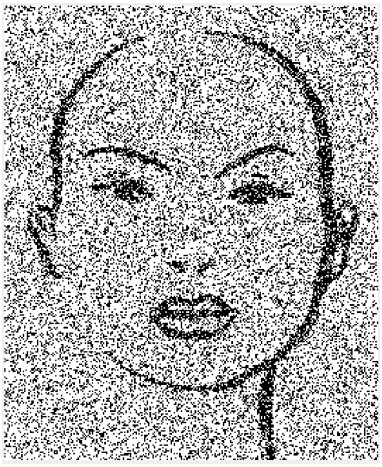
Graphical Models Assignment 2

Gonzalo Barrientos, Nick Thomson, Sam Maule,
Siddharth Chatterjee and Zobair Hasan

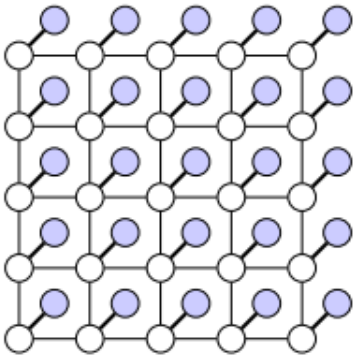
November 17, 2017

Exercise 4.16. We are tasked with denoising an image of a face. The noisy image is as follows:



The collection of pixels are modeled as a Markov Network. We observe only the noisy pixels which are modeled as independent of each other. The noisy pixels are conditionally independent of the values of all other pixels (noisy or true) given the value of the true pixel that it corresponds to.

The true (de-noised) pixels are conditionally independent of each other given the state of their neighbouring pixels (de-noised). The graph is therefore a lattice, as below:



The joint distribution of the noisy pixels (Y) and true pixels (X) factorises to:

$\mathrm{p(X,Y)} \propto \prod_{i=1}^{D} \phi'(X_i, Y_i) \prod_{i=1} \psi'(X_i, X_j)$

We may optimise log p(X,Y) wrt to X to maximise P(X,Y), this is equivalent to maximising $P(X \mid Y)$ the probability of a potential set of 'true' pixels given the observed noisy image. The log form of P(X,Y) is as follows:

$\sum_i \phi(X_i, Y_i) + \sum_{i\ j} \psi(X_i, X_j)$

For this question, we are told the objective function takes the following from with $W_{i,j}$ = 10 and $b_i = 2y_i$:

$\sum_{i,j} W_{i,j} \parallel [X_i = X_j] + \sum_i b_i x_i$

Given the form of the objective function, we prefer neighbouring pixels to be in the same state and prefer 'true' pixels which match the state of the 'noisy' pixel over those that do not. This is because despite the noise, the noisy image will still provide some knowledge of the underlying true image.

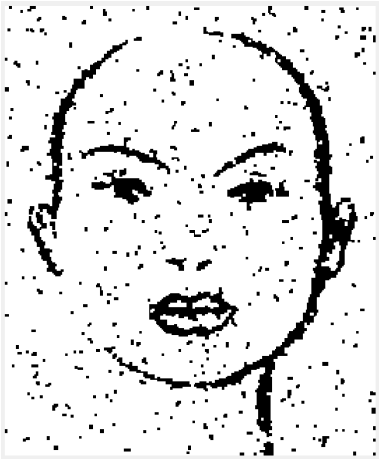
The noisy image provided is 321 × 265 pixels. We first populate a matrix of weights between the pixels. The matrix is (321 × 265) × (321 × 265) with each element constituting a weight between two pixels. We consider a pixel in position (x,y) and set it's neighbours (above, below, left and right) to 1, therefore giving preference to the state of a pixel matching that of neighoburing pixels. The code implementing the aforementioned procedure is as follows:

```
xnoisymatrix = importdata('noisyface.mat');
xnoisy = xnoisymatrix(:); %convert to column vector

Gx=321; Gy=265; N=Gx*Gy;
st = reshape(1:N,Gx,Gy); % assign each grid point a state

sp = sparse(10,10,pi);
W0=zeros(N,N,'like',sp); %data needs to be in a sparse matrix due to size
for x = 1:Gx
    for y = 1:Gy
        if validgridposition(x+1,y,Gx,Gy); W0(st(x+1,y),st(x,y))=1; end
        % validgridposition catches where a pixel has no neighbour on one
        % side (e.g. on the edge of the image)
        if validgridposition(x-1,y,Gx,Gy); W0(st(x-1,y),st(x,y))=1; end
        if validgridposition(x,y+1,Gx,Gy); W0(st(x,y+1),st(x,y))=1; end
        if validgridposition(x,y-1,Gx,Gy); W0(st(x,y-1),st(x,y))=1; end
    end
end
```

We then use the binaryMRFmap function from the tool kit to maximise our objective function. This is done using the Iterated Conditional Modes (ICM) algorithm. This algorithm starts with an initial random set of 'true' pixels, fixes all pixels in the set bar one and optimises with respect to that single pixel. This is repeated for all pixels. This is repeated for all remaining pixels. This constitutes one full iteration of the ICM. We can run multiple iterations by initialising the ICM with the 'optimal' set of true pixels from the previous run. Each iteration therefore may improve upon the previous until a local optimum is reached. The code running the ICM is as follows:

```
b = 2*xnoisy; % bias to favour the noisy image
W=10*W0; % preference for neighbouring pixels to be in same state

opts.maxit=1; opts.minit=1; opts.xinit=xnoisy(:);
for loop=1:20
    [xrestored E] = brml.binaryMRFmap(W,b,1,opts);
    opts.xinit=xrestored;
    endofloop = loop
end
final = reshape(xrestored,Gx,Gy);
maximumobjectivefunction = E;
```

Our final denoised image (after 50 iterations) is as follows:



With 50 iterations our objective function is equal to 3360028. We also ran 5 and 20 iterations which gave values 3358078 and 3358254 respectively. The denoised image after 20 iterations versus 50 iterations was almost identical.

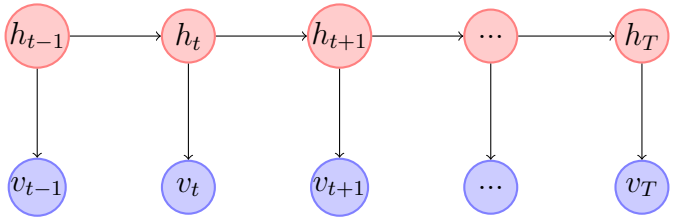
ZH

Exercise 5.8. According to the question, there are one set of first names and one set of surnames.

first names = {David,Anton, Fred, Jim, Barry}
surnames = {Barber,Ilsung, Fox, Chain, Fitzwilliam, Quinceadams,Grafvonunterhosen }

A noisy string is provided and we were tasked to denoise the string. The cleaned strings should have firstnames following by surnames. Because we have a sequence of noisy characters (observations) we can model it as an HMM.

From the problem we know that:
- First hidden state: randomly sampling a character from a to z.
- Self-transition probability of hidden state 1 is 0.8
- The first names are chosen uniformly at random, that means that Transition probabilities from the first state to the first characters of each firstname should be $A_{i',i} = 0.2/5$ since we must divide probability mass between the 5 firstnames.
- We should include a hidden state that limits the end of the firstname and the beginning of the surname. Self-transition probability of this hidden state is 0.8 - We follow the same approach from the transition probabilities of firstnames. Now we should consider $A_{i',i} = 0.2/7$
- After the last character of the surname is generated it goes back to the start (1st hidden state).
- Each character is generated correctly with probability 0.3.



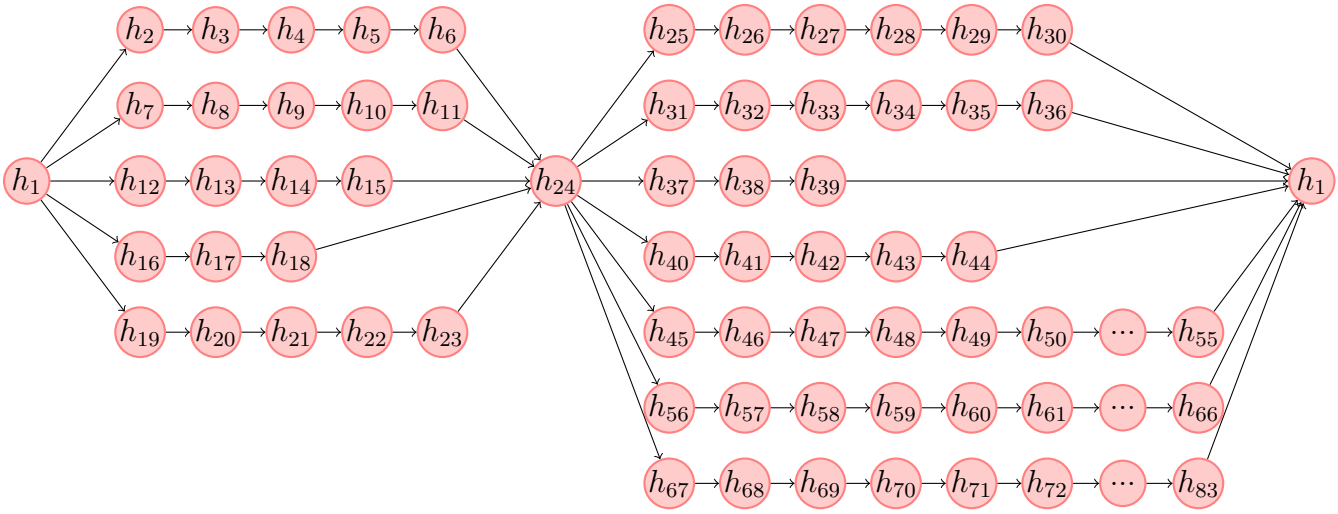
We could model it using a simple scheme as below, however I my model should handle properly the surnames and firstnames. This means that if the surname is BARBER, the model should consider that the transition probability from "B" to "A" is 1 and from "A" to "R" is 1 too and so on. (also that the surnames should came after the firstnames). That is why we decide to use a model that takes in account this criteria.

We made the following considerations for the model: h_1 is the firstate that generates random numbers. Because the firstnames and surnames need a transition probability of 1, we are considering a sequence of states for each firstname/surname:

h_2 to h_6: D,A,V,I,D
h_7 to h_{11}: A,N,T,O,N
h_{12} to h_{15}: F,R,E,D
h_{16} to h_{18}: J,I,M
h_{19} to h_{23}: B,A,R,R,Y

h_{25} to h_{30}: B,A,R,B,E,R
h_{31} to h_{36}: I,L,S,U,N,G
h_{37} to h_{39}: F,O,X
h_{40} to h_{44}: C,H,A,I,N
h_{45} to h_{55}: F,I,T,Z,W,I,L,L,I,A,M
h_{56} to h_{66}: Q,U,I,N,C,E,A,D,A,M,S
h_{67} to h_{83}: G,R,A,F,V,O,N,U,N,T,E,R,H,O,S,E,N

For these states, we consider a transition probability of 1 between each state of each first-name/surname. So, considering the 83x83 transition matrix:
$A_{i',i} = p(h_{t+1} = i' \mid ht = i)$
For h_2 to h_6, we have: $A_{3,2} = 1, A_{4,3} = 1, A_{5,4} = 1, A_{6,5} = 1$
We do the same with all the first names and surnames.
Also, h_{24} is the state that comes after the first name and before the surname.
We must point out that the data should be encoded so it can be processed easily. That means that "a" = 1, "b" = 2, "c" = 3 until "z" =26; this also will be useful to express the emission probabilities.



For the emission probabilities, we will have a matrix of size 26 x 83 (states of observations x number of states) as follows:

$B_{i,j} = p(v_t = i \mid ht = j)$
For the state h_1 and h_{24} the output will be uniformly random, so we can consider that:
$B_{i,1} = 1/26, B_{i,24} = 1/26$, since there are 26 letters in the English alphabet.

As we know each character has a probability of being generated correctly of 0.3. That means that for state 2 we have the following:
$B_{2,2} = 0.3$ and $B_{i,2} = 0.3/25, i \neq j$; these probabilities mean that we only have 0.3 of probability of generated the number of the hidden state, in relation of (2,2) the left 2 refers to the letter "b" and the right one to the state. In this case we know that state 2 should generate

"b". The other 0.7 should be shared with the other elements in the column 2. We divide by 25, since we shouldn't take in account the letter "b". Following this approach we fill all the emission probability matrix.

Regarding the initial probability vector 1 x 83, all the values will be zero, except for $p(h1 = 1) = 1$

We used and modified the "demoHMMinferenceSimple" file from the BRML Toolbox (see the Annex), in order to use the Vitervi algorihtm to find the most likely sequence. We update the number of states of observations (26), the number of states (83) and the length of the sequence (10000). We imported the 'noisystring.mat' file and state the considerations made above about the transition matrix and the emission matrix. We also modified the code in order to encode the data into numbers from 1 to 26, and then decode the result and join it into one string, mapping all random characters from state h_1 and h_{24} with the dollar sign.

Here is one sample of the output of the most likely sequence:

'$$$$$$$DAVID$$$$$$$$QUINCEADAMS$$$$DAVID$$$$$QUINCEADAMS$$
$$$$DAVID$$$$$$$$$$FITZWILLIAM$$$$$$$$$$$$$$$$$$ ... '

We used Python for the data manipulation; result string is the output from the MATLAB function

```
result_string = '$$$$$$$DAVID$$$$$$$$QUINCEADAMS$$$
$DAVID$$$$$QUINCEADAMS$$$$$$DAVID$$$$$$$$$$FITZWILLIAM$$$ ... '
```

First we remove the dollar sign from our string, and then create a list where each element is each word in order of appearance:

```
s = result_string
word = ['DAVID', 'ANTON', 'FRED', 'JIM', 'BARRY',
'BARBER', 'ILSUNG', 'FOX', 'CHAIN', 'FITZWILLIAM',
'QUINCEADAMS', 'GRAFVONUNTERHOSEN']

list_1 = [x for x in s.split('$') if len(x)>0]
```

Here is a sample of the contents of the new list:

```
list_1[:10]
['DAVID',
 'QUINCEADAMS',
 'DAVID',
 'QUINCEADAMS',
 'DAVID',
 'FITZWILLIAM',
```

'JIM' ,
'FITZWILLIAM' ,
'BARRY' ,
'QUINCEADAMS']

We created a dictionary so we could count the firstname + surname combinations in our new list:

{ 'ANTON': {'BARBER': 0,'CHAIN': 0, 'FITZWILLIAM': 0, 'FOX': 0, 'GRAFVONUNTERHOSEN': 0, 'ILSUNG': 0, 'QUINCEADAMS': 0 },

'BARRY': {'BARBER': 0, 'CHAIN': 0, 'FITZWILLIAM': 0, 'FOX': 0, 'GRAFVONUNTERHOSEN': 0, 'ILSUNG': 0, 'QUINCEADAMS': 0 },

'DAVID': {'BARBER': 0, 'CHAIN': 0, 'FITZWILLIAM': 0, 'FOX': 0, 'GRAFVONUNTERHOSEN': 0, 'ILSUNG': 0, 'QUINCEADAMS': 0},

'FRED': {'BARBER': 0, 'CHAIN': 0, 'FITZWILLIAM': 0, 'FOX': 0, 'GRAFVONUNTERHOSEN': 0, 'ILSUNG': 0, 'QUINCEADAMS': 0},

'JIM': {'BARBER': 0, 'CHAIN': 0, 'FITZWILLIAM': 0, 'FOX': 0, 'GRAFVONUNTERHOSEN': 0, 'ILSUNG': 0, 'QUINCEADAMS': 0}}

We made a script that count the values and populate the dictionary

```
for i in range(0,len(list_1)-1):
    for key, value in dict_names.items():

        if list_1[i] == key:
            dict_names[key][list_1[i+1]] += 1
```

We made a script that count the values and populate the dictionary; each key is the firstname, the value is another dictionary that has the surname as key and the count of that firstname and surname combination as a value. Below the dictionary:

{ 'ANTON': {'BARBER': 8,'CHAIN': 8, 'FITZWILLIAM': 20, 'FOX': 2, 'GRAFVONUNTERHOSEN': 14, 'ILSUNG': 9, 'QUINCEADAMS': 10 },

'BARRY': {'BARBER': 10, 'CHAIN': 11, 'FITZWILLIAM': 7, 'FOX': 9, 'GRAFVONUNTERHOSEN': 17, 'ILSUNG': 5, 'QUINCEADAMS': 10 },

'DAVID': {'BARBER': 7, 'CHAIN': 11, 'FITZWILLIAM': 16, 'FOX': 9, 'GRAFVONUNTERHOSEN': 10, 'ILSUNG': 2, 'QUINCEADAMS': 9},

'FRED': {'BARBER': 6, 'CHAIN': 10, 'FITZWILLIAM': 10, 'FOX': 8, 'GRAFVONUNTERHOSEN': 19, 'ILSUNG': 5, 'QUINCEADAMS': 8},

'JIM': {'BARBER': 9, 'CHAIN': 18, 'FITZWILLIAM': 20, 'FOX': 7, 'GRAFVONUNTERHOSEN': 17, 'ILSUNG': 9, 'QUINCEADAMS': 12}}

According to the statement of the question, the result should be presented as a matrix with the counts of the number of occurrences of the pair (firstname(i), surname(j)). So we wrote a script that uses the information in the dictionary and create this matrix.

```
list_firstnames = ['DAVID', 'ANTON', 'FRED', 'JIM', 'BARRY']
list_surnames = ['BARBER', 'ILSUNG', 'FOX', 'CHAIN',
'FITZWILLIAM', 'QUINCEADAMS', 'GRAFVONUNTERHOSEN']

matrix_question = np.zeros((5,7))

for i in range(0,5):
    for j in range(0,7):
        for key, value in dict_names.items():
            for keys,values in value.items():
                if key == list_firstnames[i]:
                    if keys == list_surnames[j]:
                        matrix_question[i,j] = values
```

This is the final output, being the rows the firstnames and the columns, the surnames (in the order stated at the beginning of the question).

$$\begin{bmatrix} 7 & 2 & 9 & 11 & 16 & 9 & 10 \\ 8 & 9 & 2 & 8 & 20 & 10 & 14 \\ 6 & 5 & 8 & 10 & 10 & 8 & 19 \\ 9 & 9 & 7 & 18 & 20 & 12 & 17 \\ 10 & 5 & 9 & 11 & 7 & 10 & 17 \end{bmatrix}$$

GB

Exercise 5.9. We are tasked with devising a method to track a dangerous drunk. We are given a matrix of data capturing a grid of the train station at each time stamp (1 to 100). If a position is occupied by a person the position on the grid has a value of 1. We can approach this model as a hidden markov model and calculating the most likely path of the dangerous drunk given observed data.

Each person can be individually. The hidden variable represents the actual position of a person in the grid at time t. The visible variable represents which positions are occupied at time t. We will treat each position in the grid as a state. The grid is 50 x 50, therefore our model will have 2500 states (see appendix for code used to assign state numbers to grid positions). As we are interested in the dangerous drunk's most likely path, we will approach this problem by modeling only the dangerous drunk.

As we are interested in the most probable path taken by the drunk, we are interested in calculating $p(h_{1:T} \mid v_{1:T})$, the probability of a path given the observed data. This is proportional to :

$$p(h_{1:T}, v_{1:T}) = \prod_t p(v_t \mid h_t) p(h_t \mid h_{t-1})$$

We may find the most probable path by then finding the maximal $p(h_{1:T}, v_{1:T})$. An algorithm that can be used to efficiently compute find the most probably path is the Viterbi algorithm. The key to the Viterbi algorithm is message passing.

$$\max_{h_{1:t}} p(h_{1:T}, v_{1:T}) = \max_{h_{1:t}} \prod_t p(v_t \mid h_t) p(h_t \mid h_{t-1})$$

Looking at a toy example where T = 2. We see that the following factorisation can be made:

$$\max_{h_{1:2}} p(h_{1:2}, v_{1:2}) = \max_{h_2} p(v_2 \mid h_2) p(h_2 \mid h_1) \max_{h_1} p(v_1 \mid h_1) p(h_1)$$

Where $\max_{h_1} p(v_1 \mid h_1) p(h_1)$ is defined as the message $\gamma(h_1)$. More generically we have have $\gamma(h_{t+1})$ as follows:

$$\gamma(h_{t+1}) = \max_{h_t} \gamma(h_t) p(v_{t+1} \mid h_{t+1}) p(h_{t+1} \mid h_t)$$

The above equation provides the foundation for recursion. The Viterbi algorithm recursively computes the messages from 1 to T, so for any T we know the value of message received. This is efficient as we are finding the maximum for each T (node in hidden markov model) and only passing the maximum forward. We do not need to store or passfoward multiple states or values. Once we have these messages, the second part of the Viterbi algorithm is backtracking.

To find the most probable path we can first identify what is the most probable state at time T. We already have the message from T-1, we are only required to compute 1 additional message and find the maximal H_t. To find the most probable state at T-1, again we already have the message from T-2, we only need to compute 1 additional message once more. This is repeated, tracking back through time for each T thus giving us the most probable path.

To apply the viterbi algorithim to our tracking problem, we need the emissions $p(v_t \mid h_t)$, the transitions $p(h_t \mid h_{t-1})$ and the initial distribution $p(h_1)$.

Calculating the emissions matrix $p(v_t \mid h_t)$ is a challenge for this problem. Our observation v_t is not a single number but a matrix grid of positions set to 1 if occupied and 0 otherwise. We can still however extract the necessary information from this matrix. We are looking for the probability of an observation given the hidden state (true position of the drunk). This can actually be deduced from our observation matrix. We can think of the value in our matrix of occupied postions as $p(v_t \mid h_t, t)$ i.e. the probability of seeing this particular position occupied at time t given the true position of the drunk. An example to bring this alive is the case where we observe a position X,Y to be 0. If the true position of the drunk is X,Y then the probability of observing the position X,Y as 0 (unoccupied) is 0. Whereas, if we observe X,Y to be 1. $p(v_t \mid h_t, t) = 1$. The probability of seeing the position X,Y occupied given the drunk is at X,Y is 1.

We do not have multiple observations (v_t) for each state (h_t) per timestamp, we have a single observation per hidden variable for each timestamp. Our emissions matrix is therefore 2500 x 100 where the rows are the 2500 unique hidden states (true position of the drunk) and the columns the timestamp. Element i,j is 1 if position i is occupied in the observations from time j. Note, this emissions matrix does not describe a probability distribution, the columns do not sum to 1. However, the individual elements of the matrix could be thought of as probabilities as described above. The code populating the emissions matrix may be found in the appendix.

For the transition matrix $p(h_t \mid h_{t-1})$ we incorporate the information regarding the drunk's movement pattern. Given the drunk is in a certain 'state' (position X,Y in the grid), we model it so that the drunk is equally likely of moving in 1 of four directions: Northwest (X-2,Y-2), Northeast (X+2,Y-2), Southwest (X-2,Y+2) and Southeast (X+2,Y+2). Assuming all four directions are within the grid. Therefore each of the four possible grid position is given 0.25 probability and all other grid positions are given 0 probability.

We also consider 8 other cases. 4 of which cover the situations where the drunk is at the edge (side) of the grid e.g. position (1,25), but not in the corner e.g. (1,1). In these cases, 2 of the four possible grid positions are still valid e.g. From a starting position of (1,25) the drunk still move to (3,23) and (3,27). We assign 0.25 probability to those. The remaining 0.5 is distributed evenly across all remaining grid positions (as out of grid people are randomly reappear in another position). This assumption is inferred from examining drunkmover.m. The direction the drunk moves in is randomly selected from the four possible directions, and it is only after selection is it tested to see if the direction is valid. Therefore, there is a 0.5 chance of selecting an invalid direction and ending up randomly reassigned position. The remaining 4 cases are at the corners where there is only 1 valid direction the drunk can move in. We therefore assign 0.25 probability to the valid position and distribute the remaining probability evenly across all other positions. The code used to populate the transition matrix may be found in the appendix.

For the initial distribution $p(h_1)$, from drunkmover.m we see that the no person (including the drunk) beings close to the edges or the corners i.e. $X < 3$ or $X > 47$ and $Y > 3$ or $Y > 47$. We therefore assign 0 probability to those positions. The rest of the valid positions are treated as equally likely. The code populating the initial distribution may be found int he appendix.

To apply the Viterbi algorithm we use a customised version of the HMMViterbi function from the BRLM toolkit. The basic function expects observations to be passed in so that we can look up these observations in the emissions matrix to find the probability of this observation given a particular hidden variable. However, in this case, we do not need to pass in observations separately as our observations are already incorporated into the probabilities of the emissions matrix. Furthermore, the basic implementation of the function also does not take into account the time dependent nature of our new emissions matrix. A standard emissions matrix has the probability of an observation given a hidden state. The emissions matrix for this problem has the probability of an observation given a hidden variable, for each timestamp. Therefore, a change is required to look up the probability from the correct timestamp.

The customised version of the HMMViterbi looks up probabilities directly from the emissions matrix based on the timestamp of the message being computed. The custom HMMViterbi has changes to the computation of the messages and the compuation of 'additional' message when backtracking. As discussed previously, the customisations mean $p(v_t \mid h_t, t)$ is used (as modeled for this problem) instead of the standard $p(v_t \mid h_t)$. The customised function may be found in the appendix.

The co-ordinates for the most probable path followed by the drunk are below. They are of the form (X,Y) starting with t=1 and progressing from left to right.

(39 , 28) (41 , 30) (39 , 32) (37 , 30) (39 , 28) (41 , 30) (43 , 28) (45 , 26) (43 , 24)
(41 , 26) (43 , 24) (41 , 22) (43 , 20) (45 , 22) (43 , 24) (41 , 22) (39 , 24) (37 , 22) (39 , 24)
(37 , 26) (39 , 28) (37 , 26) (39 , 24) (37 , 26) (35 , 24) (37 , 22) (35 , 20) (37 , 22) (39 , 24)
(41 , 26) (39 , 24) (41 , 26) (43 , 24) (41 , 26) (39 , 28) (37 , 30) (9 , 32)
(41 , 30) (43 , 32) (41 , 30) (43 , 32) (41 , 30) (39 , 32) (41 , 34) (39 , 32) (41 , 30) (39 , 32)
(41 , 34) (39 , 32) (41 , 30) (39 , 28) (41 , 30) (39 , 32) (41 , 34) (43 , 36) (41 , 38) (39 , 40)
(37 , 42) (39 , 40) (37 , 38) (35 , 40) (33 , 42) (35 , 40) (33 , 38) (35 , 36) (33 , 34) (31 , 32)
(33 , 34) (35 , 36) (37 , 38) (35 , 36) (37 , 38) (35 , 40) (33 , 38) (31 , 36) (33 , 38) (31 , 40)
(29 , 42) (31 , 40) (29 , 42) (31 , 40) (33 , 42) (31 , 44) (29 , 46) (27 , 48) (25 , 46) (27 , 48)
(25 , 46) (27 , 44) (25 , 46) (27 , 48) (25 , 46) (27 , 48) (25 , 46) (23 , 44) (25 , 46) (23 , 44)
(21 , 46) (19 , 44) (21 , 46)

ZH

Exercise 5.10. We are asked to calculate the expected price gain of a financial asset and its standard deviation. We find the expected price gain to be 1.943 (4 s.f) with standard deviation 8.541 (4 s.f).

We are given price data through $T = 200$ time periods, with the price able to take integer values from 1 to 100.

We are told that the market can either be in an unobserved 'bear' or 'bull' states in any time period. This is a hidden variable so we will denote it as $h_t = i$ where i is the state of the market at time period $t = \{1, ..., 200\}$ suggesting a hidden Markov model. We are told that $p(h_1 = bear) = 0.5$ and $p(h_1 = bull) = 0.5$. We are also told that the state of the market changes through time according to the following transition matrix:

$$T = p(h_t \mid h_{t-1}) = \begin{bmatrix} 0.8 & 0.3 \\ 0.2 & 0.7 \end{bmatrix}$$

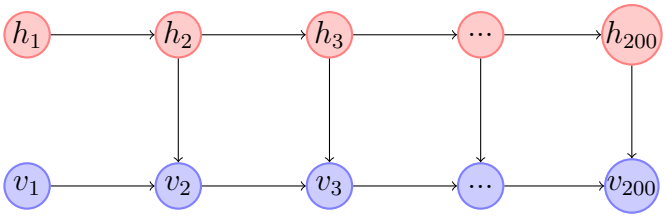
Where T_{ij} gives the probability of moving from state j to state i with $bear = 1$ and $bull = 2$.

The price in time period t, denoted v_t depends on both the price at $t-1$, and h_t, the state of the market at time period t. We are given conditional probability tables, pbear and pbull which give values for $p(v_t \mid v_{t-1}, h_t)$. At timestep one the price distribution is uniform.

From this information we can establish that the joint distribution that models the problem is as follows:

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

With the corresponding belief network:



To establish the expected future price gain, we must first calculate the probabilities associated with each state of the market at $t = 200$. That is, we are interested in:

$$p(h_t, v_{1:t}) = \sum_{h_{t-1}} p(h_t, h_{t-1}, v_{1:t})$$

$$= \sum_{h_{t-1}} p(v_t \mid h_t, v_{t-1})p(h_t \mid h_{t-1})$$

This is a filtering problem and can be solved using the BRML toolbox. To do so required some minor tweaks to the HMMforward algorithm and manipulation of the provided data. We will not outline this in detail here, but full commented MATLAB code is provided in the Appendix.

The output of the adapted HMMforward algorithm is a 2x200 matrix containing the probability of being in hidden state i at time period j. For t = 200 $p(h_{200} = bear) = 0.1135$ and $p(h_{200} = bull) = 0.8865$.

Using the transition matrix above we can calculate that $p(h_{201} = bear) = 0.3568$ and $p(h_201 = bull) = 0.6432$.

Knowing that $v_{200} = 58$ we now have all the information required to calculate the probability distribution of the price in time period 201: $\sum_{h201} p(v_{201} \mid v_{200} = 58, h_{201})$

From there it is straightforward to take a weighted average to find the expected price $v_{201} = 59.4323$. This corresponds to an expected price gain of 1.943 (4 s.f). The standard deviation of this expected price gain is 8.541 (4 s.f).

SM.

Exercise 5.11. a.
From the problem we know that $\boldsymbol{x_1} = (0,0,0)^T$ and $\boldsymbol{v_1} = (0,0,0)^T$. Also the following:

$\boldsymbol{v}_{t+1} = \boldsymbol{v}_t + \delta \boldsymbol{a}_t$
$\boldsymbol{x}_{t+1} = \boldsymbol{x}_t + \delta \boldsymbol{v}_t$

So, we can derive that for each dimension in the vectors:
$x_2 = x_1 + \delta v_1 = 0$
$x_3 = x_2 + \delta v_2 = 0 + \delta(\delta a_1) = \delta\delta a_1$
$x_4 = x_3 + \delta v_3 = \delta\delta a_1 + \delta(\delta a_1 + \delta a_2) = \delta\delta a_2 + 2\delta\delta a_1$
$x_5 = x_4 + \delta v_4 = (\delta\delta a_2 + 2\delta\delta a_1) + \delta(\delta a_1 + \delta a_2 + \delta a_3) = \delta\delta a_3 + 2\delta\delta a_2 + 3\delta\delta a_1$

Now we write the general formulation, taking in account that T is 100, so 102 will be $T+2$ and that the superscript i refers to each element in the vectors $\boldsymbol{x}$ and $\boldsymbol{a}$:

$$x^i_{102} = x^i_{T+2} = \delta\delta \sum_{i=1}^{T} (i)\, a^i_{T+1-i}$$

b.
We need to get the minimum amount of fuel given by the equation $\sum_{t=1}^{T=100} \sum_{i=1}^{3} |a_i(t)|$. We decided to model it as a Markov decision process, for this question we are going to use the file demoMDPclean.m from the BRML Toolbox. We are going to have 3 decision states: +1,0,-1; we are going to consider each state as the coordinate in each vector multiply by 100. For example for coordinate i we created a grid of 1000x1000, and the goal is to reach state 471 in time T=102. So, we populated the transition matrix using this pseudo code:

```
A = np.zeros((1000,1000,3))

delta = 0.1
a = 100
x_actual = 0.
acc = [0.]
x_new = 0.

for i in range(2,1000):
    x_new = int(round(x_actual + (delta*delta*(sum(acc[:-1]))),2))
    acc.append(a)
    x_actual = int(x_actual)

    x_state_new = x_new
    x_state_actual = x_actual
    A[x_state_new,x_state_actual,1] = 1

    x_actual = x_new
```

According to our formulation in a. the value of a_t doesn't affect x_{t+1}, so given a state x_t it will change to x_{t+1} independently of the value of a_t. In this case the decisions are the values of the acceleration in a time t (+1,0,-1). After, we change the number of goal states to 1, and set this value to be the state 471 (for coordinate i, for coordinate j 697, and for k, 859)

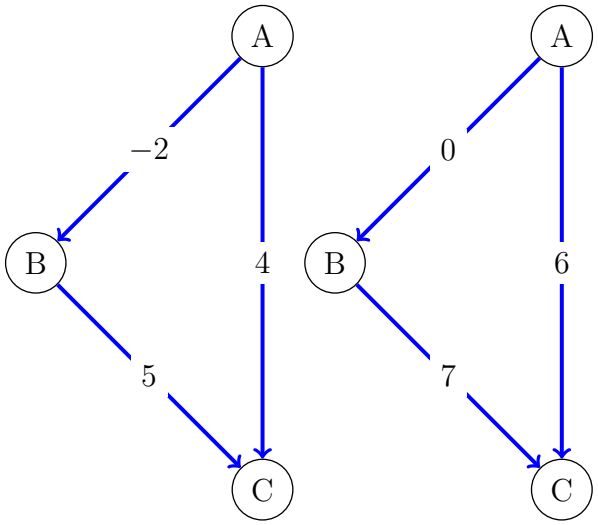
c. If we consider that an efficient polynomial time, we could use the Bellman equation, since this equation has a time complexity of $O(VE)$, being V the nodes and E the edges.

This equation define the value of being in state x_t:
$v_{t-1}(x_{t-1}) = u(x_{t-1}) + \max_{d_{t-1}} \sum_{x_t} p(x_t \mid x_{t-1}, d_{t-1}) v_t(x_t)$

The optimal decision d_t:
$d^* t_t(x_t) = argmax_{d_t} \sum_{xt+1} p(x_{t+1} \mid x_t, d_t) v_t(x_{t+1})$

GB

Exercise 5.12. The reason that this would not work is that 2 weighted paths through a graph may not have the same number of edges. This can be proven with a trivial example:



On the left, A to C has a minimum weighted path A-B-C, this has weight 3, this can be found using Dijkstra's Algorithm. However, when we subtract the minimum weight (-2) from the other weights, the minimum weighted path will now be A-C with weight 6. Path A-B-C now has weight 7. This is due to the fact that A-B-C has 2 edges along it's path while A-C has 1 and since each edge has -2 subtracted from it, increasing its' weight by 4. Therefore subtracting the least weighted edge from all other edges in a graph will not work for finding the least weight path through a graph.

NT

Exercise 5.13. We can model this question using a Hidden Markov Model and solve using the most probable path algorithm, where the most probable path is given by the path that has the least weight from source to sink. This is done by

Using this Python code we were able to pre-process the Simohurrta.mat dataset into a single transition matrix derived from the matrix A in the original dataset.

```
for idx1, value1 in enumerate(A):
    for idx2, value2 in enumerate(A):
        if A[idx1, idx2] == 0:
            A_prime[idx1, idx2] = np.inf
        else:
            A_prime[idx1, idx2] = distance(x.T[idx1], x.T[idx2]) - t.T[idx2]
```

We were then able to tweak the original demoShortestPath.m file in order to accept the new matrix and run the file.

This file takes the log weight of each path(equivilent to taking the log probability of taking that particular path) to the current planet from planet 1. The minimum weight path is then calculated by adding the weights at the current planet(to all other planets) to the previous minimum path. The minimum path is then updated and passed forward to the

next node. This process is repeated until we reach planet 1725 where the optimal path is selected as the minimum weight path.

The optimal path is 707 planets long and the resulting cost is -209.65. This complete path is in the appendix along with the tweaked demoShortestPath.m file.

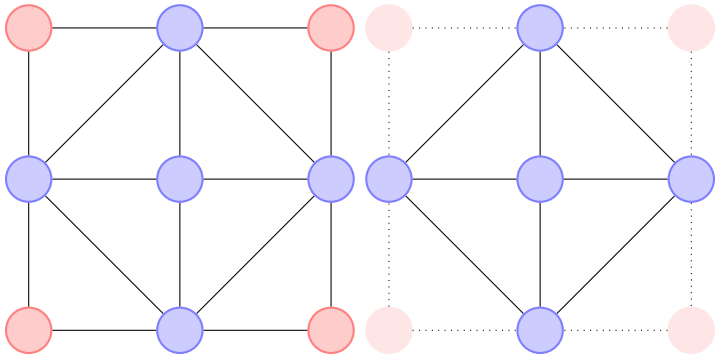
NT

Exercise 6.6. A triangulated graph is a graph for which all vertices of 4 or more nodes has a chord that connects 2 vertices in the cycle. A clique is a graph in which every 2 nodes are adjacent. Finding a minimum-maximal clique in a triangulated graph is an NP-hard problem, initially we solved using greedy variable to find a triangulated graph with a maximal clique of 5 vertices.

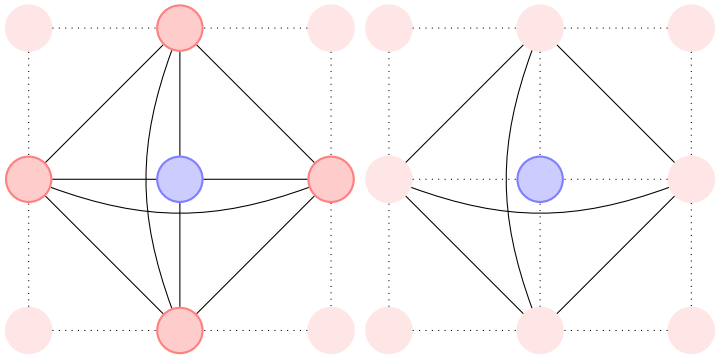
The greedy variable elimination heuristic runs as follows:

1. Take the nodes with the minimum edges connected to them (We can mark these nodes Red)
2. Join the parents of these nodes (with a solid line)
3. Eliminate these nodes (fade and dot the connected edges)
4. repeat steps 1-3 until the graph is fully triangulated

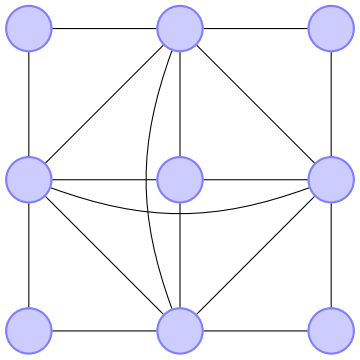
Round 1:



Round 2:



After 2 rounds we are left with the single center node. This cannot be reduced and thus we have a finalized triangulated graph with a max-clique of 5.



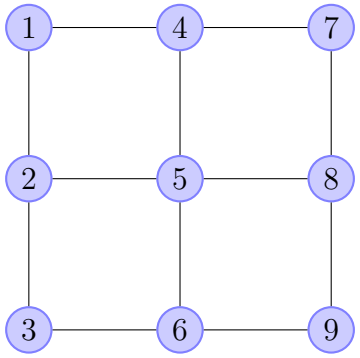
NT

Exercise 6.7. The steps to compute the normalisation constant Z are as follows(the comment code implementing this steps may be found in the appendix):

- Calculate the potentials for all 2^2 possible pairs of states two binary variables can take.
- Assign a unique number to each variable in the graph (from 1 to N*N) based on their ordering. As we are interested in columns, the ordering will go down along the columns.
- Identify all pairs of variables which have a valid potential i.e. such that i > j where i and j are the variable numbers. Furthermore, assign each pairing an id and store the potentials for each pairing against this id.

  Note: Variables in the first column only have potentials specified with 1 other variable, i.e. the variable immediately above it. Variable 1 has no variables above it and therefore has no potentials specified. For all the other 9 columns each variable has a potential with the variable above it and to its left. This is unless the variable is on the top row of the lattice in which case it only has a potential with the variable to its left.
- The singly connected graph of the cluster variables would have a joint distribution as follows: $p(X_1, ..., X_n) = \phi(X_1, X_2)\phi(X_2, X_3)...\phi(X_{n-1}, X_n)$ where X_n is the stacked variable for column n. We therefore need to find the potentials between the columns. The exception is column one as the column's potential is only dependent on the variables in its column.

  $\phi(X_{n-1}, X_n)$ is merely short hand for all the potentials between the variables for those columns in the lattice. To take an example, in the case where $n = 3$ and has the following lattice structure:

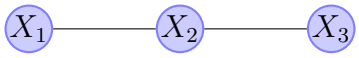


  The potential would be as follows:

  $\phi(X_2, X_3) = \phi(x_7, x_4)\phi(x_8, x_7)\phi(x_8, x_5)\phi(x_9, x_8)\phi(x_9, x_6)$

  Therefore we find the product of the potentials between the variables to find the potentials between the stacked columns. In the code (see the appendix) $\phi(X_2, X_3)$ would be stored in columnphi{3}.

- We can think of the singly connected graph below:



As a clique graph:

The normalization constant is then as follows:

$Z = \sum_x \prod_i \phi(X_i)$
This can be computed by the Sum-Product algorithm. In our simple 3 node graph this would be done as follows:

$Z = \sum_{X_3,X2} \phi(X_3, X_2) \sum_{X_1} \phi(X_2, X_1)$

The following is the message passed from node 1 to node 2: $\sum_{X_1} \phi(X_2, X_1)$
In the code in the appendix, we calculate the message and pass it on forwards.

Our calculated Log Z = 186.7916.

The operational complexity of the message passing is NS^2 as each message has S values and each value must be summed over S states for each of the N messages.

There are however 2^N possible states for each X_t as it is a cluster of n binary variables with a total of 2^N possible combinations of states. So, S = 2^N. Overall complexity is therefore O($N2^{2N}$)

ZH

Exercise 6.9. Part 1

```
%% Part 1
import brml.*
load diseaseNet.mat;
% Constructing a junction tree from jtree
[jtpot jtsep infostruct]=jtree(pot);


% Use the junction to compute all the marginals of the first symptoms, p(si =
% 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('Marginal probabilities calculated using junction tree approach:')



for i = 21:60

    % JT potential that contains symptom 'i'.
    jtpotnum = whichpot(jtpot,i,1);


    % find marginal probability of 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
```

Marginal 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

In the code here I construct a node selection algorithm that works faster than the junction tree algorithm.

```
%% Part 2

disp(' ') % new line
```

```
disp('Marginal 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);


    % From the conditional potential of the symptom, extract the potentials t
    % 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);

    % We will now check the form of analysis by reconstructing our
    % potentials with new variables.

    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;


% Now calculate the joint distribution
jointpot = multpots(new_pot); % joint distribution

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

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

end

The node selection algorithm used here is 50 percent faster than the junction tree algorithm used in part 1. The output results were the same.

Part 3

```
%% Part 3

disp(' ') % new line

disp('Marginal probabilities of diseases given evidential variables.')

% Symptoms 1 to 5 are present in (state 1) and symptoms 6 to 10 not
% present (state 2)
% Set up indices for evidential variables

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];

%any binary values would do
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);


% Propogate probabilities through absorption
[jtpot2, jtsep2, logZ2]=absorption(jtpot2,jtsep2,infostruct2);


%Transform back to original variables after absorption
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

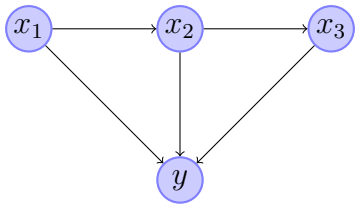
Marginal probabilities of diseases given evidential variables. p(d1=1 — s1:10) 0.029776
p(d2=1 — s1:10) 0.38176
p(d3=1 — s1:10) 0.95423
p(d4=1 — s1:10) 0.39664
p(d5=1 — s1:10) 0.49647
p(d6=1 — s1:10) 0.43515
p(d7=1 — s1:10) 0.18749
p(d8=1 — s1:10) 0.70118
p(d9=1 — s1:10) 0.043127
p(d10=1 — s1:10) 0.61031
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

SC

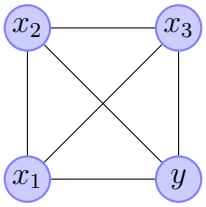
Exercise 6.10. 1. We are asked to draw a Junction tree of the distribution

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

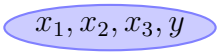
To illustrate how the Junction Tree is formed in the general case we take T = 3. Here the belief network for the distribution is given by:



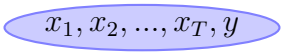
Moralisation and triangulation of this belief network yields the following graph:



From which we can draw the following Junction Tree, which only has one node since the maximal clique is the entire graph:



Since all nodes x_i for $i = 1, ..., T$ are parents of y the moralisation process will always lead to a fully connected graph. As such, in general the Junction Tree for $p(y|x_1, ..., x_T)p(x_1)\prod_{t=2}^{T} p(x_t|x_{t-1})$ is given by:



Similar to the case of disease-symptom networks given in the textbook, this means that clique sizes will rapidly become large at T increases and means that inference computations, including calculating $p(x_t)$ increase exponentially as t increases. Since the variables are binary, the time complexity is in $O(2^T)$

2. To compute $p(x_T)$ in linear time we can sum over y: $\sum_y p(y|x_1, ..., x_T)p(x_1)\prod_{t=2}^{T} p(x_t|x_{t-1})$

This gives us: $p(x_1)\prod_{t=2}^{T} p(x_t|x_{t-1})$ which is a simple linear markov chain for which marginal inference can be carried out in O(T) (as mentioned in lectures).

SM

Appendix

5.8 Modified version of the demoHMMInferenceSimple

```
function question 5.8
import brml.*
H = 83; % number of Hidden states
V = 26; % number of Visible states, letters in the alphabet
T = 10000; % length of the time-series
% setup the HMM

filename = 'noisystring.mat';
%Import variables
load(filename);

%Transition matrix
A = zeros(83,83);
A(1,1) = 0.8;

A(2,1) = 0.2/5;
A(7,1) = 0.2/5;
A(12,1) = 0.2/5;
A(16,1) = 0.2/5;
A(19,1) = 0.2/5;

A(3,2) = 1;
A(4,3) = 1;
A(5,4) = 1;
A(6,5) = 1;

A(8,7) = 1;
A(9,8) = 1;
A(10,9) = 1;
A(11,10) = 1;

A(13,12) = 1;
A(14,13) = 1;
A(15,14) = 1;

A(17,16) = 1;
A(18,17) = 1;

A(20,19) = 1;
A(21,20) = 1;
```

```
A(22,21) = 1;
A(23,22) = 1;

A(24,6) = 1;
A(24,11) = 1;
A(24,15) = 1;
A(24,18) = 1;
A(24,23) = 1;

A(24,24) = 0.8;

A(25,24) = 0.2/7;
A(31,24) = 0.2/7;
A(37,24) = 0.2/7;
A(40,24) = 0.2/7;
A(45,24) = 0.2/7;
A(56,24) = 0.2/7;
A(67,24) = 0.2/7;

A(26,25) = 1;
A(27,26) = 1;
A(28,27) = 1;
A(29,28) = 1;
A(30,29) = 1;

A(32,31) = 1;
A(33,32) = 1;
A(34,33) = 1;
A(35,34) = 1;
A(36,35) = 1;

A(38,37) = 1;
A(39,38) = 1;

A(41,40) = 1;
A(42,41) = 1;
A(43,42) = 1;
A(44,43) = 1;

A(46,45) = 1;
A(47,46) = 1;
A(48,47) = 1;
A(49,48) = 1;
A(50,49) = 1;
A(51,50) = 1;
```

```
A(52,51) = 1;
A(53,52) = 1;
A(54,53) = 1;
A(55,54) = 1;

A(57,56) = 1;
A(58,57) = 1;
A(59,58) = 1;
A(60,59) = 1;
A(61,60) = 1;
A(62,61) = 1;
A(63,62) = 1;
A(64,63) = 1;
A(65,64) = 1;
A(66,65) = 1;

A(68,67) = 1;
A(69,68) = 1;
A(70,69) = 1;
A(71,70) = 1;
A(72,71) = 1;
A(73,72) = 1;
A(74,73) = 1;
A(75,74) = 1;
A(76,75) = 1;
A(77,76) = 1;
A(78,77) = 1;
A(79,78) = 1;
A(80,79) = 1;
A(81,80) = 1;
A(82,81) = 1;
A(83,82) = 1;

A(1,30) = 1;
A(1,36) = 1;
A(1,39) = 1;
A(1,44) = 1;
A(1,55) = 1;
A(1,66) = 1;
A(1,83) = 1;

A; % Transition probability matrix

%Emission matrix
```

```
value_emission = 0.7/25;
B=ones(V,H)*value_emission; % Emssion probability

B(:,1) = 1/26;

B(:,24) = 1/26;

%State 1: D
B(4,2) = 0.3;
%State 2: A
B(1,3) = 0.3;
%State 3: V
B(22,4) = 0.3;
%State 4: I
B(9,5) = 0.3;
%State 5: D
B(4,6) = 0.3;


%State 6: A
B(1,7) = 0.3;
%State 7: N
B(14,8) = 0.3;
%State 8: T
B(20,9) = 0.3;
%State 9: O
B(15,10) = 0.3;
%State 10: N
B(14,11) = 0.3;


%State 11: F
B(6,12) = 0.3;
%State 12: R
B(18,13) = 0.3;
%State 13: E
B(5,14) = 0.3;
%State 14: D
B(4,15) = 0.3;


%State 15: J
B(10,16) = 0.3;
%State 16: I
B(9,17) = 0.3;
```

```
%State 17: M
B(13,18) = 0.3;


%State 18: B
B(2,19) = 0.3;
%State 19: A
B(1,20) = 0.3;
%State 20: R
B(18,21) = 0.3;
%State 21: R
B(18,22) = 0.3;
%State 22: Y
B(25,23) = 0.3;


%State 24: B
B(2,25) = 0.3;
%State 25: A
B(1,26) = 0.3;
%State 26: R
B(18,27) = 0.3;
%State 27: B
B(2,28) = 0.3;
%State 28: E
B(5,29) = 0.3;
%State 29: R
B(18,30) = 0.3;


%State 30: I
B(9,31) = 0.3;
%State 31: L
B(12,32) = 0.3;
%State 32: S
B(19,33) = 0.3;
%State 33: U
B(21,34) = 0.3;
%State 34: N
B(14,35) = 0.3;
%State 35: G
B(7,36) = 0.3;


%State 36: F
```

```
B(6,37) = 0.3;
%State 37: O
B(15,38) = 0.3;
%State 38: X
B(24,39) = 0.3;


%State 39: C
B(3,40) = 0.3;
%State 40: H
B(8,41) = 0.3;
%State 41: A
B(1,42) = 0.3;
%State 42: I
B(9,43) = 0.3;
%State 43: N
B(14,44) = 0.3;


%State 44: F
B(6,45) = 0.3;
%State 45: I
B(9,46) = 0.3;
%State 46: T
B(20,47) = 0.3;
%State 47: Z
B(26,48) = 0.3;
%State 48: W
B(23,49) = 0.3;
%State 49: I
B(9,50) = 0.3;
%State 50: L
B(12,51) = 0.3;
%State 51: L
B(12,52) = 0.3;
%State 52: I
B(9,53) = 0.3;
%State 53: A
B(1,54) = 0.3;
%State 54: M
B(13,55) = 0.3;


%State 55: Q
B(17,56) = 0.3;
```

```
%State 56: U
B(21,57) = 0.3;
%State 57: I
B(9,58) = 0.3;
%State 58: N
B(14,59) = 0.3;
%State 59: C
B(3,60) = 0.3;
%State 60: E
B(5,61) = 0.3;
%State 61: A
B(1,62) = 0.3;
%State 62: D
B(4,63) = 0.3;
%State 63: A
B(1,64) = 0.3;
%State 64: M
B(13,65) = 0.3;
%State 65: S
B(19,66) = 0.3;


%State 66: G
B(7,67) = 0.3;
%State 67: R
B(18,68) = 0.3;
%State 68: A
B(1,69) = 0.3;
%State 69: F
B(6,70) = 0.3;
%State 70: V
B(22,71) = 0.3;
%State 71: O
B(15,72) = 0.3;
%State 72: N
B(14,73) = 0.3;
%State 73: U
B(21,74) = 0.3;
%State 74: N
B(14,75) = 0.3;
%State 75: T
B(20,76) = 0.3;
%State 76: E
B(6,77) = 0.3;
```

```
%State 77: R
B(18,78) = 0.3;
%State 78: H
B(8,79) = 0.3;
%State 79: O
B(15,80) = 0.3;
%State 80: S
B(19,81) = 0.3;
%State 81: E
B(5,82) = 0.3;
%State 82: N
B(14,83) = 0.3;

%phghm = condp(rand(H,H)+3*eye(H));% transition distribution p(h(t)|h(t-1))
%pvgh = condp(rand(V,H).^5);% emission distribution p(v(t)|h(t))

phghm = A;
pvgh = B;

C=zeros(H,1);
C(1,1) = 1;

%ph1 = condp(ones(H,1)); % initial p(h)
ph1 = C;
size(ph1);

v = noisystring; % data
data = v-'a'+1;

% Perform Inference tasks:
[alpha,loglik]=HMMforward(data,phghm,ph1,pvgh); % forward
% smoothed posteriors:
gamma=HMMgamma(alpha,phghm); % alternative alpha-gamma (RTS) method
[viterbimaxstate logprob]=HMMviterbi(data,phghm,ph1,pvgh) % most likely joint

%numbers = [25,26,27,28,29,30];
Substitutions = {'$', 'D', 'A', 'V','I','D','A','N','T','O','N',
'F','R','E','D','J','I','M','B','A','R','R','Y','$','B','A','R',
'B','E','R','I','L','S','U','N','G','F','O','X','C','H','A','I',
'N','F','I','T','Z','W','I','L','L','I','A','M','Q','U','I','N',
'C','E','A','D','A','M','S','G','R','A','F','V','O','N','U','N',
'T','E','R','H','O','S','E','N'};
mapped = Substitutions(viterbimaxstate);
strjoin(mapped, {''})
```

Here is a sample of the output:

```
'$$$$$$$DAVID$$$$$$$$QUINCEADAMS$$$$DAVID$$$$$QUINCEADAMS$$$$$$
DAVID$$$$$$$$$$FITZWILLIAM$$$$$$$$$$$$$$$$$$$$$$$$$$$JIM$FITZWI
LLIAM$$$$$BARRY$$$$$$$$$$$$QUINCEADAMS$$$$$JIM$$$$$$FITZWILLIAM
$$$$$$$$$$$$$$$BARRY$$FITZWILLIAM$$$$$$$$$$$$$$$$$ANTON$$$$BARB
ER$$$$$$$DAVID$$$$$$$CHAIN$$$$$ANTON$$$$$$$$$ILSUNG$$$$$$$$$$JI
M$$$$$$$$$$$$$$$$ILSUNG$$$$$$$$$$$$ANTON$$$$$$GRAFVONUNTERHOSEN
$$$$$$$$$$$$$$$$$$$$$$$BARRY$$$$$$$$$$$$$$$$$$ILSUNG$$BARRY$$$$
$$$$$$$$CHAIN$$$$JIM$$$$FITZWILLIAM$$$$$DAVID$$CHAIN$$$$$ANTON$
$QUINCEADAMS$$$$BARRY$$$FITZWILLIAM$$$$$FRED$$$$$GRAFVONUNTERHO
SEN$$$$$$$$$FRED$$$CHAIN$$$FRED$GRAFVONUNTERHOSEN$$$$FRED$$$$$G
RAFVONUNTERHOSEN$$$$$$$ANTON$$$$$$$$$$$$$$QUINCEADAMS$$$$$FRE
D$$BARBER$$$DAVID$$$$$$FOX$$BARRY$$$$$$CHAIN$$$$$$$$$$JIM$$$$$
GRAFVONUNTERHOSEN$JIM$$FITZWILLIAM$$DAVID$$$$$$FITZWILLIAM$$$$$
$$ANTON$$$$$$$ILSUNG$$$DAVID$$$$$$$$$$$$$$$$$$GRAFVONUNTERHOSE
N$$$$$$$FRED$$GRAFVONUNTERHOSEN$$$$$FRED$$$$$$GRAFVONUNTERHOSEN
$$$$$$$$$$$$$$$ANTON$$$$$CHAIN$$$$$$$$$$$$$$$$$$$$$JIM$$$$$$$
FOX$$$$$JIM$$$$$$$$$$ILSUNG$$$$$$JIM$BARBER$$FRED$$$$$GRAFVONU
NTERHOSEN$$$$$$$$$$$$$$$$FRED$$$QUINCEADAMS$$$$$$$$$FRED$QUINCE
ADAMS$$$$$$$$$$$$$$$$$JIM$GRAFVONUNTERHOSEN$$$$JIM$FITZWILLIAM$$
$$$$$$BARRY$$$$$$$$$$$$$$$$$$$$CHAIN$$$$$JIM$GRAFVONUNTERHOSEN$
$$$$$$$$BARRY$$$$$$$$$$$$$$$$$$$$$$GRAFVONUNTERHOSEN$$$$$$$$$$
$$$$$$$ANTON$FITZWILLIAM$$$$$$$$DAVID$$QUINCEADAMS$$$$$$$$DAV
ID$$$$$FITZWILLIAM$$$$$$$$JIM$CHAIN$$JIM$$FOX$$$$$$$$ANTON$FITZ
WILLIAM$FRED$$$$FITZWILLIAM$$$FRED$$$$$BARBER$$$$BARRY$FOX$$BAR
RY$$$$$$GRAFVONUNTERHOSEN$$$$$FRED$FOX$$$$ANTON$FITZWILLIAM$$$$
DAVID$$$$$$$$$$$$$$$$$$$$CHAIN$$$$$$$ANTON$$$$$$$$$$$$GRAFVONUNT
ERHOSEN$$$$$$$$$$$$$$$$$$$$$$$FRED$CHAIN$$$$$$$$$$$$$$$$$$$BARRY
$$$$$$$$$$$$$$$$BARBER$$$$$$$$$DAVID$$$$$$$$$$$$$$$$$$$$BARBER$
$$$$$$$$$$DAVID$$$$$$$$$$$$$$$$$$$$$$$$$FITZWILLIAM$$$$$FRED$$
$$$$$$BARBER$$$$$$DAVID$$$$$$$$$$$$$FITZWILLIAM$$$$$$$$$$$$JIM
$$$BARBER$$$$BARRY$$$$$CHAIN$$$$DAVID$FOX$$$$$$$ANTON$BARBER$$$
DAVID$$$$$$$CHAIN$$$ANTON$$$$$$$$FITZWILLIAM$$$$$FRED$$$$$$$$$
$$$$CHAIN$$$JIM$$FITZWILLIAM$$$$$$$$$$$$$$$$JIM$$$$$$$$$QUINCEAD
AMS$$DAVID$$GRAFVONUNTERHOSEN$$$$$$$$JIM$FITZWILLIAM$$ANTON$$$
$$$$$$BARBER$$$$$$$JIM$$$$$$$$$$$$$$$$$$$$$FOX$$ANTON$$$$$$$$FO
X$$$$$JIM$$$$$$CHAIN$$$$BARRY$$$$$$$$$CHAIN$$$$$$$$$$$$$$$ANTON$ . . '
```

5.9
Code for assigning states to the grid:

```
states=zeros(50, 50);
i=1;
for sy=1:50
    for sx=1:50
        states(sy,sx)=i;
        i=i+1;
    end
end
```

Code for populating emissions matrix:

```
dx=50 dy=50
emissions=zeros(dx*dy,100);
for t=1:100
k=1;
    for ey=1:dy
        for ex=1:dx
            if data{t}(ex,ey)==1
                emissions(k,t)=1;
            else
                    emissions(k,t)=0;
            end
            k=k+1;
        end
    end
end
```

Code for populating transition matrix:

```
for dx=1:x
    for dy=1:y
        lastposition = states(dx,dy);

        if dx >= 3 && dx <= 48 && dy >= 3 && dy <= 48
            southeast = states(dx+2,dy+2);
            northeast = states(dx+2,dy-2);
            southwest = states(dx-2,dy+2);
            northwest = states(dx-2,dy-2);
            drunk(southeast,lastposition)=0.25;
            drunk(northeast,lastposition)=0.25;
            drunk(southwest,lastposition)=0.25;
            drunk(northwest,lastposition)=0.25;
```

```
elseif dx >= 1 && dx < 3 && dy >= 3 && dy <= 48
    southeast = states(dx+2,dy+2);
    northeast = states(dx+2,dy-2);
    drunk(:,lastposition)=(0.5/(x*y -2));
    drunk(southeast,lastposition)=0.25;
    drunk(northeast,lastposition)=0.25;


elseif dx > 48 && dx <= 50 && dy >= 3 && dy <= 48
    southwest = states(dx-2,dy+2);
    northwest = states(dx-2,dy-2);
    drunk(:,lastposition)=(0.5/(x*y -2));
    drunk(southwest,lastposition)=0.25;
    drunk(northwest,lastposition)=0.25;

elseif dx >= 3 && dx <= 47 && dy >= 1 && dy < 3
    southwest = states(dx-2,dy+2);
    southeast = states(dx+2,dy+2);
    drunk(:,lastposition)=(0.5/(x*y -2));
    drunk(southwest,lastposition)=0.25;
    drunk(southeast,lastposition)=0.25;

elseif dx >= 3 && dx <= 48 && dy > 48 && dy <= 50
    northwest = states(dx-2,dy-2);
    northeast = states(dx+2,dy-2);
    drunk(:,lastposition)=(0.5/(x*y -2));
    drunk(northwest,lastposition)=0.25;
    drunk(northeast,lastposition)=0.25;

elseif dx >= 1 && dx < 3 && dy >= 1 && dy < 3
    southeast = states(dx+2,dy+2);
    drunk(:,lastposition)=(0.75/(x*y -3));
    drunk(southeast,lastposition)=0.25;

elseif dx > 48 && dx <= 50 && dy >= 1 && dy < 3
    southwest = states(dx-2,dy+2);
    drunk(:,lastposition)=(0.75/(x*y -3));
    drunk(southwest,lastposition)=0.25;

elseif dx >= 1 && dx < 3 && dy > 48 && dy <= 50
    northeast = states(dx+2,dy-2);
    drunk(:,lastposition)=(0.75/(x*y -3));
    drunk(northeast,lastposition)=0.25;
```

```
        elseif dx > 48 && dx <= 50 && dy > 48 && dy <= 50
            northwest = states(dx-2,dy-2);
            drunk(:,lastposition)=(0.75/(x*y -3));
            drunk(northwest,lastposition)=0.25;
        end
    end
end
```

Code for populating initial distribution matrix:

```
initialdistribution = zeros(dx*dy,1);

total =(dx-4)*(dy-4);
for ix=3:48
    for iy=3:48
        s = states(ix,iy);
        initialdistribution(s,1) = 1/total;
    end
end
```

Code for modified Viterbi function:

```
function [maxstate logprob]=CustomHMMviterbi(phghm,ph1,pvgh)
%HMMVITERBI Viterbi most likely joint hidden state of a HMM
% [maxstate logprob]=HMMviterbi(v,phghm,ph1,pvgh)
%
% Inputs:
% phghm : homogeneous transition distribution phghm(i,j)=p(h(t)=i|h(t-1)=j)
% ph1 : initial distribution
% pvgh : homogeneous emission distribution pvgh(i,j)=p(v(t)=i|h(t)=j)
%
% Outputs:
% maxstate : most likely joint hidden (latent) state sequence
% logprob : associated log probability of the most likely hidden sequence
% See also demoHMMinference.m
import brml.*
T=size(pvgh,2); H=size(phghm,1);
mu(:,T)=ones(H,1);
for t=T:-1:2
    tmp = repmat(pvgh(:,t).*mu(:,t),1,H).*phghm;
    mtmp = max(tmp)';
    % message at time t:
    mu(:,t-1)= condp(max(tmp)'); % normalise to avoid underflow
end
% backtrack
```

```
[val hs(1)]=max(ph1.*pvgh(:,1).*mu(:,1)); %most likely state when t = 1
for t=2:T
    tmp = pvgh(:,t).*phghm(:,hs(t-1)); %additional message
    [val hs(t)]=max(tmp.*mu(:,t)); %additional message * message received at t
end
maxstate=hs;
logprob=log(ph1(hs(1)))+log(pvgh(hs(1,t)));
for t=2:T
    logprob=logprob+log(phghm(hs(t),hs(t-1)))+log(pvgh(hs(t),t));
end
end
```

5.10 Code used to calculate expected price gain and standard deviation:

```
%Inputs information from question
transition = [0.8,0.3;0.2,0.7];
ph1=[0.5;0.5];

%Generates a 200x2 matrix of p(v(t)|h(t),p(t-1) where entry i,j is
%p(v(t)=p(i)|h(t)=j,p(t-1))

pvghv = zeros(2,200);
pvghv(:,1) = 0.01;

for t=2:200
    pvghv(1,t) = pbear(p(t),p(t-1));
    pvghv(2,t) = pbull(p(t),p(t-1));
end
pvghv = pvghv';

%Implements filtering using altered HMMforward algorithm
alpha = HMMforwardbullbear(p,transition,ph1,pvghv);

%Computes p201 distribution
ph201 = transition*alpha(:,200);
p201bear = pbear(:,p(200))*ph201(1);
p201bull = pbull(:,p(200))*ph201(2);
p201 = p201bear + p201bull;
prices = (1:100);

%Gives expected price in time t = 201 of 59.4323
ep201 = prices*p201;

% Gives expected price gain
epgain = ep201 - p(200);

%Computes standard deviation of price gain

sqdev = ((prices - p(200))-epgain).^2;
sqdevp = sqdev'.* p201;
std = sqrt(sum(sqdevp));
```

The modified HMMforward algorithm is as follows:

```
function [alpha,loglik]=HMMforwardbullbear(v,phghm,ph1,pvgh)
%HMMFORWARD HMM Forward Pass
% [alpha,loglik]=HMMforward(v,phghm,ph1,pvgh)
%
% Inputs:
```

```
% v : visible (observation) sequence being a vector v=[2 1 3 3 1 ...]
% phghm : homogeneous transition distribution phghm(i,j)=p(h(t)=i|h(t-1)=j)
% ph1 : initial distribution
% CHANGED:
% pvgh :  emission distribution pvghv(ij) = p(v(t)=p(i)|h(t)=j,p(t-1))
%
% Outputs:
% alpha : alpha messages: p(h(t)|v(1:t))
% loglik : sequence log likelihood log p(v(1:T))
% See also HMMbackward.m, HMMviterbi.m,  HMMsmooth.m, demoHMMinference.m
import brml.*
T=length(v); H=length(ph1);
if 1==0
% logalpha recursion
logalpha(:,1) = log(pvgh(v(1),:)'.*ph1);
for t=2:T
        logalpha(:,t)=logsumexp(repmat(logalpha(:,t-1),1,H),repmat(pvgh(v(t),
end
loglik = logsumexp(logalpha(:,T),ones(H,1)); % log likelihood
end

% alpha recursion (with normalisation to avoid numerical underflow)
    alpha=zeros(H,T);
    z=zeros(1,T); % local normalisation factors
        alpha(:,1) = ph1; %Changed to ph1 since the probability
    %of the hidden state in t=1 is given
    z(1)=sum(alpha(:,1));
    alpha(:,1)=alpha(:,1)/z(1);
        for t=2:T
                alpha(:,t)=pvgh(t,:)'.*(phghm*alpha(:,t-1)); %Changed to pvgh
        %to account for our new input.
        z(t)=sum(alpha(:,t));
        alpha(:,t)=alpha(:,t)/z(t);
        end
        loglik = sum(log(z(:))); % log likelihood
end
```

6.7

```
import brml.*

N=10;

  %Calculate the potentials between the variables
 potentials = zeros(2,2);
 for x=1:2
     for y=1:2
         if x == y
         potentials(x,y)=exp(1);
         else
         potentials(x,y)=exp(0);
         end
     end
 end

 variables = zeros(N);
 i = 1;
 for y = 1:N
     for x=1:N
         variables(x,y) = i;
         i = i+1;
     end
 end
 c=0;

 %identify the pairs of neighbouring variables (where i>j)
 pair = 1;
 neighbours = zeros(N*N,N*N);
 for i = 1:N*N
      for j = 1:N*N
        if ((j == i-1) && (floor((i-1)/N) == floor((j-1)/N))) ||   (j==i-10)
            neighbours (i,j) = pair;
            phi{pair}=array([i j],potentials);
            pair = pair + 1;
        end
      end
 end

 %Calculate the potential of column 1
 %(variables only have potentials with variables above)
  columnphi{1}=const(1);
 for i=2:N
```

```
    variable = variables(i,1);
    intermid = neighbours(variable,:);
    neighbourpair = nonzeros(intermid');
    neighbour1 = neighbourpair(1);
    columnphi{1} = multpots([columnphi{1} phi{neighbour1}]);
end

%Calculate the potentials for the other columns
for j=2:N
    columnphi{j}=const(1);
    for i=2:N
        variable = variables(i,j);
        intermid = neighbours(variable,:);
        neighbourpair = nonzeros(intermid');
        neighbour1 = neighbourpair(1);
        neighbour2 = neighbourpair(2);
        columnphi{j}=multpots([columnphi{j} phi{neighbour1}]);
        columnphi{j}=multpots([columnphi{j} phi{neighbour2}]);
    end
    %below section is multiplying the potentials from the 1st row
    %they are handled separately as they also only have one potential
    %i.e. with the variable to their left in the lattice
    variable = variables(1,j);
    intermid = neighbours(variable,:);
    neighbourpair = nonzeros(intermid');
    neighbour1 = neighbourpair(1);
    columnphi{j}=multpots([columnphi{j} phi{neighbour1}]);

end

%message passing below
sum = setdiff(columnphi{1}.variables,intersect(columnphi{1}.variables,column
message = sumpot(columnphi{1},sum);

for i = 2:N-1
    sum = setdiff(columnphi{i}.variables,intersect(columnphi{i}.variables,co
    message = sumpot(multpots([message columnphi{i}]),sum);
end
Z = sumpot(multpots([message columnphi{N}]),[],0);
logZ = log(Z.table)
```

5.13 Full Planet list:
1 5 7 10 12 13 14 20 23 24 29 30 31 32 38 46 49 52 53 58 59 60 62 63 70 71 76 80 81 82 83 84 85 87 88 90 95 96 98 101 102 104 107 109 111 113 121 122 123 124 125 126 128 133 135 138 145 148 151 155 156 163 167 168 169 170 173 174 180 185 190 191 193 194 201 202 208 210 214 216 221 224 225 235 236 241 242 243 244 245 247 250 252 257 261 266 267 270 274 280 282 284 292 294 296 298 302 304 307 311 312 313 314 316 318 322 325 326 331 332 334 337 338 339 342 345 346 347 351 359 360 361 362 364 365 369 370 372 373 376 377 381 382 384 389 390 393 395 396 398 400 402 404 405 406 407 410 412 413 415 417 423 424 426 427 429 430 431 434 435 440 442 444 446 448 450 452 454 455 461 464 466 469 472 473 474 477 479 481 483 486 489 490 494 496 497 499 500 502 503 505 511 526 527 534 535 542 544 545 546 550 551 555 557 559 563 566 568 569 570 580 582 583 586 590 591 592 594 596 597 601 605 609 614 615 618 619 624 625 626 627 629 630 631 632 633 634 635 638 639 643 646 647 652 653 654 655 656 658 659 660 661 663 664 665 666 672 674 675 676 678 683 685 687 692 694 695 698 700 702 703 707 709 710 711 719 720 721 725 728 732 733 735 736 737 740 741 744 746 747 748 750 753 755 757 761 763 769 771 772 773 774 775 776 777 778 779 781 783 785 789 790 794 797 801 802 804 807 811 814 815 817 819 820 821 824 828 832 833 837 838 839 842 844 845 846 848 849 855 857 858 861 863 866 871 875 877 880 886 889 890 892 898 899 901 903 905 909 910 911 914 915 920 921 922 924 925 926 927 928 929 934 935 939 943 946 947 948 949 950 954 956 959 961 965 969 970 975 981 984 985 987 991 992 997 998 1002 1005 1012 1013 1014 1025 1026 1028 1031 1032 1037 1039 1049 1050 1051 1052 1058 1061 1063 1065 1066 1071 1072 1073 1076 1079 1085 1088 1091 1093 1095 1104 1105 1106 1108 1110 1111 1112 1113 1114 1115 1116 1120 1123 1125 1128 1129 1130 1134 1137 1140 1142 1148 1151 1152 1154 1162 1167 1168 1169 1176 1177 1178 1179 1180 1181 1184 1186 1188 1189 1193 1194 1196 1198 1200 1205 1208 1209 1210 1211 1212 1213 1214 1215 1217 1218 1219 1220 1221 1222 1224 1227 1229 1230 1232 1234 1237 1238 1239 1242 1246 1248 1251 1253 1254 1255 1258 1260 1261 1265 1266 1269 1271 1273 1274 1275 1278 1279 1280 1286 1289 1293 1294 1295 1296 1299 1302 1304 1305 1308 1309 1312 1314 1320 1322 1325 1326 1335 1336 1337 1339 1340 1341 1344 1345 1347 1352 1354 1358 1360 1361 1366 1368 1372 1375 1376 1381 1386 1389 1391 1392 1394 1395 1396 1397 1399 1400 1409 1414 1415 1416 1417 1422 1425 1426 1428 1430 1434 1437 1439 1440 1441 1444 1448 1451 1452 1454 1455 1456 1459 1460 1464 1466 1467 1468 1469 1475 1477 1480 1481 1483 1484 1485 1486 1488 1493 1497 1499 1500 1501 1502 1503 1505 1510 1512 1514 1515 1518 1523 1524 1539 1543 1544 1546 1553 1554 1557 1564 1565 1568 1569 1571 1572 1581 1586 1595 1596 1597 1599 1600 1601 1602 1606 1608 1609 1610 1613 1614 1616 1617 1618 1619 1620 1623 1624 1625 1626 1629 1630 1633 1634 1635 1636 1640 1641 1645 1646 1647 1650 1651 1654 1655 1657 1659 1663 1664 1665 1668 1672 1675 1677 1678 1680 1682 1684 1696 1698 1699 1701 1704 1707 1709 1711 1712 1713 1715 1719 1721 1722 1723 1725

Code for 5.13:

```
function demoMostProbablePathMult

import brml.*
data = load newSimohurtta.mat;
A = data["A_prime"]
p=A./repmat(sum(A),size(A,1),1);
N=size(A,1);
[optpathmult,logprobmult]=mostprobablepathmult(log(p));
for startstate=1:N
    for endstate=1:N
        [optpath logprob]=mostprobablepath(log(p),startstate,endstate);
        if isfinite(logprob)
            s1= noselfpath(squeeze(optpathmult(startstate,endstate,:)),');
        end
    end
end
```