Graphical Models - Assignment 1

James Gin, John Goodacre, Christopher Loy, Mark Neumann

November 2015

Division of work

We initially divided the questions up as follows:

- C Loy: 3.20, 3.24, 4.16
- J Gin: 3.21, 3.25, 5.10
- J Goodacre: 3.22, 3.26, 5.11
- M Neumann: 3.23, 4.13, 5.12
- Unassigned: 5.13

This allowed us to start on questions individually, however almost every solution given below involved some collaboration between two or more members of the group, and the final submission has been agreed between all members to represent a collaborative effort.

Problem 3.20

The code listed below calculates two quantities:

$$\begin{array}{c} p(w,h|inc) - p(w|inc)p(h|inc) \\ p(w,h) - p(w)p(h) \end{array}$$

The former is found to be zero (floating point errors notwithstanding), implying that $h \perp\!\!\!\perp w|inc$.

The second quantity is substantially non-zero, indicating that w and h are not marginally independent.

```
function ex_3_20

import brml.*

% setup
wife = 1; husband = 2; income = 3;
```

```
cheap1 = 1; cheap2 = 2; expensive1 = 3; expensive2 = 4;
high = 1; low = 2;

cars = {'cheap1','cheap2', 'expensive1', 'expensive2'};

variable(wife).name = 'wife';
variable(wife).domain = cars;

variable(husband).name = 'husband';
variable(husband).domain = cars;

variable(income).name = 'income';
variable(income).domain = {'high', 'low'};

% known quantities
pot{income} = array;
pot{income}.variables = income;
pot{income}.table(high) = 0.8;
pot{income}.table(low) = 0.2;

pot{wife} = array;
pot{wife}.variables = [income wife];
pot{wife}.table(low, cheap1) = 0.7;
pot{wife}.table(low, cheap2) = 0.3;
pot{wife}.table(low, expensive1) = 0.0;
pot{wife}.table(low, expensive2) = 0.0;
pot{wife}.table(high, cheap1) = 0.2;
pot{wife}.table(high, cheap2) = 0.1;
pot{wife}.table(high, expensive1) = 0.4;
pot{wife}.table(high, expensive2) = 0.3;

pot{husband} = array;
pot{husband}.variables = [income husband];
pot{husband}.table(low, cheap1) = 0.2;
pot{husband}.table(low, cheap2) = 0.8;
pot{husband}.table(low, expensive1) = 0.0;
pot{husband}.table(low, expensive2) = 0.0;
pot{husband}.table(high, cheap1) = 0.0;
pot{husband}.table(high, cheap2) = 0.0;
pot{husband}.table(high, expensive1) = 0.3;
pot{husband}.table(high, expensive2) = 0.7;

% calculations
pi   = pot{income};  % p(income)
phgi = pot{husband}; % p(husband | income)
pwgi = pot{wife};    % p(wife | income)

pwhi  = multpots(pot);               % p(wife, husband, income)
pwh   = sumpot(pwhi, income);        % p(wife, husband)
pw    = sumpot(pwhi, [income husband]); % p(wife)
```

```
ph    = sumpot(pwhi, [income wife]); % p(husband)

% p(wife, husband | income)
pwhgi = condpot(setpot(pwhi, income, high), [wife husband]);

for cond = [high low]

  % p(wife, husband | income)
  pwhgi = condpot(setpot(pwhi, income, cond), [wife husband]);

  % p(husband | income)
  phgi = condpot(setpot(pot{husband}, income, cond));

  % p(wife | income)
  pwgi = condpot(setpot(pot{wife}, income, cond));
  product_conditional = (pwgi * phgi);

  % In both cases, this matrix is zero (excluding floating point errors)
  % thus showing that wife and husband are independent given income i.e.
  % p(wife,husband|income) = p(wife|income)p(husband|income)
  conditional_independence = pwhgi.table - product_conditional.table;
end

product_general = (pw * ph);

% This matrix is non-zero, showing that in general
% p(wife, husband) != p(wife)p(husband)
general_independence = pwh.table - product_general.table;
```

Problem 3.21

Draw a belief network that represents the independence assumptions above.

Are the skill levels of the players a posteriori (i.e. given the game outcomes) independent?

It is clear that all the skill factors involved in a match necessarily have common child nodes within the belief network, and are thus joined. As part of the conditioning set, all game state variables have their neighbouring connections severed and directed connections changed to undirected. As all players have played at least one other player in the set of players, this new graph connects the skill variable nodes. Therefore, conditioned on the games, the skill variables are conditionally dependent.

Calculate the probability that player D will beat player A in a game of BinGame

The graph represents the joint probability distribution:

$$P(Joint) = P(S_A, S_B, S_C, S_D, g_{AB}, g_{BC}, g_{CD}, g_{AC}, g_{AD})$$

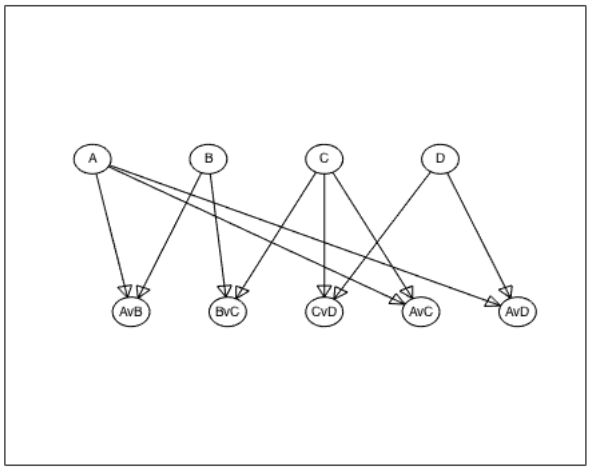

Figure 1: (A,B,C,D) represent the hidden skill levels S_A, S_B, S_C, S_D, and variables in the form XvY represents the resulting score state of games played between player X and player Y.

$$P(Joint) = P(S_A)P(S_B)P(S_C)P(S_D)P(g_{AB}|S_A, S_B)P(g_{BC}|S_B, S_C) \\ P(g_{CD}|S_C, S_D)P(g_{AC}|S_A, S_C)P(g_{AD}|S_A, S_D) \quad (1)$$

As we are interested in the quantity $P(g_{AD})$ given the states of all the games, we see that from Bayes Rule:

$$P(g_{AD}|\hat{g}) = P(g_{AD}|g_{AB} = (2,0), g_{BC} = (2,0), g_{CD} = (2,0), g_{AC} = (2,1)) \quad (2)$$

$$P(g_{AD}|\hat{g}) = \frac{P(g_{AD}, \hat{g})}{P(\hat{g})} = \frac{\sum_{\mathbf{S}} P(Joint|\hat{g})}{\sum_{\mathbf{S}, g_{AD}} P(Joint|\hat{g})} \quad (3)$$

$$P(g_{AD}|\hat{g}) = \frac{\sum_{\mathbf{S}} P(Joint|\hat{g})}{\sum_{\mathbf{S}} P(Joint|\hat{g}, g_{AD} = (1,0)) + \sum_{\mathbf{S}} P(Joint|\hat{g}, g_{AD} = (0,1))} \quad (4)$$

Where $\mathbf{S}$ is the set of all skill variables. We know the following relationship for $P(g_{XY})$

$$P(XbeatsY) = \frac{1}{1 + e^{S_Y - S_X}} \quad (5)$$

Therefore we can represent the scorelines between players as a binomial distribution:

$$P(g_{XY} = (x, y)) = \binom{x+y}{x} \left(\frac{1}{1+e^{S_Y - S_X}}\right)^x \left(\frac{1}{1+e^{S_X - S_Y}}\right)^y \tag{6}$$

The binomial coefficient reflects the number of alternative ways that the scoreline can be reached in the total number of games - for all given values this number is 1 apart from the games between A and C where the score is (2,1), thus giving a binomial coefficient multiplier of $\binom{3}{1} = 3$.

Therefore when evaluating Eq. 4 using the following code, we arrive at a value of **0.045**

```
dbeata = 0;
ss = zeros(10,10);
for a = 1:10
    for b = 1:10
        for c = 1:10
            for d = 1:10
                p = (1 / (1 + exp(b - a)))^2;
                p = p*(1 / (1 + exp(c - b)))^2;
                p = p*(1 / (1 + exp(d - c)))^2;
                p = p*3 * ((1 / (1 + exp(c - a)))^2*(1 / (1 + exp(a -
                    c))));
                p = p*(1 / (1 + exp(a - d)));
                dbeata = dbeata + p;
            end
        end
    end
end

abeatd = 0;
ss = zeros(10,10);
for a = 1:10
    for b = 1:10
        for c = 1:10
            for d = 1:10
                p = (1 / (1 + exp(b - a)))^2;
                p = p*(1 / (1 + exp(c - b)))^2;
                p = p*(1 / (1 + exp(d - c)))^2;
                p = p*3 * ((1 / (1 + exp(c - a)))^2*(1 / (1 + exp(a -
                    c))));
                p = p*(1 / (1 + exp(d - a)));
                abeatd = abeatd + p;
            end
        end
    end
end

dbeata / (abeatd+dbeata)
```

Calculate the expected skill level of each of the 4 players. By the same inference:

$$P(S_i = x|\hat{g}) = \frac{P(S_i = x, \hat{g})}{P(\hat{g})} = \frac{\sum_{\mathbf{S}_{i\neq j}} P(Joint|\hat{g}, S_i = x)}{\sum_{\mathbf{S}} P(Joint|\hat{g})} \tag{7}$$

And the definition of expected value being:

$$E(S_i) = \sum_{x \in \mathbf{dom}(S_i)} xP(S_i = x) \tag{8}$$

Therefore the following code calculates the expected values of each variable $\mathbf{S}_i$ as $S_A = 7.8671, S_B = 6.8493, S_C = 5.9844, S_D = 3.0461$:

```
denom = 0;
pa = zeros(10,1);
pb = zeros(10,1);
pc = zeros(10,1);
pd = zeros(10,1);
for a = 1:10
    for b = 1:10
        for c = 1:10
            for d = 1:10
                p = (1 / (1 + exp(b - a)))^2;
                p = p*(1 / (1 + exp(c - b)))^2;
                p = p*(1 / (1 + exp(d - c)))^2;
                p = p*3 * ((1 / (1 + exp(c - a)))^2*(1 / (1 + exp(a -
                    c))));
                denom = denom + p;
                pa(a) = pa(a) + p;
                pa(a) = pa(a) + p;
                pb(b) = pb(b) + p;
                pc(c) = pc(c) + p;
                pd(d) = pd(d) + p;
            end
        end
    end
end

ea = (pa / denom)' * (1:10)'
eb = (pb / denom)' * (1:10)'
ec = (pc / denom)' * (1:10)'
ed = (pd / denom)' * (1:10)'
```

Problem 3.22

Assuming that the interest levels are a priori independent and uniform and that each viewer clicks on adverts independently of the oth-

ers (given the advert interest levels), calculate the expected interest levels in each advert.

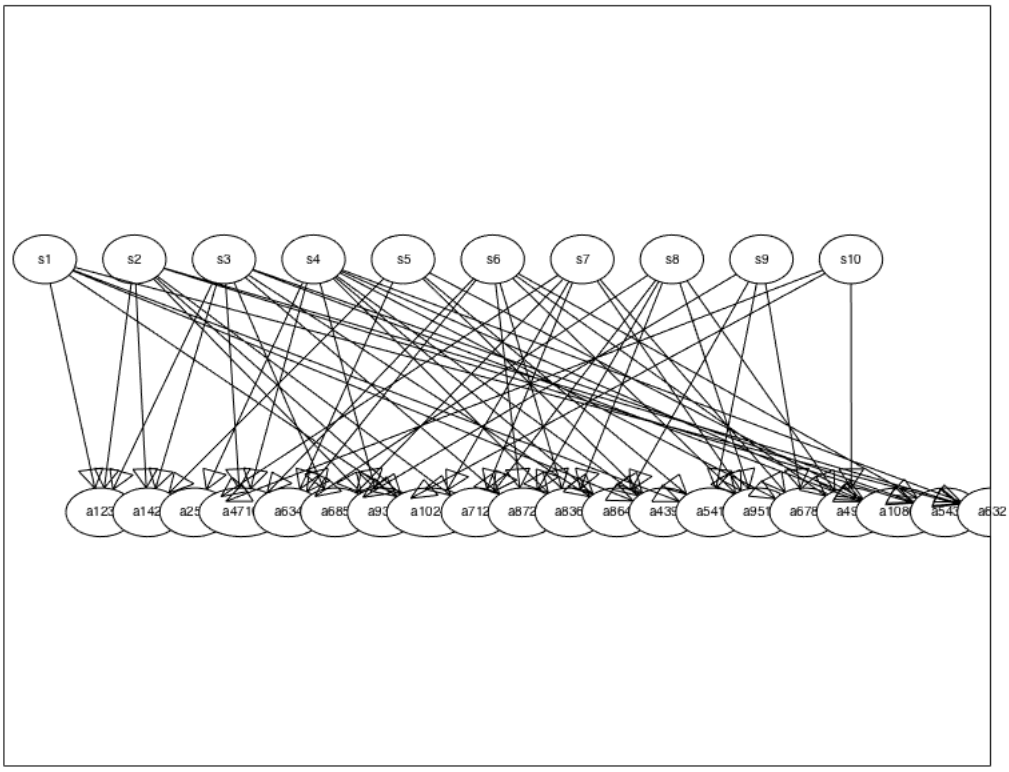

Figure 2: Full bayesian network for the daily fail

The full bayesian network for this problem is shown below. s1-s10 represent the probabilities of the interest levels of each advert. aijk are conditional probability tables dependent upon s_i, s_j, s_k

The question has many similarities to the previous question, (interest levels being analogous to skill levels and click events being 'contests' albeit between three 'players'). Thus the answer, despite the complicated code, is a similar implementation to the code for the previous question.

One difference is we are given a proportion, not a probability. I have thus assumed that for each 'contest' we need to divide by the probability that any of the other adverts could have been clicked to create a probability. Given that one of the adverts must be clicked, then this sums to one. To clarify with a

simple example. I have said that for a click contest involving adverts 1,2,3 - the result say advert 1 clicked in favour of adverts 2 or 3 (where of course we are conditionally dependent on interest levels for adverts 1,2,3) is a probability formed by taking the proportion given and dividing by all the possibilities. (ie (123)+(213)+(312)).

The graph represents the joint probability distribution

$$p(joint) = p(s1, s2, s3, .., s10, a123, a243, ..., a142)$$

$$p(joint) = p(s1)p(s2)...p(s10)* \\ p(a123|s1, s2, s3)p(a243|s2, s4, s3)...p(a142|s1, s4, s2) \quad (9)$$

We are told that

$$P(a(ijk)|s_i, s_j, s_k) \propto e^{S_i - max(S_j, S_k)} \quad (10)$$

Reframing as a probability this is:

$$P(a(ijk)|s_i, s_j, s_k) = \frac{e^{S_i - max(S_j, S_k)}}{e^{S_i - max(S_j, S_k)} + e^{S_j - max(S_i, S_k)} + e^{S_k - max(S_i, S_j)}} \quad (11)$$

After the following set of click events - (1, 2, 3)(2, 5, 3)(4, 7, 10)(6, 3, 4)(6, 8, 5)(9, 3, 7)(10, 2, 4)(7, 1, 2)(8, 7, 2)(8, 3, 6) (8, 6, 4)(4, 3, 9)(5, 4, 1)(9, 5, 1)(6, 7, 8)(4, 9, 7)(10, 8, 6)(5, 4, 3)(6, 3, 2)(1, 4, 2)

Calculate the expected interest level of each of the 10 adverts.

$$P(S_i = l|\hat{a}) = \frac{P(S_i = l, \hat{a})}{P(\hat{a})} = \frac{\sum_{\mathbf{S}_{i \neq j}} P(Joint|\hat{a}, S_i = l)}{\sum_{\mathbf{S}} P(Joint|\hat{a})} \quad (12)$$

And the definition of expected value being:

$$E(S_i) = \sum_{l \in \mathbf{dom}(S_i)} lP(S_i = l) \quad (13)$$

Where $\hat{a}$ represents the advert click data and l an interest level from 1-5

The code below calculates the expected values of each advert interest level $\mathbf{S}_i$ as $S_1 = 3.2090, S_2 = 2.6287, S_3 = 1.6506, S_4 = 3.2460, S_5 = 3.2509, S_6 = 4.3784, S_7 = 2.5463, S_8 = 4.3545, S_9 = 3.2049, S_{10} = 3.9502$

Code Explanation : I understand matlab doesn't like for loops and certainly not horribly deep nested ones. Apologies, this is clearly a brute force approach and there are no doubt better ways. The nested loops are simply marginalising over all the variables bar the outermost one, the outside loop is the variable we are trying to calculate. To enable reading of the code in latex I have only indented once. The code calculates all values given above but for the reader's sanity I have shown only the code for s1 and s2. The code repeats where we simply pull si to the outermost loop for each i.

Within the innermost loop we have our probability of each advert being clicked given the skill levels.

At the end we have posterior probabilities of the interest levels for each advert given the data. The code then turns this into an expectation in the usual way.

```
function df

denom = 0;
ps1 = zeros(5,1);
ps2 = zeros(5,1);

for s1 = 1:5
for s2 = 1:5
for s3 = 1:5
for s4 = 1:5
for s5 = 1:5
for s6 = 1:5
for s7 = 1:5
for s8=1:5
for s9=1:5
for s10=1:5
  p =
      exp(s1-max(s2,s3))/(exp(s1-max(s2,s3))+exp(s2-max(s1,s3))+exp(s3-max(s1,s2)));
  p =
      p*exp(s2-max(s5,s3))/(exp(s2-max(s5,s3))+exp(s5-max(s2,s3))+exp(s3-max(s5,s2)));
  p =
      p*exp(s4-max(s7,s10))/(exp(s4-max(s7,s10))+exp(s7-max(s4,s10))+exp(s10-max(s4,s7)));
  p =
      p*exp(s6-max(s3,s4))/(exp(s6-max(s3,s4))+exp(s3-max(s6,s4))+exp(s4-max(s3,s6)));
  p =
      p*exp(s6-max(s8,s5))/(exp(s6-max(s8,s5))+exp(s8-max(s6,s5))+exp(s5-max(s6,s8)));
  p =
      p*exp(s9-max(s3,s7))/(exp(s9-max(s3,s7))+exp(s3-max(s9,s7))+exp(s7-max(s9,s3)));
  p =
      p*exp(s10-max(s2,s4))/(exp(s10-max(s2,s4))+exp(s2-max(s4,s10))+exp(s4-max(s2,s10)));
  p =
      p*exp(s7-max(s1,s2))/(exp(s7-max(s1,s2))+exp(s1-max(s7,s2))+exp(s2-max(s7,s1)));
  p =
      p*exp(s8-max(s7,s2))/(exp(s8-max(s7,s2))+exp(s7-max(s8,s2))+exp(s2-max(s7,s8)));
  p =
      p*exp(s8-max(s3,s6))/(exp(s8-max(s3,s6))+exp(s3-max(s8,s6))+exp(s6-max(s3,s8)));
  p =
      p*exp(s8-max(s6,s4))/(exp(s8-max(s6,s4))+exp(s6-max(s8,s4))+exp(s4-max(s8,s6)));
  p =
      p*exp(s4-max(s3,s9))/(exp(s4-max(s3,s9))+exp(s3-max(s4,s9))+exp(s9-max(s4,s3)));
  p =
      p*exp(s5-max(s4,s1))/(exp(s5-max(s4,s1))+exp(s4-max(s5,s1))+exp(s1-max(s5,s4)));
  p =
```

```
        p*exp(s9-max(s5,s1))/(exp(s9-max(s5,s1))+exp(s5-max(s9,s1))+exp(s1-max(s9,s5)));
    p =
        p*exp(s6-max(s7,s8))/(exp(s6-max(s7,s8))+exp(s7-max(s6,s8))+exp(s8-max(s6,s7)));
    p =
        p*exp(s4-max(s9,s7))/(exp(s4-max(s9,s7))+exp(s9-max(s4,s7))+exp(s7-max(s4,s9)));
    p =
        p*exp(s10-max(s8,s6))/(exp(s10-max(s8,s6))+exp(s8-max(s10,s6))+exp(s6-max(s10,s8)));
    p =
        p*exp(s5-max(s4,s3))/(exp(s5-max(s4,s3))+exp(s4-max(s5,s3))+exp(s3-max(s4,s5)));
    p =
        p*exp(s6-max(s3,s2))/(exp(s6-max(s3,s2))+exp(s3-max(s6,s2))+exp(s2-max(s6,s3)));
    p =
        p*exp(s1-max(s4,s2))/(exp(s1-max(s4,s2))+exp(s4-max(s1,s2))+exp(s2-max(s1,s4)));
    ps1(s1) = ps1(s1) + p;
    denom = denom + p;
end
end
end
end
end
end
end
end
end
end

for s2 = 1:5
for s1 = 1:5
for s3 = 1:5
for s4 = 1:5
for s5 = 1:5
for s6 = 1:5
for s7 = 1:5
for s8=1:5
for s9=1:5
for s10=1:5


    p =
        exp(s1-max(s2,s3))/(exp(s1-max(s2,s3))+exp(s2-max(s1,s3))+exp(s3-max(s1,s2)));
    p =
        p*exp(s2-max(s5,s3))/(exp(s2-max(s5,s3))+exp(s5-max(s2,s3))+exp(s3-max(s5,s2)));
    p =
        p*exp(s4-max(s7,s10))/(exp(s4-max(s7,s10))+exp(s7-max(s4,s10))+exp(s10-max(s4,s7)));
    p =
        p*exp(s6-max(s3,s4))/(exp(s6-max(s3,s4))+exp(s3-max(s6,s4))+exp(s4-max(s3,s6)));
    p =
        p*exp(s6-max(s8,s5))/(exp(s6-max(s8,s5))+exp(s8-max(s6,s5))+exp(s5-max(s6,s8)));
    p =
        p*exp(s9-max(s3,s7))/(exp(s9-max(s3,s7))+exp(s3-max(s9,s7))+exp(s7-max(s9,s3)));
```

```
    p =
        p*exp(s10-max(s2,s4))/(exp(s10-max(s2,s4))+exp(s2-max(s4,s10))+exp(s4-max(s2,s10)));
    p =
        p*exp(s7-max(s1,s2))/(exp(s7-max(s1,s2))+exp(s1-max(s7,s2))+exp(s2-max(s7,s1)));
    p =
        p*exp(s8-max(s7,s2))/(exp(s8-max(s7,s2))+exp(s7-max(s8,s2))+exp(s2-max(s7,s8)));
    p =
        p*exp(s8-max(s3,s6))/(exp(s8-max(s3,s6))+exp(s3-max(s8,s6))+exp(s6-max(s3,s8)));
    p =
        p*exp(s8-max(s6,s4))/(exp(s8-max(s6,s4))+exp(s6-max(s8,s4))+exp(s4-max(s8,s6)));
    p =
        p*exp(s4-max(s3,s9))/(exp(s4-max(s3,s9))+exp(s3-max(s4,s9))+exp(s9-max(s4,s3)));
    p =
        p*exp(s5-max(s4,s1))/(exp(s5-max(s4,s1))+exp(s4-max(s5,s1))+exp(s1-max(s5,s4)));
    p =
        p*exp(s9-max(s5,s1))/(exp(s9-max(s5,s1))+exp(s5-max(s9,s1))+exp(s1-max(s9,s5)));
    p =
        p*exp(s6-max(s7,s8))/(exp(s6-max(s7,s8))+exp(s7-max(s6,s8))+exp(s8-max(s6,s7)));
    p =
        p*exp(s4-max(s9,s7))/(exp(s4-max(s9,s7))+exp(s9-max(s4,s7))+exp(s7-max(s4,s9)));
    p =
        p*exp(s10-max(s8,s6))/(exp(s10-max(s8,s6))+exp(s8-max(s10,s6))+exp(s6-max(s10,s8)));
    p =
        p*exp(s5-max(s4,s3))/(exp(s5-max(s4,s3))+exp(s4-max(s5,s3))+exp(s3-max(s4,s5)));
    p =
        p*exp(s6-max(s3,s2))/(exp(s6-max(s3,s2))+exp(s3-max(s6,s2))+exp(s2-max(s6,s3)));
    p =
        p*exp(s1-max(s4,s2))/(exp(s1-max(s4,s2))+exp(s4-max(s1,s2))+exp(s2-max(s1,s4)));
    ps2(s2) = ps2(s2) + p;


end
end
end
end
end
end
end
end
end
end

%note that this works in this specific case where there is ad click data
%for each skill level - else i would have to adjust for non updated
%probabilities and set them uniform

ps1 / denom
es1 = (ps1 / denom)' * (1:5)'
ps2 / denom
es2 = (ps2 / denom)' * (1:5)'
```

Problem 3.23

Below is an implementation of Q 2.23 in Matlab, used to compute all numerical answers in this question and provide reasoning behind the conclusions.

- Q1: $P(x^1_{2:T}|x^2_{1:T-1}) = 2.1871 \times 10^{-10}$
- Q2: How much more likely than random is it that P1 plays according to this strategy? 6.8633 times more likely.
- Q3: Calculate the probabilities for Rock, Paper and Scissors for P1 at time T + 1. P(Rock) = 0.75, P(Paper) = 0.25, P(Scissors) = 0.0.
- Q4: What would be the best move for player 2 to make at time T+1? Player 2 should play Paper.

```
close all
import brml.*

rock = 1
paper = 2
scissors = 3
% Index joint moves: eg PR21 = Player 1 plays Paper, Player 2 plays Rock
RR11 = 1;
PP22 = 2;
SS33 = 3;
RP12 = 4;
RS13 = 5;
SP32 = 6;
PR21 = 7;
SR31 = 8;
PS23 = 9;

% Define sequences of moves and index combined seqs as above
Player1 = [1,2,1,2,1,3,2,1,3,2,2,1,1,1,1,2,1,3,1,2,2,3,1];
Player2 = [3,2,1,3,2,3,2,3,1,2,3,1,2,2,1,1,3,2,1,3,3,2,1];
combo =  [5,2,1,9,4,3,2,5,8,2,9,1,4,4,1,7,5,6,1,9,9,6,1];


% fit a MM based on both player's prev moves
P=zeros(9,3);
t=24;
for i=2:t-1
    P(combo(i-1),Player1(i))=P(combo(i-1),Player1(i)) + 1; % count the
        number of transitions
```

```
    Pred=condp(P);
    prediction(:,i-1)=Pred(:,Player1(i-1));
end


% Find the probability of observing the chain
ChainProb = 1;
for x = 2:t-2 % index up to t-2 as prediction array is initialised at t=2
    ChainProb = ChainProb*prediction(combo(x-1),x);
end

% ChainProb = 2.1871 *10^(-10)

% Calc probability of randomly generating Player 1's sequence
RandomProb = (1/3)^22; % = 3.1866 *10^(-11)

Q1Ans = ChainProb/RandomProb; % = 6.8633



%sum rows corresponding to a particular move for player 1
RockPred = prediction(1,:) + prediction(4,:) + prediction(5,:);
PaperPred = prediction(2,:) + prediction(7,:) + prediction(9,:);
ScissorsPred = prediction(3,:) + prediction(6,:) + prediction(8,:);
FinalPred = [RockPred;PaperPred;ScissorsPred];


% Find probabilities for timestep T+1 for each possible move
RockProb = FinalPred(1,22); % = 0.75
PaperProb = FinalPred(2,22); % = 0.25
ScissorsProb = FinalPred(3,22); % = 0

% Best move for player 2 would be Paper as Paper beats Rock, and Player 1
% plays Rock with probability 0.75.
```

Problem 3.24

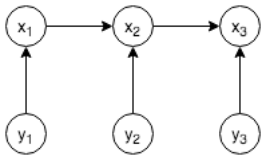

Figure 3: Belief network for $p(x_1|y_1)p(y_1)p(y_2)p(x_2|x_1,y_2)p(y_3)p(x_3|x_2,y_3)$.

Part 1

Given that:

$$p_1(x_1, y_1, x_2, y_2, x_3, y_3) = p(x_1|y_1)p(y_1)p(y_2)p(x_2|x_1, y_2)p(y_3)p(x_3|x_2, y_3)$$

Show that this can be written as:

$$p(x_1, y_1)p(x_2, y_2|x_1, y_1)p(x_3, y_3|x_2, y_2)$$

By inspecting the belief network shown in Figure 3, we can assert the following marginal independence statements:

$$\{y_1 \perp\!\!\!\perp y_2, y_2 \perp\!\!\!\perp y_3, x_1 \perp\!\!\!\perp y_2, x_2 \perp\!\!\!\perp y_3\}$$

We can also assert the following conditional independence statements:

$$\{y_1 \perp\!\!\!\perp x_2|x_1, y_2 \perp\!\!\!\perp x_3|x_2\}$$

Remembering that:

$$p(a, b|c) = \frac{p(a, b, c)}{p(c)} = \frac{p(a|b, c)p(b, c)}{p(c)} = p(a|b, c)p(b|c) \tag{14}$$

And that, if $a \perp\!\!\!\perp c|b$:

$$p(a|b, c) = \frac{p(a, b, c)}{p(b, c)} = \frac{p(a, c|b)p(b)}{p(c|b)p(b)} = \frac{p(a|b)p(c|b)}{p(c|b)} = p(a|b) \tag{15}$$

We can use (14) to rearrange the joint distribution into the following form:

$$\begin{aligned} p'(x_1, y_1, x_2, y_2, x_3, y_3) &= p(x_1, y_1)p(x_2, y_2|x_1, y_1)p(x_3, y_3|x_2, y_2) \\ &= p(x_1, y_1)p(x_2|y_2, x_1, y_1)p(y_2|x_1, y_1)p(x_3|y_3, x_2, y_2)p(y_3|x_2, y_2) \\ &= p(x_1|y_1)p(y_1)p(x_2|y_2, x_1, y_1)p(y_2|x_1, y_1)p(x_3|y_3, x_2, y_2)p(y_3|x_2, y_2) \end{aligned}$$

By applying (15) we can cancel redundant conditioning terms:

$$\begin{aligned} y_1 \perp\!\!\!\perp y_2, x_1 \perp\!\!\!\perp y_2 &\implies p(y_2|x_1, y_1) = p(y_2) \\ y_2 \perp\!\!\!\perp y_3, x_2 \perp\!\!\!\perp y_3 &\implies p(y_3|x_2, y_2) = p(y_3) \\ y_1 \perp\!\!\!\perp x_2|x_1 &\implies p(x_2|y_2, x_1, y_1) = p(x_2|x_1, y_2) \\ y_2 \perp\!\!\!\perp x_3|x_2 &\implies p(x_3|y_3, x_2, y_2) = p(x_3|x_2, y_3) \end{aligned}$$

Therefore we are left with the original form of $p1$:

$$\begin{aligned} p' &= p(x_1|y_1)p(y_1)p(x_2|x_1, y_2)p(y_2)p(x_3|x_2, y_3)p(y_3) \\ &= p_1(x_1, y_1, x_2, y_2, x_3, y_3) \end{aligned} \tag{16}$$

Part 2

Show that the distribution can be written as:

$p_2(x_1, y_1, x_2, y_2, x_3, y_3) = p(x_3, y_3)p(x_2, y_2|x_3, y_3)p(x_1, y_1|x_2, y_2)$

This can be straightforwardly proven using Bayes' rule and cancelling terms:

$$\begin{aligned}
&= p(x_3, y_3)\frac{p(x_3, y_3|x_2, y_2)p(x_2, y_2)}{p(x_3, y_3)}\frac{p(x_2, y_2|x_1, y_1)p(x_1, y_1)}{(x_2, y_2)} \\
&= p(x_3, y_3|x_2, y_2)p(x_2, y_2|x_1, y_1)p(x_1, y_1) \\
&= p(x_1, y_1)p(x_2, y_2|x_1, y_1)p(x_3, y_3|x_2, y_2)
\end{aligned}$$

This is the equivalent form of p_1 given in (16) hence $p_2 = p_1$.

Part 3

Draw the belief networks for p_1 and p_2 and show that the number of edges in p_2 is larger than in p_1 (even though they represent the same distribution). This can be considered a form of causal heuristic. Representation p_1 is the causal one in which root causes are independent. In representation p_2, the independence of the causes is graphically lost, and this is therefore not the appropriate causal representation of the distribution. A heuristic for finding the 'right' causal graph is to select the graph (amongst all those that are equally likely) that has the least number of edges.

See Figure 3 for the belief network for p_1. To draw p_2, we must rearrange into the following form:

$$p(x_1, ..., x_n) = p(x_n)\prod_{i=1}^{n-1} p(x_i|x_{i+1}, ..., x_n)$$

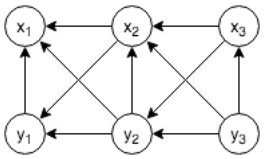

Figure 4: Belief network for $p(x_3, y_3)p(x_2, y_2|x_3, y_3)p(x_1, y_1|x_2, y_2)$.

For p_2, one way of displaying this is:

$$\begin{aligned}
p_2(x_1, y_1, x_2, y_2, x_3, y_3) &= p(x_3, y_3)p(x_2, y_2|x_3, y_3)p(x_1, y_1|x_2, y_2) \\
&= p(x_3|y_3)p(y_3)p(x_2|y_2, x_3, y_3)p(y_2|x_3, y_3)p(x_1|y_1, x_2, y_2)p(y_1|x_2, y_2)
\end{aligned}$$

This is displayed in Figure 4. Note that the direction of the vertical arrows is arbitrary given the way the expression is rearranged. The number of edges in p_2 is 11, more than double the 5 edges in p_1.

Problem 3.25

Draw a belief network for $p(x_{1:T})$

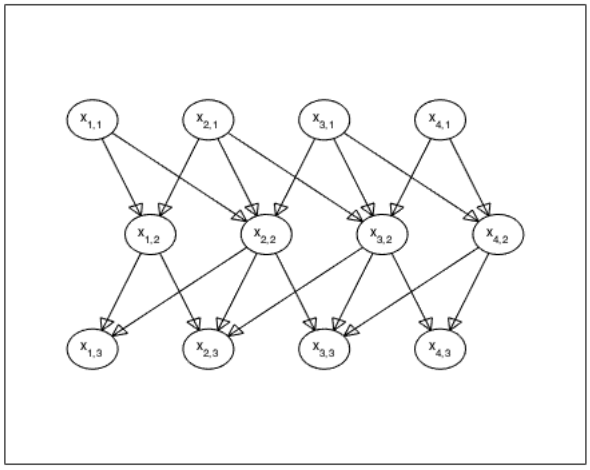

Figure 5: $x_{i,t}$ is the state of spacial variable i at time t for $T = 3$

Show that $p(\mathbf{x}_{1:T})$ **can also be written as the 'time reversed' form**

$$p(\mathbf{x}_T) \prod_{t=1:T-1} p(\mathbf{x}_t|\mathbf{x}_{t+1}) \tag{17}$$

and draw a belief network on the collection of variables $\{x_{1,t}, x_{2,t}, x_{3,t}, x_{4,t}\}, t = 1, ..., T$ **for this alternative representation of** $p(x_{1:T})$

From the causal representation of the graph, we can see the vector $\mathbf{x}_t$ is conditioned directly on $\mathbf{x}_{t-1}$, and we can then express this new notation such that

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_1) \prod_{t=2}^{T} p(\mathbf{x}_t|\mathbf{x}_{t-1}) \tag{18}$$

From Bayes' Rule recall:

$$p(\mathbf{x}_t|\mathbf{x}_{t-1}) = \frac{p(\mathbf{x}_{t-1}|\mathbf{x}_t)p(\mathbf{x}_t)}{p(\mathbf{x}_{t-1})} \tag{19}$$

Considering the beginning and end of 18:

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_1)\left(\frac{p(\mathbf{x}_1|\mathbf{x}_2)p(\mathbf{x}_2)}{p(\mathbf{x}_1)}\right)\left(\frac{p(\mathbf{x}_2|\mathbf{x}_3)p(\mathbf{x}_3)}{p(\mathbf{x}_2)}\right)\dots \\ \left(\frac{p(\mathbf{x}_{T-2}|\mathbf{x}_{T-1})p(\mathbf{x}_{T-1})}{p(\mathbf{x}_{T-2})}\right)\left(\frac{p(\mathbf{x}_{T-1}|\mathbf{x}_T)p(\mathbf{x}_T)}{p(\mathbf{x}_{T-1})}\right) \quad (20)$$

It denominator terms cancel with their term on the left, apart from the final $p(\mathbf{x}_T)$, leaving:

$$p(\mathbf{x}_{1:T}) = p(\mathbf{x}_T)\prod_{t=2:T} p(\mathbf{x}_{t-1}|\mathbf{x}_t) = p(\mathbf{x}_T)\prod_{t=1:T-1} p(\mathbf{x}_t|\mathbf{x}_{t+1}) \quad (21)$$

In order to display this graph, we need to consider the independence rules implied by the original distribution, and ensure that they are enforced in the new graph - with the constraint that inter-level directed edges now all point from $\mathbf{x}_{t+1}$ to $\mathbf{x}_t$.

In order to draw a belief network, one must obtain an expression for the joint distribution in the form of $\prod^i p(x_i|\chi_i)$ where χ_i is a set of variables not including x_i or the null set. In the time-reversed interpretation, we therefore want to rearrange the joint distribution such that variables are conditioned only on x_{t+1} instead of x_{t-1}. Let us substitute variables for $t-1=u$ for notation. In between two layers we therefore have:

$$p(layer) = p(\mathbf{x}_t|\mathbf{x}_u) = p(x_{1,t}|x_{1,u},x_{2,u})p(x_{2,t}|x_{1,u},x_{2,u},x_{3,u}) \\ p(x_{3,t}|x_{2,u},x_{3,u},x_{4,u})p(x_{4,t}|x_{3,u},x_{4,u}) \quad (22)$$

Consider reversing the first term with Bayes' Rule as the first step of deriving $p(\mathbf{x}_u|\mathbf{x}_t)$

$$p(x_{1,t}|x_{1,u},x_{2,u}) = \frac{p(x_{1,u}|x_{1,t},x_{2,u})p(x_{2,u}|x_{1,t})}{p(x_{1,u}|x_{1,t})} \quad (23)$$

It is quite clear that continuing will not result in a factorisation such that $p(\mathbf{x}_u|\mathbf{x}_t)$ is the product of functions of the components of $\mathbf{x}_u$. Therefore we cannot exploit the factorisation to remove any edges from the fully connected graph. Therefore every node in $\mathbf{x}_u$ is connected to every other node in both $\mathbf{x}_u$ and $\mathbf{x}_t$, which creates a rather messy belief network as shown in fig 6, shown only for $T=2$ for relative clarity.

Show that, in general, the number of edges in the 'causal' representation is $10\times(T-1)$, whereas in this time reversed form, the spatial sparsity has been lost and the number of edges is $22\times T-16$.

Clearly in the original causal network, there are T layers and correspondingly $T-1$ inter-connections between layers. Each layer $\mathbf{x}_t$ is connected to the layer $\mathbf{x}_{t+1}$ according to the distribution in the question, with outer nodes causally

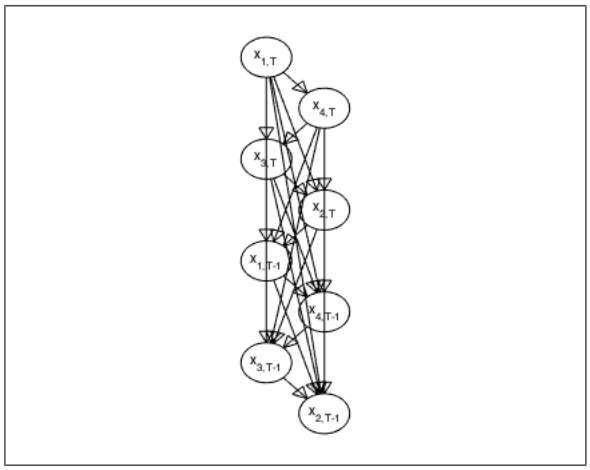

Figure 6: The time-reversed BN for layers T and $T-1$ only for clarity

connected to their two spatial neighbours and inner nodes connected to their three spacial neighbours. This gives $2\times2+2\times3=10$ nodes per inter-connection, and therefore $10\times(T-1)$ edges for the entire graph.

In the time-reversed version we have lost the factorisation of the graph which allowed us to express the distribution in a more sparse fashion with local spatial effects. We can no longer exploit this factorisation leading to a fully connected by layer graph, which necessarily has $4^2=16$ layer to layer connections, as well as 6 intra-layer. As there are again $T-1$ layer intersections and T layers, there are $16\times(T-1)+6\times T=22\times T-16$ edges in this representation.

Problem 3.26

The belief network for question 3.26

A) If we have no additional information about a student's skills, which modules (in addition to the project) should she take in order to maximise her chance of passing the Masters? State the probability of passing the Masters with this set of modules.

The best probability of passing the course given no conditioning information regarding the skills of the student is 0.636.

The module choices that achieve this probability are: Information Retrieval, Advanced Topics, Computational Modelling, Machine Vision, Graphical Models, Applied Machine Learning, Natural Language Processing and Affective Computing. In addition the student must pass the project which is included in

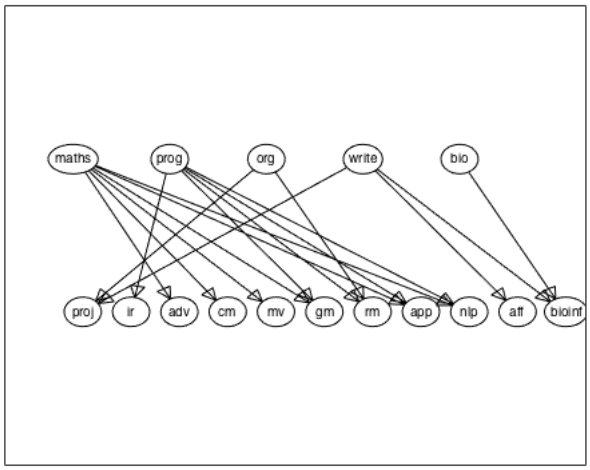

Figure 7: 3.26 Bayesian Network: We choose 8 of 10 modules plus the project

the probability calculation.

B)If we know the student is skilled in Maths and Organisation, which would be the best set of modules to take and the associated probability of passing the Masters?

The best probability of passing the course given we know the student is skilled in Maths and Organisation is 0.7698.

The module choices that achieve this are: Advanced Topics, Computational Modelling, Machine Vision, Graphical Models, Research Methods, Applied Machine Learning, Natural Language Processing and Affective Computing. Again the student must pass the project which is included in the probability calculation.

C)If we know the student is skilled in Biology and Writing, which would be the best set of modules to take and the associated probability of passing the Masters?

The best probability of passing the course given we know the student is skilled in Biology and Writing is 0.7283.

The module choices that achieve this are: Advanced Topics, Computational Modelling, Machine Vision, Graphical Models, Applied Machine Learning, Natural Language Processing, Affective Computing and Bioinformatics. Again the student must pass the project which is included in the probability calculation.

This was coded using the brml toolbox.

Approach and Code explanation.

For this exercise I chose to use the brml toolbox to create and populate the bayesian network. After creating a joint potential. I then looped through all

combinations of modules, ie choosing 8 from 10 plus the project. Each time I would condition the joint potential first on choosing these modules and then I set the potential given all passes. I stored both the probabilities and the module choices.

In the cases where the student also had some skills I conditioned upon these creating separate potentials. But the overall process was the same.

Intuition check:

Given skills in Biology I was pleased to see Bio-informatics added to the module choices. Similarly Research methods was added given mathematics skills. Given the Bayesian network this felt intuitive. Also I expected the overall probability of passing to increase given skills, which also appears to tie with intuition.

I chose this approach after becoming uncomfortable with two prior approaches, the first where I looped through each click event and had a far smaller network and simply updated iteratively, however i believe this introduces an order dependency to the advert clicks, which does not address the question.

The other approach I looked at was to marginalise over all variables bar the module in question. Although this does give the probability of passing each module individually, my discomfort was that I do not believe these probabilities are independent anymore due to new links being created when marginalising over skill potentials with joint children.

```
function ex326

import brml.*

[maths prog org write bio proj ir adv cm mv gm rm app nlp aff
    bioinf]=assign(1:16);
skill=1; unskilled=2;% define states, starting from 1.
pass=1;fail=2;

%5 skill variables and 10 modules - note the question is imprecise here
    so
%I have gone for the more complicated version - with 8 modules chosen out
%of 10

%num_modules = 9; % number of chosen modules including the project
varnames={'maths' 'prog' 'org' 'write' 'bio' 'proj' 'ir' 'adv' 'cm' 'mv'
    'gm' 'rm' 'app' 'nlp' 'aff' 'bioinf'};
for v=1:5;
    variable(v).name=varnames{v};
        variable(v).domain={'skill','unskilled'};
end

for v=6:16;
    variable(v).name=varnames{v}; variable(v).domain={'pass','fail'};
end
```

```
% For a BN associate p(i|pa(i)) with potential i: core skill nodes
pot(maths).variables=maths; pot(maths).table(skill)=0.95;
    pot(maths).table(unskilled)=1-pot(maths).table(skill);
pot(prog).variables=prog; pot(prog).table(skill)=0.8;
    pot(prog).table(unskilled)=1-pot(prog).table(skill);
pot(org).variables=org; pot(org).table(skill)=0.7;
    pot(org).table(unskilled)=1-pot(org).table(skill);
pot(write).variables=write; pot(write).table(skill)=0.95;
    pot(write).table(unskilled)=1-pot(write).table(skill);
pot(bio).variables=bio; pot(bio).table(skill)=0.3;
    pot(bio).table(unskilled)=1-pot(bio).table(skill);

%modules dependent on a single
pot(adv).variables=[adv maths];
tmptable1(pass, skill)=0.995; tmptable1(pass, unskilled)=0.1;
tmptable1(fail, skill)=0.005; tmptable1(fail, unskilled)=0.90;
pot(adv).table=tmptable1;

pot(ir).variables=[ir prog];
pot(ir).table=tmptable1;

pot(cm).variables=[cm maths];
pot(cm).table=tmptable1;

pot(mv).variables=[mv maths];
pot(mv).table=tmptable1;

pot(aff).variables=[aff write];
pot(aff).table=tmptable1;

% modules dependent on multiple skills
pot(gm).variables=[gm maths prog];
tmptable2(pass, skill, skill)=0.99; tmptable2(pass, skill,
    unskilled)=0.75;
tmptable2(pass, unskilled, skill)=0.75; tmptable2(pass, unskilled,
    unskilled)=0.01;
tmptable2(fail,:,:)=1-tmptable2(pass,:,:); % due to normalisation
pot(gm).table=tmptable2;

pot(rm).variables=[rm prog org];
pot(rm).table=tmptable2;

pot(app).variables=[app maths prog];
pot(app).table=tmptable2;

pot(nlp).variables=[nlp maths prog];
pot(nlp).table=tmptable2;

pot(proj).variables=[proj write org];
pot(proj).table=tmptable2;
```

```
pot(bioinf).variables=[bioinf bio write];
pot(bioinf).table=tmptable2;

pot=setpotclass(pot,'array'); % convert to cell array
drawNet(dag(pot),variable);

jointpot = multpots(pot(1:16));

% jointpot is the joint distribution of all variables
% we will be choosing 8 modules plus the project and to pass the msc need
% to pass each
all_passes = [pass,pass,pass,pass,pass,pass,pass,pass,pass];
course_res = [];
course_res_bw = [];
course_res_mo = [];

%loop through the module options
%we will chooose the project plus another 8 options from 10
% list of modules - [proj ir adv cm mv gm rm app nlp aff bioinf]

modules2selectfrom = [ir adv cm mv gm rm app nlp aff bioinf];
c = combnk(1:10,8);

for row = 1:size(c,1)
    module_selection = modules2selectfrom(c(row,:));
    %we always have to take the project
    module_selection = [proj module_selection];


    %note at the beginning i have a bayesian network with all the
    %potentials - in the first case we need our module choices and we
        know
    %nothing about our skills - so i marginalise over everything bar the
    %modules selected
    modules_pot = condpot(jointpot,module_selection);

    %choose the probability of all passes from the new potential
    selection_prob = {setpot(modules_pot,module_selection,all_passes
        ).table, module_selection};
    course_res(row,1) = selection_prob{1};

    %choose module choices given that the student has skills in maths
    %and organisation

    %first I set the overall network to ensure we condition on having
    %skills in maths and organisation
    mathorgpot = setpot(jointpot,[maths org],[skill skill]);
    %again on the new network marginise away everything bar modules
    %selected and the project
```

```
    modules_pot = condpot(mathorgpot,module_selection);

    %choose all passes
    selection_prob_mo = {setpot(modules_pot,module_selection,all_passes
        ).table, module_selection};
    course_res_mo(row,1) = selection_prob_mo{1};

    %choose module choices give that the student has skills in biology
    %and writing - same as above
    biowritingpot = setpot(jointpot,[bio write],[skill skill]);
    modules_pot = condpot(biowritingpot,module_selection);

    selection_prob_bw = {setpot(modules_pot,module_selection,all_passes
        ).table, module_selection};
    course_res_bw(row,1) = selection_prob_bw{1};
end

%find the maximum probabilities for no conditioning info, skilled in
    maths
%and org and skilled in bio and writing and show probs and module choices
%warning - be careful here - i am using the row of the max probability
    then
%referencing by the combo function i used earlier. Then I grab the hard
%coded course values ie 6 onwards from this reference. I would not do
    this
%professionally and this is for the purpose of this exercise only and is
%way too dependent on things earlier in the code

[partAprob partAI] = max(course_res);
partA_v = [6 c(partAI,:)+6];

partA_modules = [];
for ind = 1:size(partA_v,2);
    partA_modules = [partA_modules variable(partA_v(ind)).name];
end

[partBprob partBI] = max(course_res_mo);
partB_v = [6 c(partBI,:)+6];

partB_modules = [];
for ind = 1:size(partB_v,2)
    partB_modules = [partB_modules variable(partB_v(ind)).name];
end

[partCprob partCI] = max(course_res_bw);
partC_v = [6 c(partCI,:)+6];

partC_modules = [];
for ind = 1:size(partC_v,2);
    partC_modules = [partC_modules variable(partC_v(ind)).name];
```

```
end

partAprob
partA_modules

partBprob
partB_modules

partCprob
partC_modules
```

4

Problem 4.13

Show how for any singly-connected Markov network, one may construct a Markov equivalent belief network.

For any singly-connected graph, the only possible maximal clique is of size 2. This is trivially the case - if the size of the maximum clique was larger than 2, the graph would not be singly-connected by definition, as a clique of size 3 is not singly connected. If a clique of size 1 exists, the graph cannot be connected(assuming non-triviality of the graph's vertex set).

If we define:

$$\mathbf{V} = \{v|\text{v is a vertex in the Markov Network G}\} \tag{24}$$

$$\mathbf{E} = \{u_{ij}|\text{u is the edge connecting } i, j \forall i, j \in V\} \tag{25}$$

then given that the probability distribution over a Markov Network is the product of the potential functions over its maximal cliques and for a singly connected graph, the cliques are the set of all pairs of connected verticies. Therefore, we can write:

$$P(V) = \frac{1}{Z} \prod_{(i,j):u_{ij} \in E} \psi(i,j) \tag{26}$$

As a singly connected graph has no cliques larger than 2, any Belief Network which could have created it must have no colliders, as a minimal collider(two nodes colliding at a third) has a size 3 clique as it's Markov Network. Therefore, the Belief Network constructed from the pairwise representation of the singly connected Markov Network must contain no colliders.

We can then arbitrarily create a BN from the MN by adding directed edges - once an edge is added from a leaf node inwards on the graph, this induces a tree structure in the rest of the graph. Therefore, the number of leaf nodes

dictates how many possible Markov Equivelent BNs can be created from the described MN. Below we show that these are guaranteed to represent the same independence conditions as the Markov Network.

As established, any Markov Equivalent Belief Network generated from a singly connected Markov Network cannot contain any colliders. Therefore, the set of all conditionally independent nodes in a Belief Network is defined by:

$$C = \{(X \perp Y|Z) : \text{Z is a vertex in the path between X and Y}\} \tag{27}$$

However, this also defines the set of conditionally independent vertices of the Markov Network G by the Separation Procedure, as if Z is a vertex on the path between X and Y on a singly connected graph, the removal of vertex Z creates two unconnected components which must separate X and Y. By definition, the MN and the generated BN have the same number of verticies and from the above we can see that they represent the same set of independence conditions, so they are Markov Equivalent.

Problem 4.16

The objective function is found to have a maximum at 3338668, corresponding to the image plotted in Figure 8.

```
function ex_4_16

import brml.*

load('xnoisy','xnoisy');

[isize, jsize] = size(xnoisy);
n = isize * jsize;
indexes = reshape(1:n,isize,jsize);

W=spalloc(isize * jsize, isize * jsize, 339088);
for i = 1:isize
  fprintf('Row %d\n', i);
  for j = 1:jsize
    neighbours = [i + 1, j; i - 1, j; i, j + 1; i, j - 1];
    for row = 1:4
      ni = neighbours(row, 1);
      nj = neighbours(row, 2);
      if ni <= isize && ni > 0 && nj <= jsize && nj > 0
        W(indexes(ni, nj), indexes(i, j)) = 10;
      end
    end
  end
end
```

```
colormap bone;

opts.maxit=1;
opts.minit=1;

for i = 1:10
    fprintf('Loop %d ', i);
    opts.xinit=xnoisy(:);
    [xnoisy, score] = brml.binaryMRFmap(W, 2 * xnoisy, 1, opts);
    imagesc(reshape(xnoisy, isize, jsize));
    drawnow;
    fprintf('Score %d\n', score);
end
```

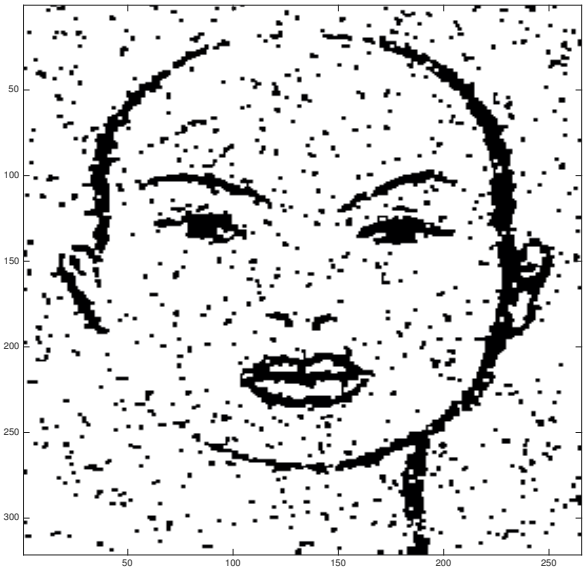

Figure 8: Final cleaned image.

5

Problem 5.10

The file BearBulldata contains the price of an asset through T = 200 timepoints. The price takes values from 1 to 100. If the market is in a 'bear' state, the price changes from time $t-1$ to t with probability transition matrix pbear$(t, t-1)$. If the market is in the 'bull' state, the price changes from time $t-1$ to t with transition matrix pbull$(t, t-1)$. If the market is in a 'bear' state it will remain so with probability 0.8. If the market is in a bull state it will remain so with probability 0.7. You may assume that at timestep 1, the market is uniformly in either a bear or bull state and also that the price distribution at timestep 1 is uniform. Use this model to compute the probability of the price at time $T+1$ given all the observed prices from time 1 to T; that is p(price$(T+1)$—price$(1:T)$). Using this probability, compute the expected gain price$(T+1)$ - price(T) in the price of the asset, and also the standard deviation in this price gain price$(T+1)$ - price(T).

The problem can be modelled as a system of two hidden random variables of

price P such that $dom(P) \in \{1:100\}$ and market state M such that $dom(M) \in \{bull, bear\}$. At each timestep t, $p(P_t|P_{t-1}, M_{t-1})$ and $p(M_t|M_{t-1})$.

This specifies a joint distribution:

$$p(P_{1:T}, M_{1:T}) = p(P_1)p(M_1)\prod_{t=2}^{T} p(P_t|P_{t-1}, M_{t-1})p(M_t|M_{t-1}) \tag{28}$$

$$= p(Joint_T) \tag{29}$$

We are given some observed prices $\hat{P}$ and the transition matrices which represent the functions by which M_t and P_t interact to form new price P_{t+1}. We also have the transition probabilities which govern how M_t progress to M_{t+1} We wish to consider the quantity $p(P_{T+1}|P_1 = \hat{P}_1, P_2 = \hat{P}_2, \ldots, P_T = \hat{P}_T)$ which we shall write as $p(P_{T+1}|P = \hat{P})$.

$$p(P_{T+1}|P = \hat{P}) = \frac{p(P_{T+1}, P = \hat{P})}{p(P = \hat{P})} \tag{30}$$

$$= \frac{\sum_{M_{1:T}} p(Joint_{T+1}|\hat{P})}{\sum_{M_{1:T}, P_{T+1}} p(Joint_{T+1}|\hat{P})} \tag{31}$$

In order to calculate this result naively, we would have to sum over $2^T = 2^{200}$ states, which is clearly computationally difficult. Instead we can use the bucket elimination algorithm which reduces the computational complexity to $\mathbf{O(n)}$. For this we utilise the `bucketelim.m` script from the BRML toolbox. The key difference from this example and the `demoBucketElim.m` methodology is the inclusion of evidential variables P_i. We use the toolbox to set the values of the potentials before running bucket elimination, and remove them from the order - which is optimal by summing over variables $M_{i:T}, P_{T+1}$.

This produces the expected value of 60.7035, and therefore an expected gain of 4.67%. This is as expected as we can see that the price is on an upwards trend towards $t = T$ and this would suggest a high probability that M_{200} is in the *bull* state.

The standard deviation is simply $\sqrt{p(P_{T+1} = x)(x - E(P_{T+1})^2}$, which returns a value of 8.237.

The code which generates these results is included below:

```
function r = bullbear
%variables 1-100 represent P_{1:100}, 101-200 represent M_{1:100} and
    201 is P_101
load BearBullproblem.mat
import brml.*

T = 200;
variables=1:(2*T+1);
bear = 1; bull = 2;
```

```
px = 1; pnx = 2;

pot{1}.variables=1;
pot{1}.table(1:100) = ones(1,100)*0.01;

pot{T+1}.variables=T+1;
pot{T+1}.table(bear) = 0.5;
pot{T+1}.table(bull) = 0.5;

for i=2:T
    pot{i}.variables=[i, i-1, T+i-1];
    pot{i}.table(:,:, bear) = pbear;
    pot{i}.table(:,:, bull) = pbull;

    pot{T+i}.variables=[T+i-1, T+i];
    pot{T+i}.table(bear, bear) = 0.7;
    pot{T+i}.table(bull, bull) = 0.8;
    pot{T+i}.table(bear, bull) = 0.3;
    pot{T+i}.table(bull, bear) = 0.2;
end

res = 2*T+1;
pot{res}.variables=[res, T, 2*T];
pot{res}.table(:,:, bear) = pbear;
pot{res}.table(:,:, bull) = pbull;

pot=setpotclass(pot,'array');
pot = setpot(pot,[1:T],p(1:T));

% figure; drawNet(dag(pot));

bucket = bucketelim(pot, (T+1):(res));

%return the final potential

p_fin = bucket{201}.table;

%normalise to find the price distribution

p_fin_dist = p_fin / sum(p_fin);

expected_price = sum(p_fin_dist .* (1:100)')
expected_gain = (expected_price - p(200)) / p(200)
expected_sd = sqrt(sum(p_fin_dist.*((1:100)' - expected_price).^2))
```

Problem 5.11

1. For each of the three spatial dimensions i=1,2,3, write down a single equation that relates a_i(t), t = 1,2,...,100 to the rescue station at $\mathbf{x}_{102}$.

The question uses x_{102} for the position of the spaceship [4.71,-6.97,8.59]' and for the astronaut. I will thus use p for the position of the astronaut without loss of generality.

For each dimension we know that:

$$v_i(t+1) = v_i(t) + \delta a_i(t), p_i(t+1) = p_i(t) + \delta v_i(t) \tag{32}$$

also for each dimension $p_i(1) = v_i(1) = 0$, and we have 102 time steps.

it is clear that $v_i(102) = \delta(\mathrm{a}_i(1) + a_i(2) + ... + a_i(101))$

and thus $p_i(102) = \delta^2(100 * \mathrm{a}_i(1) + 99 * a_i(2) + ... + a_i(100))$, therefore in each dimension the equations relating the position of the astronaut from the spaceship are as follows:-

$$4.71 - \delta^2 \sum_{t=1}^{100} (101 - t) a_1(t) \tag{33}$$

$$-6.97 - \delta^2 \sum_{t=1}^{100} (101 - t) a_2(t) \tag{34}$$

$$8.59 - \delta^2 \sum_{t=1}^{100} (101 - t) a_3(t) \tag{35}$$

2. What is the minimum amount of fuel that can be used to rendezvous with the rescue station at time 102?

The minimum amount of fuel is 22 units. 5 used in dimension 1, 8 in dimension 2, and 9 in dimension 3. The times at which the astronaut accelerates to achieve this minimum fuel value need not be unique, unless we seek to also reach the spaceship in the minimum time. Which is the way I have chosen my implementation.

3. Assuming that there is a sequence a1, . . . , a100 that will obtain a rendezvous with the rescue station at time 102, explain if you believe there is an efficient way to calculate the minimum amount of fuel. If there is an efficient algorithm, state it. Otherwise explain why no efficient algorithm is available.

There is an efficient way of achieving this. If we look at the position of the astronaut at time 102.

$$\mathrm{p}_i(102) = \delta^2(100 * a_i(1) + 99 * a_i(2) + ... + a_i(100)) \tag{36}$$

We can see that although each burst of acceleration costs 1 fuel unit. The effects are not equal through time. Indeed using fuel earlier is always more fuel efficient given a specific position. Thus an optimal algorithm would to be use fuel as early as possible and then when one is at point where one can reach the spaceship at time=102, with one more burst, then apply that burst at the correct timestep.

For example in dimension 1, multiplying through by δ^2, we have that

$$100a(0) + 99a(1) + ... + a(100) = 471 \tag{37}$$

if we apply a positive acceleration at times t=1,2,3,4 we reach the following equation

$$96a(5) + 95a(6) + ... + 73a(28) + ... + a(100) = 77 \tag{38}$$

Therefore we now drift until timestep t = 24, to apply our final burst arriving at point 4.71 at time t = 102, using 5 fuel units.

I have implemented this in matlab below. Together with a script to test the result.

```
function spaceman2

spaceship = [4.71,-6.97,8.59];
plusminus = repmat(sign(spaceship),101,1)';

delta = 0.1;
lost_in_space = int16(abs(spaceship(:))/delta^2);
not_there_yet = sum(lost_in_space);

acc = zeros(3,101); %note the last is not needed
coeff(1:3) = 100;
i(1:3) = 1;
while not_there_yet>0
    for j=1:3
        if lost_in_space(j) > coeff(j)
            lost_in_space(j) = lost_in_space(j) - coeff(j);
            coeff(j) = coeff(j)-1;
            acc(j,i(j)) = 1;
            i(j)=i(j)+1;
        elseif lost_in_space(j) <= coeff(j) && lost_in_space(j)>0
            final_index = coeff(j)-lost_in_space(j)+i(j);
            acc(j,final_index) = 1;
            lost_in_space(j) = 0;
        end
    end
    not_there_yet = sum(lost_in_space);
end

fuel_used = sum(sum(acc))
```

```
acc = acc.*plusminus

%test the solution
%because matlab indexes from 1 rather than starting at time zero i start
    at
%timestep 1timestep is one step further, note that the solution for
    minimum
%fuel in terms of when the spaceman accelerates is not unique. I have
%implemented an optimum solution assuming he wishes to also reach the
    spaceship
%as quickly as possible

pos = [];
vel = [];

fuel = 0;
pos(1:3,1) = 0;
vel(1:3,1) = 0;
a = zeros(3,101);
a(1,1) = 1;
a(3,1) =1;
a(2,1) =-1;
fuel = 3;

for t=2:102
    %In direction 1 test solution
    %that is apply +1 acceleration at timesteps
    %1,2,3,4 and then finally at timestep 24
    if t <5 || t==24
        a(1,t) = 1;
        fuel = fuel+1;
    end

    %In direction 2 test solution
    %that is apply -1 acceleration at timesteps
    %1,2,3,4,5,6,7 and then finally at timestep 83
    if t <8 || t==83
        a(2,t) = -1;
        fuel = fuel+1;
    end

    %In direction 3 test solution
    %that is apply +1 acceleration at timesteps
    %1,2,3,4,5,6,7,8 and then finally at timestep 14
    if t <9 || t==14
        a(3,t) = 1;
        fuel = fuel+1;
    end

    %use the explicit equations to test
```

```
    vel(:,t) = vel(:,t-1) + 0.1*a(:,t-1);
    pos(:,t) = pos(:,t-1) + 0.1*vel(:,t-1);
end

%check position is correct : fuel units used 22
final_pos = pos(:,102)
fuel

disp('distance to spaceship:')
spaceship'-pos(:,102)
```

Problem 5.12

An algorithm exists (for example Dijkstra's algorithm) that finds the minimum weighted path between two specified nodes on a graph; the algorithm however only generally works if all weights are non-negative. You have a minimum weighted path problem but with weights u_{ij} that can be negative. A friend suggests that you can still use Dijkstra's algorithm by making a new set of edge weights that are all non-negative, $w_{ij} = u_{ij}u^*$, where u^* is the minimum value of all the u_{ij} weights. Explain why this will not generally give the correct minimum weight path.

Dijkstra's algorithm applied to a graph with negative weights will not yield the shortest path between two vertices because it is a greedy algorithm, meaning that once it has evaluated the shortest path to a node, it never re-evaluates the node again. For this reason, negative weights cause the following issue:

Given a path

$$|P_{m,n,k}| = \sum_{u_{ij} \in P} u_{ij} \tag{39}$$

where $|P_{m,n,k}|$ is the set of edges and nodes forming a path from node m to node n of length k and u_{ij} represents the weight of the edge between nodes i and j.

If we consider this path to be the shortest such path between nodes m and n given by Dijkstras's algorithm, it is possible that there exists a path $P_{m,n,k+1}$ via a node l, which would not have been considered by Dijkstra on the evaluation of node $n-1$ in the path $P_{m,n,k}$ as the weight from node $n-1$ to l may have been larger than the weight from $n-1$ to n and hence not considered. The key step here is that the path from l to n may be negative, meaning that the path $n-1$ to n via l may in fact be shorter, but is not considered by Dijkstra's algorithm. For this reason, Dijkstra does not work for graphs with negative weights.

On consideration of the corrected negative weight graph created by subtracting the smallest weight from each edge, it is clear that for a path $P_{m,n,k}$ in the

original graph G, the weight of the same path in the updated graph G^* is given by:

$$|P^*_{m,n,k}| = |P_{m,n,k}| - k(u^*) \tag{40}$$

where u^* is the minimum weight edge in the original graph G. However, this edge weight transformation does not preserve shortest paths. In order to see this, it is instructive to consider the following example:

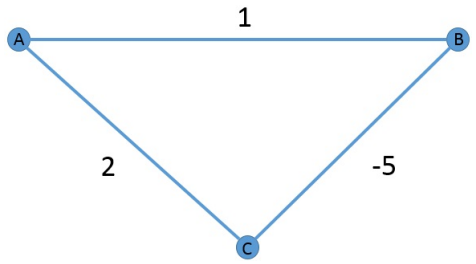

Figure 9: Negative Weight Graph G. Using Dijkstra's algorithm to compute the shortest weight from A to B would result in the path AB, even though ACB is the shortest path between A and B for this graph.

However, we see that when we apply the edge weight transformations, the result is Figure 3. In this graph, the minimum weight path from A to B is AB, not ACB. This demonstration shows that shortest paths do not endure under weight edge transformations on negative weight directed graphs, meaning that this alteration is not guaranteed to return the correct minimum weight path for the original graph G.

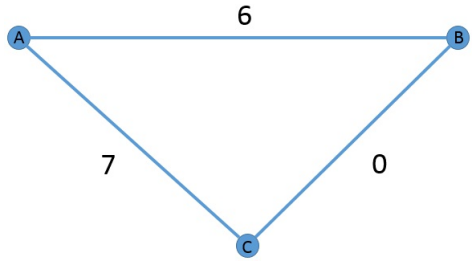

Figure 10: Adjusted Weight Graph G^*. The shortest path between A and B is not preserved under the edge weight transformation.

```
The code below uses message passing on a Markov Network to calculate the
    minimum weight path from planet 1 to planet 1725.
```

```
function r = Ex5_13_Sub

import brml.*
load SimoHurrta.mat
graphStruct = A;
tax = t;
location = x;

nonEdges = tril(inf(2000)); % path weight is infinite if no link i->j

graphStruct = graphStruct + nonEdges;

% calculate one weight for the edge to each node
for(j = 2:2000)
    for(i = 1:(j-1))
        graphStruct(i,j) = sqrt((location(i) - location(j))*(location(i)
            - location(j))') - tax(j);
    end
end
save ('TransitionMatrix','graphStruct')

start = 1;
finish = 1725;

pathweight=mostprobpath(-graphStruct',start,finish); % use negative
    transition matrix since we want shortest path
% transpose is required since mostprobablepath assumes a transition
    p(planet1|planet1725)

fprintf('shortest path has weight %g\n',-pathweight);

%=========================================================================
function logprob=mostprobpath(logtransition,startstate,endstate)
%This function finds the weight of the shortest path in a Markov Chain

import brml.*
N=size(logtransition,1);
E(1)=logtransition(endstate,startstate); % starting log path probability
lambda{1}=logtransition(:,startstate); % initial message

for t=2:N
   for s=1:N
      lambda{t}(s,1)= max(lambda{t-1}+logtransition(s,:)'); % message
          recursion
   end
   E(t)=lambda{t}(endstate); % optimal log path probability after t
       timesteps
end
[logprob]=max(E); % optimal log path probability and number of timesteps
```

For clarity of exposition, the function mostprobablepath.mat from the BMRL Toolbox is re-displayed here, with a reduction in functionality for simplicity. The shortest path between Planet 1 and Planet 1725 is -490.337, meaning Simo is collecting more in tax revenue from the planets than his outlay on fuel.