only considering s_1 and diseases d_{17}, d_{18} and d_{19}, and just ignoring the other variables. The resulting belief network is simply as follows. Because this is now such a small belief network, calculating the joint probability requires very little computation. This approach is taken in the Matlab script, section 'Part 2'.



Part 3

See Matlab output, section 'Part 3'.

Table of Contents

Part 1 1
Part 2 2
Part 3 4

Part 1

```
import brml.*
load diseaseNet.mat;

% Construct a junction tree
[jtpot jtsep infostruct]=jtree(pot);

% Use the junction to compute all the marginals of the symptoms, p(si
 = 1).
% Do the full round of absorption
jtpot=absorption(jtpot,jtsep,infostruct);

% Loop through all the symptoms, calculate the marginal probability of
% each symptom.
disp(' ') % new line
disp('Probabilities calculated using junction tree approach:')

for i = 21:60
    % find a single JT potential that contains symptom 'i'.
    jtpotnum = whichpot(jtpot,i,1);

    % sum over everything but symptom 'i'.
    margpot=sumpot(jtpot(jtpotnum),i,0);

    % calculate and display output of symptom probability
    symptom_prob = margpot.table(1)./sum(margpot.table);
    disp(['p(' variable(i).name '=1) ' num2str(symptom_prob)]);
end
```

```
Probabilities calculated using junction tree approach:
p(s1=1) 0.44183
p(s2=1) 0.45668
p(s3=1) 0.44141
p(s4=1) 0.49127
p(s5=1) 0.49389
p(s6=1) 0.65748
p(s7=1) 0.50456
p(s8=1) 0.26869
p(s9=1) 0.64908
p(s10=1) 0.49074
p(s11=1) 0.42255
p(s12=1) 0.4291
p(s13=1) 0.54502
```

```
p(s14=1) 0.63296
p(s15=1) 0.42954
p(s16=1) 0.45879
p(s17=1) 0.42756
p(s18=1) 0.40426
p(s19=1) 0.58209
p(s20=1) 0.58959
p(s21=1) 0.76127
p(s22=1) 0.69559
p(s23=1) 0.5087
p(s24=1) 0.41996
p(s25=1) 0.35194
p(s26=1) 0.38961
p(s27=1) 0.32597
p(s28=1) 0.46962
p(s29=1) 0.52287
p(s30=1) 0.71731
p(s31=1) 0.5242
p(s32=1) 0.3537
p(s33=1) 0.51268
p(s34=1) 0.5294
p(s35=1) 0.38575
p(s36=1) 0.48909
p(s37=1) 0.6336
p(s38=1) 0.5896
p(s39=1) 0.42316
p(s40=1) 0.52823
```

Part 2

```
disp(' ') % new line
disp('Probabilities calculated using node selection approach:')

% Loop through all symptoms
for i = 21:60

    % find a potential that contains symptom 'i'
    symptom_pot = pot(i);
    % this is already in the right form.  just need
    % to access the other variables in the array

    % From the conditional pot of the symptom, extract the potentials
 that
    % the symptom is conditionally dependent on
    disease_pot_1_index = symptom_pot{1}.variables(2);
    disease_pot_2_index = symptom_pot{1}.variables(3);
    disease_pot_3_index = symptom_pot{1}.variables(4);

    % extract the disease potentials from the data, using the indices
    disease_pot_1 = pot(disease_pot_1_index);
    disease_pot_2 = pot(disease_pot_2_index);
    disease_pot_3 = pot(disease_pot_3_index);
```

```
    % reconstruct with new potentials with variables numbered 1,2,3,4,
    % so that the BN can be drawn for this subset of sympto and three
    % diseases, in order to check the form of the analysis.  Strictly
    % speaking this is not a necessary step and could be left out.
    symptom = 1;
    disease_1 = 2;
    disease_2 = 3;
    disease_3 = 4;

    new_pot{symptom} = array;
    new_pot{symptom} = symptom_pot{symptom};
    new_pot{symptom}.variables =
 [symptom,disease_1,disease_2,disease_3];

    new_pot{disease_1} = array;
    new_pot{disease_1} = disease_pot_1{1};
    new_pot{disease_1}.variables = disease_1;

    new_pot{disease_2} = array;
    new_pot{disease_2} = disease_pot_2{1};
    new_pot{disease_2}.variables = disease_2;

    new_pot{disease_3} = array;
    new_pot{disease_3} = disease_pot_3{1};
    new_pot{disease_3}.variables = disease_3;

    new_variable(symptom).name = variable(i).name;
    new_variable(disease_1).name = variable(disease_pot_1_index).name;
    new_variable(disease_2).name = variable(disease_pot_2_index).name;
    new_variable(disease_3).name = variable(disease_pot_3_index).name;

    % Calculate joint distribution
    jointpot = multpots(new_pot); % joint distribution

    % Draw new belief net, if required
    %drawNet(dag(new_pot),new_variable);

    % Calculate the marginal probability for this symptom 'i'
    result = condpot(jointpot, symptom);

    % display the results
    disp(['p(' variable(i).name '=1) ' num2str(result.table(1))]);

end


Probabilities calculated using node selection approach:
p(s1=1) 0.44183
p(s2=1) 0.45668
p(s3=1) 0.44141
p(s4=1) 0.49127
p(s5=1) 0.49389
p(s6=1) 0.65748
p(s7=1) 0.50456
```

```
p(s8=1) 0.26869
p(s9=1) 0.64908
p(s10=1) 0.49074
p(s11=1) 0.42255
p(s12=1) 0.4291
p(s13=1) 0.54502
p(s14=1) 0.63296
p(s15=1) 0.42954
p(s16=1) 0.45879
p(s17=1) 0.42756
p(s18=1) 0.40426
p(s19=1) 0.58209
p(s20=1) 0.58959
p(s21=1) 0.76127
p(s22=1) 0.69559
p(s23=1) 0.5087
p(s24=1) 0.41996
p(s25=1) 0.35194
p(s26=1) 0.38961
p(s27=1) 0.32597
p(s28=1) 0.46962
p(s29=1) 0.52287
p(s30=1) 0.71731
p(s31=1) 0.5242
p(s32=1) 0.3537
p(s33=1) 0.51268
p(s34=1) 0.5294
p(s35=1) 0.38575
p(s36=1) 0.48909
p(s37=1) 0.6336
p(s38=1) 0.5896
p(s39=1) 0.42316
p(s40=1) 0.52823
```

Part 3

```
disp(' ') % new line
disp('Probabilities of diseases given evidential variables.')

% Set symptoms 1 to 5 to present (state 1) and symptoms 6 to 10 not
% present (state 2)

% Set up indices for evidential variables, for readability
s1 = 21;
s2 = 22;
s3 = 23;
s4 = 24;
s5 = 25;
s6 = 26;
s7 = 27;
s8 = 28;
s9 = 29;
s10 = 30;
```

```
set_symptoms_index = [s1 s2 s3 s4 s5 s6 s7 s8 s9 s10];
set_symptoms_values = [1 1 1 1 1 2 2 2 2 2];

% Create a new potential with the evidential variables set
newpot = setpot(pot, set_symptoms_index, set_symptoms_values);

% Setpot removes variables, so we must eliminate redundant variables
 by
% squeezing the potentials
[newpot newvars oldvars]=squeezepots(newpot);

% Set up the new junction tree
[jtpot2 jtsep2 infostruct2]=jtree(newpot);

% Perform absorption
[jtpot2 jtsep2 logZ2]=absorption(jtpot2,jtsep2,infostruct2);

% After absorbing, transform back to original variables
jtpot2=changevar(jtpot2,newvars,oldvars);

% Loop through each disease and calcuate the new probability given the
% evidential variables.

for i = 1:20
    % find a single JT potential that contains disease 'i'.
    jtpotnum = whichpot(jtpot2,i,1);

    % sum over everything but symptom 'i'.
    margpot=sumpot(jtpot2(jtpotnum),i,0);

    % calculate and display output of symptom probability
    symptom_prob = margpot.table(1)./sum(margpot.table);
    disp(['p(' variable(i).name '=1 | s1:10) '
 num2str(symptom_prob)]);
end
```

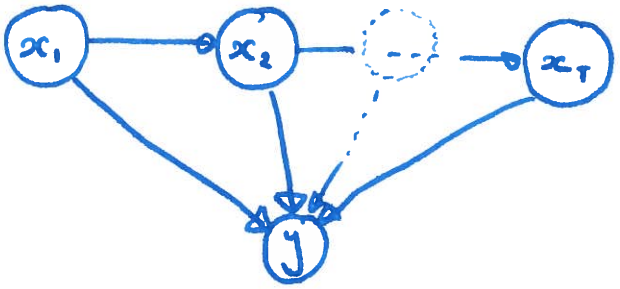
```
Probabilities of diseases given evidential variables.
p(d1=1 | s1:10) 0.029776
p(d2=1 | s1:10) 0.38176
p(d3=1 | s1:10) 0.95423
p(d4=1 | s1:10) 0.39664
p(d5=1 | s1:10) 0.49647
p(d6=1 | s1:10) 0.43515
p(d7=1 | s1:10) 0.18749
p(d8=1 | s1:10) 0.70118
p(d9=1 | s1:10) 0.043127
p(d10=1 | s1:10) 0.61031
p(d11=1 | s1:10) 0.28732
p(d12=1 | s1:10) 0.48983
p(d13=1 | s1:10) 0.8996
p(d14=1 | s1:10) 0.61956
p(d15=1 | s1:10) 0.92048
```

```
p(d16=1 | s1:10) 0.7061
p(d17=1 | s1:10) 0.20125
p(d18=1 | s1:10) 0.90849
p(d19=1 | s1:10) 0.86497
p(d20=1 | s1:10) 0.88393
```

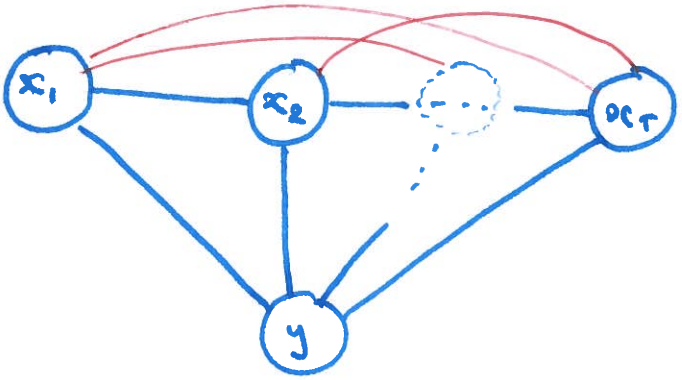
Published with MATLAB® R2016b

6.10

1) Drawing as a Belief Net first gives



The moralisation step introduces links between all the x_t as they are all parents of y. Shown below:



Because this network is a single maximal clique, the junction-tree representation is a single supernode, making any inference exponentially complex in the number of variables.

ie Junction Tree is:

2) Node y, when removed, leaves a singly connected subgraph.

Node y can be removed by marginalising over y first, ie:

$$\sum_y p(y \mid x_1, \ldots, x_T)\, p(x_1) \prod_{t=2}^{T} p(x_t \mid x_{t-1})$$

$$= p(x_1) \prod_{t=2}^{T} p(x_t \mid x_{t-1})$$

Now $p(x_T)$ can be computed in time that scales linearly with T by doing message passing

$1 \rightarrow 2 \rightarrow \ldots \rightarrow T$.

Note that normally "cut-set conditioning" would be used, but because y is only a child in this belief network, nothing is "conditionally dependent" on y, so cut-set conditioning is not needed.

Exercise 6.12

1. By explicitly denoting the clique neighbouring 1 as d, $P(X)$ can be rewritten as:

$$P(X) = \frac{\phi_d(X_d)\,\phi_1(X_1)\,\prod_{c\geq 2,\, c\neq d}\phi_c(X_c)}{\psi_1(S_1)\;\prod_{s\geq 2}\psi_s(S_s)}$$

where $\phi_1(X_1)$ is the only factor that is a function of $\{X_1 \setminus S\}$

Summing over $\{X_1 \setminus S\}$ therefore gives:

$$\left(\frac{\phi_d(X_d)\sum_{X_1\setminus S_1}\phi_1(X_1)}{\psi_1(S_1)}\right)\frac{\prod_{c\geq 2,\, c\neq d}\phi_c(X_c)}{\prod_{s\geq 2}\psi_s(S_s)}$$

where the part in brackets is only a function of X_d so can be renamed $\phi'_d(X_d)$.

Note that the left-hand side of the equation has changed and is now

$$\sum_{X_1\setminus S_1} P(X).$$

Relation to the absorption procedure:

The equivalent absorption $1 \to d$ would see the following two steps:

1. $$\psi_1^*(s_1) = \sum_{x_1 \setminus s_1} \phi_1(x_1)$$

2. $$\phi_d^*(x_d) = \frac{\phi_d(x_d)\,\psi_1^*(s_1)}{\psi_1(s_1)} = \frac{\phi_d(x_d) \sum_{x_1 \setminus s_1} \phi_1(x_1)}{\psi_1(s_1)}$$

The expression for $\phi_d'(x_d)$ derived in the question is identical to $\phi_d^*(x_d)$ from the absorption procedure.

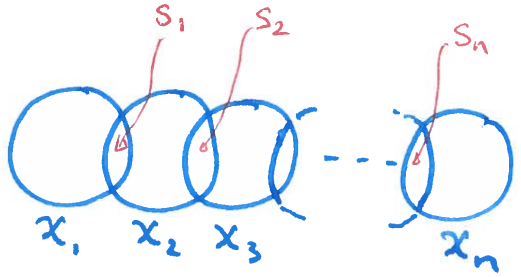
The differences to the absorption procedure are that we did not do the $\psi_1^*(s_1)$ step and that we have marginalised out the variables $x_1 \setminus s_1$ from $p(x)$.

So this is perhaps "absorbing and collapsing".

2)

It is easiest to show this result using venn diagrams to represent the sets of clique and separator variables.

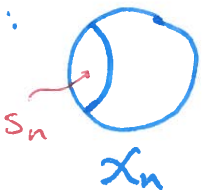
I will show for a chain, but this method generalises to a tree structure.



Summing over the variables $X_1 \setminus S_1$ simply removes the variables in the leftmost ◯ part of the diagram.

Continuing in this way, removing a ◯ at each step, the final clique contains:

ie all the variables in X_n.



Applying the same summation (over X / X_n) to $P(X)$ leaves $P(X_n)$.

So the final clique must equal $P(X_n)$.

3)

Demonstrate using a simple chain of cliques $1, \ldots, n$, so that it is clear that 1 neighbours 2 etc. Result will generalise to a tree.

For notational clarity, let $\phi_1 = \phi_1(x_1)$, $\psi_1 = \psi_1(s_1)$ etc.

Using the procedure outlined in part 1:

$$\phi_2' = \frac{\phi_2 \sum_{x_1 \setminus s_1} \phi_1}{\psi_1} = \sum_{x_1 \setminus s_1} \left(\frac{\phi_1 \phi_2}{\psi_1} \right)$$

$$\phi_3' = \frac{\phi_3 \sum_{x_2 \setminus s_2} \phi_2'}{\psi_2} = \sum_{x_1 \setminus s_1} \sum_{x_2 \setminus s_2} \left(\frac{\phi_1 \phi_2 \phi_3}{\psi_1 \psi_2} \right)$$

$$\vdots$$

$$\phi_n' = \sum_{x_1 \setminus s_1} \cdots \sum_{x_{n-1} \setminus s_{n-1}} \left(\frac{\phi_1 \phi_2 \cdots \phi_n}{\psi_1 \cdots \psi_{n-1}} \right) = p(x_n)$$

An important point is that we can consider

$$p(x) = \frac{\phi_1(x_1) \left(\prod_{c \geq 2}^{n-1} \phi_c(x_c) \right) \phi_n'(x_n)}{\psi_1(s_1) \prod_{s \geq 2} \psi_s(s_s)}$$

ie we have scaled ϕ_n. Effectively in doing so we have constrained the free variable 'z'.

So now using factors : $\phi_1, \phi_2', \phi_3', \cdots, \phi_n'$ and working back with the update procedure; using n as the (example) starting point.

$$\phi_d^* = \frac{\phi_d' \sum_{x_n \setminus s_{n-1}} \phi_n'}{\psi_d} \qquad \text{---- (1)}$$

Then $\phi_j^* = p(x_j)$ as required.

For example:

$$\phi_{n-1}^* = \frac{\phi_{n-1}' \sum_{x_n \setminus s_{n-1}} \phi_n'}{\psi_{n-1}}$$ using equation (1) above.

Substitute in for ϕ_{n-1}' from previous page

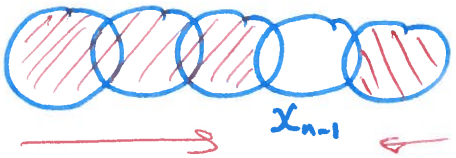
$$\phi_{n-1}^* = \frac{\left(\sum_{x_1 \setminus s_1} \cdots \sum_{x_{n-2} \setminus s_{n-2}} \left(\frac{\phi_1 \phi_2 \cdots \phi_{n-1}}{\psi_1 \cdots \psi_{n-2}} \right) \right) \sum_{x_n \setminus s_{n-1}} \phi_n'}{\psi_{n-1}}$$

rearranging summations gives:

this $= p(x)$

$$\phi_{n-1}^* = \sum_{x_1 \setminus s_1} \cdots \sum_{x_{n-2} \setminus s_{n-2}} \sum_{x_n \setminus s_{n-1}} \left(\frac{\phi_1 \phi_2 \cdots \phi_{n-1} \phi_n'}{\psi_1 \psi_2 \cdots \psi_{n-1}} \right)$$

The summations marginalise out all variables apart from x_{n-1}, working from both ends



recognising that the expression inside the summations is in fact $p(x)$, the summations simply calculate $p(x_{n-1})$, the marginal distribution.

This process applies to any of the cliques.

4) Once all the factors are probability distributions over the clique variables, the cliques must be globally consistent. (i.e. after the forward and reverse elimination schedules).

For example, for neighbouring cliques x_i and x_j

$\sum_{x_i \setminus I} \phi_i(x_i) = \sum_{x_j \setminus I} \phi_j(x_j)$ is correct, as both are probability distributions.

$\rightarrow \sum_{x_i \setminus I} P(x_i) = \sum_{x_j \setminus I} P(x_j)$

$\rightarrow P(I) = P(I)$.

for non-neighbouring cliques the equality holds too $(0 = 0)$.

$\rightarrow$ globally consistent.