Belief Propagation in Bayesian Networks

Anmol Dwivedi

1 Introduction

The goal of Bayesian Network inference is to evaluate the probability distribution over some set of variables or evaluate their most likley states, given the values of other set of variables. In general, there are four types of inference:

- Posterior probability inference
- Maximum a Posteriori (MAP) inference
- Most Probable Explanation (MPE) inference
- Model Likelihood (ML) inference

Assume a bayesian network $\mathcal{G}$, parameterized by Θ, over random variables $X = \{X_U, X_E\}$ where the X_U denotes the set of unknown nodes and X_E denotes the set of evidence nodes that are realized.

Then the problem of posterior probability inference, or the sum-product inference, deals with evaluating $\mathbb{P}(X_q|X_E)$ where X_q denotes the query variables such that $X_q \subseteq X_U$.

The problem of Maximum a Posteriori (MAP) inference, or the max-sum inference, deals with finding out ${X_q}^* = \arg\max_{X_q} \mathbb{P}(X_q|X_E)$, i.e., the most likely configuration for the query variables given evidence.

The problem of Most Probable Explanation (MPE) inference, or the max-product inference, deals with finding out ${X_U}^* = \arg\max_{X_U} \mathbb{P}(X_U|X_E)$, i.e., the most likely configuration for all the unknown random variables given evidence.

The problem of Model Likelihood (ML) inference is to evaluate the liklihood of the model in terms of its structure and parameters for a given set of observations. i,e., $\mathbb{P}(X_E|\Theta, \mathcal{G})$. Usually done to select the model, or bayesian network, that best explains the observations X_E.

For example, for the bayesian network given in Figure 1, and asusuming that the random variables F, X are realized, the query variables $X_U = \{S, A, B, L, T, E\}$, $X_E = \{X, F\}$ and $X_q \subseteq X_U$.

2 Problem Statement

The goal of this project is to implement belief propagation algorithms for posterior probability and most probable explanation (MPE) inference for the Bayesian Network given in Figure 1 where each node is binary, assuming a value of 1 or 0 and CPT for each node is given. Specifically, there are two problems that this project addresses:

- First is to implement marginal posterior probability (sum-product) inference algorithm using MATLAB to compute $\mathbb{P}(X_q|F = 1, X = 0)$ where X_q is the query variable that belongs to the set $X_q \in \{S, A, B, L, T, E\}$.
- Second is to implement the MPE (max-product) inference algorithm to compute s^*, a^*, b^* l^* t^* e^* = $\arg\max_{s,a,b,l,t,e}$ $\mathbb{P}(s, a, b, l, t, e|F = 1, X = 0)$

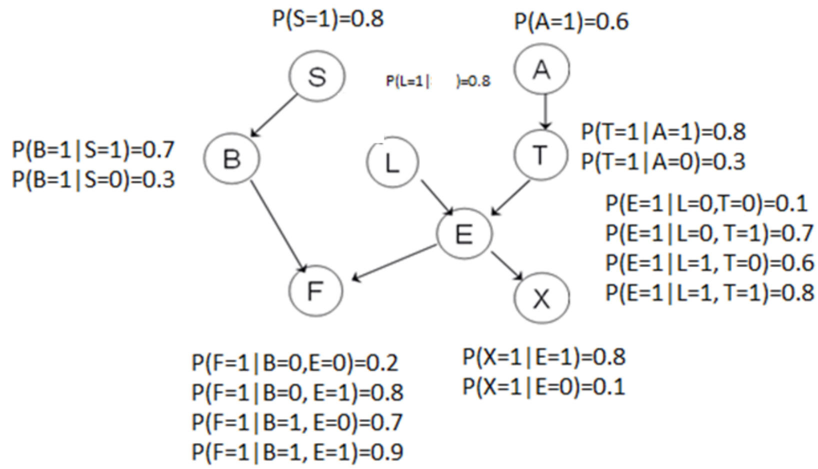

Figure 1

3 Theory

The general elimination algorithm for inference allows us to compute a single marginal distribution or in other words, perform inference for a given random variable at a time. That means that the algorithm must be run separately for each node to perform inference for multiple random variables leading to additional complexity. Belief propagation is an algorithm for exact inference on directed graphs without loops. The belief propagation algorithm allows us to compute all the marginal and conditionals by computing intermediate factors commonly known as messages that are common to many elimination orderings. It efficiently computes all the marginals especially in the special case singly connected graph. A DAG is singly connected if its underlying undirected graph is a tree, i.e., there is only one undirected path between any two nodes. A generalized version of belief propagation for arbitrary graphs is also known as the junction tree algorithm.

The idea behind belief propagation for tree structured graphs can be summarized as follows: Since we are dealing with singly connected graphs, every node X in the DAG $\mathcal{G}$ divides the evidence E into upstream and downstream evidence. We denote E^+ part of the evidence accessible through the parents of node X, also referred to as the prior message that we denote by π and analogously we denote E^- part of the diagnostic evidence through the children of node X, also referred to as the likelihood message that we denote by λ. The fresh idea in this method is that the evidence bisected into two E^+ and E^- when viewed from the perspective of any arbitrary node in the the DAG $\mathcal{G}$. Now in order to evaluate the probability of variable X, we have the following:

$$\mathbb{P}(X|E) = \frac{\mathbb{P}(X,E)}{\mathbb{P}(E)} = \frac{\mathbb{P}(X,E^+,E^-)}{\mathbb{P}(E^+,E^-)} = \alpha \cdot \mathbb{P}(X|E^+) \cdot \mathbb{P}(E^-|X,E^+) \tag{1}$$

$$= \alpha \cdot \mathbb{P}(X|E^+) \cdot \mathbb{P}(E^-|X) \tag{2}$$

$$= \alpha \cdot \pi(X) \cdot \lambda(X) \tag{3}$$

$$= \text{Belief}(X) \tag{4}$$

3.1 Sum-Product Inference

Where the $\pi(X)$ denotes the prior message that is the prior that is received from the parents of RV X, $\lambda(X)$ denotes the liklihood message that is the liklihood received form the children of RV X and α is the normalization constant. The advantage of this approach is that each of $\pi(X)$ and $\lambda(X)$ can be computed via the local message passing algorithm between nodes in the DAG as follows:

$$\pi(X) = \mathbb{P}(X|E^+) = \sum_{Parents(X)} \mathbb{P}(X|Parents(X)) \cdot \mathbb{P}(Parents(X)|E^+) \tag{5}$$

$$= \sum_{Parents(X)} \mathbb{P}(X|Parents(X)) \cdot \pi_{Parents(X)}(X) \tag{6}$$

$$= \sum_{P_1,P_2,\ldots,P_N} \mathbb{P}(X|P_1,P_2,\ldots,P_N) \cdot \prod_{P_i} \pi_{P_i}(X) \tag{7}$$

and similarly,

$$\lambda(X) = \mathbb{P}(E^-|X) = \lambda_{Children(X)}(X) \tag{8}$$

$$= \prod_{C_i} \lambda_{C_i}(X) \tag{9}$$

Where the messages received by X from its parents are denoted by $\pi_{P_i}(X)$ and the messages received by X from its children is denoted by $\lambda_{C_i}(X)$.

$$\pi_{P_i}(X) = \pi(P_i) \cdot \prod_{C \in Children(P_i) \setminus X} \lambda_C(P_i) \tag{10}$$

$$\lambda_{C_i}(X) = \sum_{C_i} \lambda(C_i) \sum_{u_k \in Parents(C_i) \setminus X} \mathbb{P}(C_i|X,u_1,\ldots,u_N) \prod_k \pi_{u_k}(C_i) \tag{11}$$

Before we discuss the algorithm, we discuss the initializations that must be done before the start of the belief propagation algorithm.

Algorithm 1 Sum-Product Algorithm [1]

1: Initialization step for evidence nodes: $\forall X \in X_E$, if $\quad x_i = e_i \quad \lambda(x_i) = 1, \pi(x_i) = 1$, otherwise $\lambda(x_i) = 0, \pi(x_i) = 0$
2: Initialization step for nodes without parents: $\lambda(x_i) = 1,\ \pi(x_i) = \mathbb{P}(x_i)$
3: Initialization step for nodes without children: $\lambda(x_i) = 1$
4: **while** Until no change in messages **do**
5:     Compute the message π message from each of the parents $\pi_{P_i}(X)$ (10)
6:     Calculate the total parent message $\pi(X)$ (5)
7:     Compute the λ message from each $\lambda_{C_i}(X)$ (11)
8:     Calculate the total message $\lambda(X)$ (8)
9:     Update belief Belief(X) and normalize.
10: **end while**

3.2 Max-Product Inference

Replacing the sum by the max operator, the equation for the max-product messages are given by:

$$\pi(X) = \mathbb{P}(X|E^+) = \max_{P_1,P_2,\ldots,P_N} \mathbb{P}(X|P_1,P_2,\ldots,P_N) \cdot \prod_{P_i} \pi_{P_i}(X) \tag{12}$$

$$\lambda(X) = \mathbb{P}(E^{-}|X) = \prod_{C_i} \lambda_{C_i}(X) \tag{13}$$

$$\pi_{P_i}(X) = \pi(P_i) \cdot \prod_{C \in Children(P_i) \setminus X} \lambda_C(P_i) \tag{14}$$

$$\lambda_{C_i}(X) = \max_{C_i} \lambda(C_i) \max_{u_k \in Parents(C_i) \setminus X} \mathbb{P}(C_i|X, u_1, \ldots, u_N) \prod_k \pi_{u_k}(C_i) \tag{15}$$

Algorithm 2 Max-Product Algorithm [1]

Initialization step for evidence nodes: $\forall X \in X_E$, if $\quad x_i = e_i \quad \lambda(x_i) = 1, \pi(x_i) = 1 \quad$, otherwise $\lambda(x_i) = 0, \pi(x_i) = 0$
Initialization step for nodes without parents: $\lambda(x_i) = 1,\ \pi(x_i) = \mathbb{P}(x_i)$
Initialization step for nodes without children: $\lambda(x_i) = 1$
while Until no change in messages **do**
    Compute the message π message from each of the parents $\pi_{P_i}(X)$ (14)
    Calculate the total parent message $\pi(X)$ (12)
    Compute the λ message from each $\lambda_{C_i}(X)$ (15)
    Calculate the total message $\lambda(X)$ (13)
    Update belief Belief(X) and find the best state for each node independently by finding the state that maximizes its marginal for each node in $\mathcal{G}$.
end while

4 Experiments

The following are the simplified equations for the sum-product algorithm that I have implemented for the DAG in Figure 1. In the code attached, I compute the each of the messages in the following order to obtain beliefs for the DAG.

$$\pi(T) = \sum_A \mathbb{P}(T|A) \cdot \pi(A) \tag{16}$$

$$\pi(T) = \sum_{L,T} \mathbb{P}(E|L, T) \cdot \pi(L) \cdot \pi(T) \tag{17}$$

$$\pi(B) = \sum_S \mathbb{P}(B|S) \cdot \pi(S) \tag{18}$$

$$\lambda(B) = \sum_F \lambda(F) \sum_E P(F|B, E) \cdot \pi(E) \cdot \sum_E P(X|E)\lambda(X) \tag{19}$$

$$\lambda(E) = \sum_F \lambda(F) \sum_B P(F|B, E) \cdot \pi(E) \cdot \sum_X P(X|E)\lambda(X) \tag{20}$$

$$\lambda(T) = \sum_E \lambda(E) \sum_L P(E|L,T) \cdot \pi(L) \tag{21}$$

$$\lambda(S) = \sum_B \mathbb{P}(B|S) \cdot \lambda(B) \tag{22}$$

$$\lambda(L) = \sum_E \lambda(E) \sum_T \mathbb{P}(E|L,T) \cdot \pi(T) \tag{23}$$

$$\lambda(A) = \sum_T \lambda(T) \cdot \mathbb{P}(T|A) \tag{24}$$

Replacing the sum by max gives the equations for the Max-Product algorithms.
Simulation results are as follows:

Listing 1: Output for 'main.m'

```
"Marginal Posterior Probability given evidence P(A=1| F=1, X=0) is 0.56401"

"Marginal Posterior Probability given evidence P(A=0| F=1, X=0) is 0.43599"


"Marginal Posterior Probability given evidence P(S=1| F=1, X=0) is 0.83893"

"Marginal Posterior Probability given evidence P(S=0| F=1, X=0) is 0.16107"


"Marginal Posterior Probability given evidence P(L=1| F=1, X=0) is 0.75544"

"Marginal Posterior Probability given evidence P(L=0| F=1, X=0) is 0.24456"


"Marginal Posterior Probability given evidence P(T=1| F=1, X=0) is 0.52802"

"Marginal Posterior Probability given evidence P(T=0| F=1, X=0) is 0.47198"


"Marginal Posterior Probability given evidence P(E=1| F=1, X=0) is 0.43043"

"Marginal Posterior Probability given evidence P(E=0| F=1, X=0) is 0.56957"


"Marginal Posterior Probability given evidence P(B=1| F=1, X=0) is 0.76332"

"Marginal Posterior Probability given evidence P(B=0| F=1, X=0) is 0.23668"
```

Listing 2: Output for mainMaxProduct

```
"Random Variable A takes MPE assignment 0"

"Random Variable S takes MPE assignment 1"
```

```
"Random Variable L takes MPE assignment 1"

"Random Variable T takes MPE assignment 0"

"Random Variable E takes MPE assignment 0"

"Random Variable B takes MPE assignment 1"
```

5 Conclusion

This project performs MPE and posterior probability inference for the DAG in Figure 1 using belief propagation algorithm in MATLAB. Issues I encountered were figuring out how to store the the probability values in an appropriate data structure in order to efficiently compute the factor product and then marginalize out relevant variables to complete inference. Learnt about various algorithms in order to perform inference in DAGs.

6 Appendix

The idea behind my code is that I am considering the conditional probabailty table values and the messages $\pi(.)$, $\lambda(.)$ as a factor. I am storing these factors in a $struct$ data structure in MATLAB such that each factor has fields such as its name (stored in field $VarName$), the number of values it takes, binary in this homework (stored in the field $Cardinality$) and the conditional probability values (stored in the field $ProbabilityValues$). In order to evaluate a message $\pi(.), \lambda(.)$, I break the operations into two steps. First I carry out the factor product and then I marginalize the relevant variable needed to obtain that particular message.

In order to reproduce results, run then "main.m" file under both folders Sum Product and Max Product. All other files are function that are called from the main.m file.

Listing 3: file 'main.m' under folder SumProduct

```
clc
clear all

% Dictionary for storing the Random Variable representation as their equivalent numerical ↩
    values;
keys = {'A', 'S', 'L', 'T', 'E', 'X', 'B', 'F'};
value = {1, 2, 3, 4, 5, 6, 7, 8};
Let_to_Num = containers.Map(keys, value);
Num_to_Let = containers.Map(value, keys);

% Input the probability tables;
probab.input(1) = struct('Var_Name', [Let_to_Num('A')], 'Cardinality', [2], 'Probability_Values↩
    ', [0.6 0.4]);
probab.input(2) = struct('Var_Name', [Let_to_Num('S')], 'Cardinality', [2], 'Probability_Values↩
    ', [0.8 0.2]);
probab.input(3) = struct('Var_Name', [Let_to_Num('L')], 'Cardinality', [2], 'Probability_Values↩
    ', [0.8 0.2]);
probab.input(4) = struct('Var_Name', [Let_to_Num('T'), Let_to_Num('A')], 'Cardinality', [2, 2],↩
     'Probability_Values', [0.8 0.2 0.3 0.7]);
probab.input(5) = struct('Var_Name', [Let_to_Num('E'), Let_to_Num('L'), Let_to_Num('T')], '↩
    Cardinality', [2, 2, 2], 'Probability_Values', [0.8 0.2 0.7 0.3 0.6 0.4 0.1 0.9]);
probab.input(6) = struct('Var_Name', [Let_to_Num('X'), Let_to_Num('E')], 'Cardinality', [2, 2],↩
     'Probability_Values', [0.8 0.2 0.1 0.9]);
```

```
probab.input(7) = struct('Var_Name', [Let_to_Num('B'), Let_to_Num('S')], 'Cardinality', [2, 2], 'Probability_Values', [0.7 0.3 0.3 0.7]);
probab.input(8) = struct('Var_Name', [Let_to_Num('F'), Let_to_Num('B'), Let_to_Num('E')], 'Cardinality', [2, 2, 2], 'Probability_Values', [0.9 0.1 0.8 0.2 0.7 0.3 0.2 0.8]);

% Initializations for PI and LAMBDA messages for Random Variables without parents;
PI.A = struct('Var_Name', [Let_to_Num('A')], 'Cardinality', [2], 'Probability_Values', [0.6 0.4]);
LAMBDA.A = struct('Var_Name', [Let_to_Num('A')], 'Cardinality', [2], 'Probability_Values', [1 1]);

PI.L = struct('Var_Name', [Let_to_Num('L')], 'Cardinality', [2], 'Probability_Values', [0.8 0.2]);
LAMBDA.L = struct('Var_Name', [Let_to_Num('L')], 'Cardinality', [2], 'Probability_Values', [1 1]);

PI.S = struct('Var_Name', [Let_to_Num('S')], 'Cardinality', [2], 'Probability_Values', [0.8 0.2]);
LAMBDA.S = struct('Var_Name', [Let_to_Num('S')], 'Cardinality', [2], 'Probability_Values', [1 1]);

% Initializations for PI and LAMBDA messages for Random Variables that are observed as Evindence;
PI.X = struct('Var_Name', [Let_to_Num('X')], 'Cardinality', [2], 'Probability_Values', [0 1]);
LAMBDA.X = struct('Var_Name', [Let_to_Num('X')], 'Cardinality', [2], 'Probability_Values', [0 1]);

PI.F = struct('Var_Name', [Let_to_Num('F')], 'Cardinality', [2], 'Probability_Values', [1 0]);
LAMBDA.F = struct('Var_Name', [Let_to_Num('F')], 'Cardinality', [2], 'Probability_Values', [1 0]);

% Forming PI(T)
temp = Product(probab.input(4), PI.A);
PI.T = Sum_Marginalization(temp,[Let_to_Num('A')]);

% Computing PI(E)
temp = Product(probab.input(5), Product(PI.T, PI.L));
PI.E = Sum_Marginalization(temp,[Let_to_Num('L'), Let_to_Num('T')]);

% Computing PI(B)
temp = Product(probab.input(7), PI.S);
PI.B = Sum_Marginalization(temp,[Let_to_Num('S')]);

% Computing LAMBDA(B)
temp = Product(probab.input(6), LAMBDA.X);
LAMBDA_X_TO_E = Sum_Marginalization(temp, [Let_to_Num('X')]);
PI_E_TO_F = Product(PI.E, LAMBDA_X_TO_E);

temp = Product(probab.input(8), PI_E_TO_F);
temp_1 = Sum_Marginalization(temp, [Let_to_Num('E')]);
temp_2 = Product(temp_1, LAMBDA.F);
LAMBDA.B = Sum_Marginalization(temp_2, [Let_to_Num('F')]);

% Computing LAMBDA(E)
temp = Product(probab.input(8), PI.B);
temp_1 = Sum_Marginalization(temp, [Let_to_Num('B')]);
temp_2 = Product(temp_1, LAMBDA.F);
LAMBDA_F_TO_E = Sum_Marginalization(temp_2, [Let_to_Num('F')]);
LAMBDA.E = Product(LAMBDA_F_TO_E, LAMBDA_X_TO_E);

% Computing LAMBDA(T)
temp = Product(probab.input(5), PI.L);
```

```
temp_1 = Sum_Marginalization(temp, [Let_to_Num('L')]);
temp_2 = Product(temp_1, LAMBDA.E);
LAMBDA_E_TO_T = Sum_Marginalization(temp_2, [Let_to_Num('E')]);
LAMBDA.T = LAMBDA_E_TO_T;

% Computing LAMBDA(S)
temp = Product(probab.input(7), LAMBDA.B);
LAMBDA_B_TO_S = Sum_Marginalization(temp, [Let_to_Num('B')]);
LAMBDA.S = LAMBDA_B_TO_S;

% Computing LAMBDA(L)
temp = Product(probab.input(5), PI.T);
temp_1 = Sum_Marginalization(temp, [Let_to_Num('T')]);
temp_2 = Product(temp_1, LAMBDA.E);
LAMBDA_T_TO_E = Sum_Marginalization(temp_2, [Let_to_Num('E')]);
LAMBDA.L = LAMBDA_T_TO_E;

% Computing LAMBDA(A)
temp = Product(probab.input(4), LAMBDA.T);
LAMBDA_T_TO_A = Sum_Marginalization(temp, [Let_to_Num('T')]);
LAMBDA.A = LAMBDA_T_TO_A;

% Calculating Beliefs, Normalizing them in order to obtain the marginal probability given evidence
temp = Product(LAMBDA.A, PI.A);
A_1 = normalization(temp.Probability_Values);
display(strcat('Marginal Posterior Probability given evidence'," P(", Num_to_Let(1), '=1| F=1, X=0) is ', " ", string(A_1(1))))
display(strcat('Marginal Posterior Probability given evidence'," P(", Num_to_Let(1), '=0| F=1, X=0) is ', " ", string(A_1(2))))
fprintf('\n\n')

temp = Product(LAMBDA.S, PI.S);
S_2 = normalization(temp.Probability_Values);
display(strcat('Marginal Posterior Probability given evidence'," P(", Num_to_Let(2), '=1| F=1, X=0) is ', " ", string(S_2(1))))
display(strcat('Marginal Posterior Probability given evidence'," P(", Num_to_Let(2), '=0| F=1, X=0) is ', " ", string(S_2(2))))
fprintf('\n\n')

temp = Product(LAMBDA.L, PI.L);
L_3 = normalization(temp.Probability_Values);
display(strcat('Marginal Posterior Probability given evidence'," P(", Num_to_Let(3), '=1| F=1, X=0) is ', " ", string(L_3(1))))
display(strcat('Marginal Posterior Probability given evidence'," P(", Num_to_Let(3), '=0| F=1, X=0) is ', " ", string(L_3(2))))
fprintf('\n\n')

temp = Product(LAMBDA.T, PI.T);
T_4 = normalization(temp.Probability_Values);
display(strcat('Marginal Posterior Probability given evidence'," P(", Num_to_Let(4), '=1| F=1, X=0) is ', " ", string(T_4(1))))
display(strcat('Marginal Posterior Probability given evidence'," P(", Num_to_Let(4), '=0| F=1, X=0) is ', " ", string(T_4(2))))
fprintf('\n\n')

temp = Product(LAMBDA.E, PI.E);
E_5 = normalization(temp.Probability_Values);
display(strcat('Marginal Posterior Probability given evidence'," P(", Num_to_Let(5), '=1| F=1, X=0) is ', " ", string(E_5(1))))
display(strcat('Marginal Posterior Probability given evidence'," P(", Num_to_Let(5), '=0| F=1, X=0) is ', " ", string(E_5(2))))
```

```
fprintf('\n\n')

temp = Product(LAMBDA.B, PI.B);
B_7 = normalization(temp.Probability_Values);
display(strcat('Marginal Posterior Probability given evidence'," P(", Num_to_Let(7), '=1| F=1, X=0) is ', " ", string(B_7(1))))
display(strcat('Marginal Posterior Probability given evidence'," P(", Num_to_Let(7), '=0| F=1, X=0) is ', " ", string(B_7(2))))
fprintf('\n\n')
```

By default, I store the probability table values in such a way that follows the following rule. Consider the conditional distribution over varibles $X_1|X_2, X_3$ each of which takes two values 0 and 1. In order to store these values I store it such the following form pertaining to the particular order of the input variables. You can cross check how to feed in values for the homework assignment at the start of the main.m file.

Listing 4: Illustration for storing CPD values in the field Probability Values of the structure for each variable

```
% -+-----+-----+-----+---------------------+
%  | X_1 | X_2 | X_3 | factor(X_1, X_2, X_3)|
% -+-----+-----+-----+---------------------+
%  |  1  |  1  |  1  |     factor.val(1)    |
% -+-----+-----+-----+---------------------+
%  |  0  |  1  |  1  |     factor.val(2)    |
% -+-----+-----+-----+---------------------+
%  |  1  |  0  |  1  |     factor.val(3)    |
% -+-----+-----+-----+---------------------+
%  |  0  |  0  |  1  |     factor.val(4)    |
% -+-----+-----+-----+---------------------+
%  |  1  |  1  |  0  |     factor.val(5)    |
% -+-----+-----+-----+---------------------+
%  |  0  |  1  |  0  |     factor.val(6)    |
% -+-----+-----+-----+---------------------+
%  |  1  |  0  |  0  |     factor.val(7)    |
% -+-----+-----+-----+---------------------+
%  |  0  |  0  |  0  |     factor.val(8)    |
% -+-----+-----+-----+---------------------+
```

Hence, in order to convert a given index $i \in [1, 8]$ into an assignment for the RVs X_1, X_2 and X_3, I write the following function that takes the index and the length of assignment as input and return the assignments for the random variables as an array.

Listing 5: MATLAB function for conversion to assignment

```
function assignment = conv_to_assignment(index, length_of_assignment)

assignment = [];
for i = 1:length(index)
temp = flip(de2bi(index(i)-1,log(length_of_assignment)/log(2),'left-msb')) + 1;
assignment = [assignment; temp];
end

end
```

On the other hand, in order to convert an assignment for a set of random variables X_1, X_2, X_3 etc. in to an index i, I write the following function that takes the assignemnt and the length as input and returns the index for that specific assignment.

Listing 6: MATLAB function for conversion to index

```
function index = conv_to_index(A, len)

% Given Input Vector Consisting of Sequence
size_temp = size(A);
for j = 1:size_temp(1)
temp = 0;
for i = 1:size_temp(2)
temp = temp + (A(j, i) - 1)*(2^(i-1));
end
index(j, 1) = temp + 1;
end
end
```

This function just carries out normalization for an input vector.

Listing 7: MATLAB function to carry out normalization

```
function B = normalization(A)

B = zeros(2, 1);
TEMP_1 = A(1);
TEMP_2 = A(2);
B(1) = TEMP_1/(TEMP_1 + TEMP_2);
B(2) = TEMP_2/(TEMP_1 + TEMP_2);
return
end
```

This function just carries out marginalization given a factor A and the marginalization variable V such that V is summed out from the variables present in factor A.

Listing 8: MATLAB function to carry out Sum Marginalization

```
function B = Sum_Marginalization(A, V)

[B.Var_Name, mapB] = setdiff(A.Var_Name, V);
B.Cardinality = A.Cardinality(mapB);
B.Probability_Values = zeros(1, prod(B.Cardinality));

assignments = conv_to_assignment(1:length(A.Probability_Values), prod(A.Cardinality));
indxB = conv_to_index(assignments(:, mapB), B.Cardinality);

for i=1:prod(B.Cardinality)
for j=1:prod(A.Cardinality)
if indxB(j)==i
B.Probability_Values(i)=B.Probability_Values(i)+A.Probability_Values(j);
end
end
end

end
```

This function just carries out the product between two input factors A and B.

Listing 9: MATLAB function to carry out the Product of two given probability tables

```

function C = Product(A, B)

C.Var_Name = union(A.Var_Name, B.Var_Name);

[dummy, mapA] = ismember(A.Var_Name, C.Var_Name);
[dummy, mapB] = ismember(B.Var_Name, C.Var_Name);

C.Cardinality = zeros(1, length(C.Var_Name));
C.Cardinality(mapA) = A.Cardinality;
C.Cardinality(mapB) = B.Cardinality;

assignments = conv_to_assignment(1:prod(C.Cardinality), prod(C.Cardinality));  % Creating the ↩
    assignment table for each index of the DESIRED .val representation

assignments(:, mapA);
assignments(:, mapB);

indxA = conv_to_index(assignments(:, mapA), A.Cardinality);  % in order to access values as ↩
    given in A structure
indxB = conv_to_index(assignments(:, mapB), B.Cardinality);

for i=1:prod(C.Cardinality)
C.Probability_Values(i)=A.Probability_Values(indxA(i))*B.Probability_Values(indxB(i));
end

end
```

Listing 10: file 'mainMaxProduct.m' under folder MaxProduct

```
clc
clear all

keys = {'A', 'S', 'L', 'T', 'E', 'X', 'B', 'F'};
value = {1, 2, 3, 4, 5, 6, 7, 8};
Let_to_Num = containers.Map(keys, value);
Num_to_Let = containers.Map(value, keys);

probab.input(1) = struct('Var_Name', [Let_to_Num('A')], 'Cardinality', [2], 'Probability_Values', [0.6 0.4]);
probab.input(2) = struct('Var_Name', [Let_to_Num('S')], 'Cardinality', [2], 'Probability_Values', [0.8 0.2]);
probab.input(3) = struct('Var_Name', [Let_to_Num('L')], 'Cardinality', [2], 'Probability_Values', [0.8 0.2]);
probab.input(4) = struct('Var_Name', [Let_to_Num('T'), Let_to_Num('A')], 'Cardinality', [2, 2], 'Probability_Values', [0.8 0.2 0.3 0.7]);
probab.input(5) = struct('Var_Name', [Let_to_Num('E'), Let_to_Num('L'), Let_to_Num('T')], 'Cardinality', [2, 2, 2], 'Probability_Values', [0.8 0.2 0.7 0.3 0.6 0.4 0.1 0.9]);
probab.input(6) = struct('Var_Name', [Let_to_Num('X'), Let_to_Num('E')], 'Cardinality', [2, 2], 'Probability_Values', [0.8 0.2 0.1 0.9]);
probab.input(7) = struct('Var_Name', [Let_to_Num('B'), Let_to_Num('S')], 'Cardinality', [2, 2], 'Probability_Values', [0.7 0.3 0.3 0.7]);
probab.input(8) = struct('Var_Name', [Let_to_Num('F'), Let_to_Num('B'), Let_to_Num('E')], 'Cardinality', [2, 2, 2], 'Probability_Values', [0.9 0.1 0.8 0.2 0.7 0.3 0.2 0.8]);

% Boundary Initializations;
PI.A = struct('Var_Name', [Let_to_Num('A')], 'Cardinality', [2], 'Probability_Values', [0.6 0.4]);
LAMBDA.A = struct('Var_Name', [Let_to_Num('A')], 'Cardinality', [2], 'Probability_Values', [1 1]);

PI.L = struct('Var_Name', [Let_to_Num('L')], 'Cardinality', [2], 'Probability_Values', [0.8 0.2]);
LAMBDA.L = struct('Var_Name', [Let_to_Num('L')], 'Cardinality', [2], 'Probability_Values', [1 1]);

PI.S = struct('Var_Name', [Let_to_Num('S')], 'Cardinality', [2], 'Probability_Values', [0.8 0.2]);
LAMBDA.S = struct('Var_Name', [Let_to_Num('S')], 'Cardinality', [2], 'Probability_Values', [1 1]);

% Evindence Initializations;
PI.X = struct('Var_Name', [Let_to_Num('X')], 'Cardinality', [2], 'Probability_Values', [0 1]);
LAMBDA.X = struct('Var_Name', [Let_to_Num('X')], 'Cardinality', [2], 'Probability_Values', [0 1]);

PI.F = struct('Var_Name', [Let_to_Num('F')], 'Cardinality', [2], 'Probability_Values', [1 0]);
LAMBDA.F = struct('Var_Name', [Let_to_Num('F')], 'Cardinality', [2], 'Probability_Values', [1 0]);

% Forming PI(T)
temp = Product(probab.input(4), PI.A);
PI.T = Max_Marginalization(temp, [Let_to_Num('A')]);

% Computing PI(E)
temp = Product(probab.input(5), Product(PI.T, PI.L));
PI.E = Max_Marginalization(temp,[Let_to_Num('L'), Let_to_Num('T')]);

```

```
% Computing PI(B)
temp = Product(probab.input(7), PI.S);
PI.B = Max_Marginalization(temp,[Let_to_Num('S')]);

% Computing LAMBDA(B)
temp = Product(probab.input(6), LAMBDA.X);
LAMBDA_X_TO_E = Max_Marginalization(temp, [Let_to_Num('X')]);
PI_E_TO_F = Product(PI.E, LAMBDA_X_TO_E);

temp = Product(probab.input(8), PI_E_TO_F);
temp_1 = Max_Marginalization(temp, [Let_to_Num('E')]);
temp_2 = Product(temp_1, LAMBDA.F);
LAMBDA.B = Max_Marginalization(temp_2, [Let_to_Num('F')]);

% Computing LAMBDA(E)
temp = Product(probab.input(8), PI.B);
temp_1 = Max_Marginalization(temp, [Let_to_Num('B')]);
temp_2 = Product(temp_1, LAMBDA.F);
LAMBDA_F_TO_E = Max_Marginalization(temp_2, [Let_to_Num('F')]);
LAMBDA.E = Product(LAMBDA_F_TO_E, LAMBDA_X_TO_E);

% Computing LAMBDA(T)
temp = Product(probab.input(5), PI.L);
temp_1 = Max_Marginalization(temp, [Let_to_Num('L')]);
temp_2 = Product(temp_1, LAMBDA.E);
LAMBDA_E_TO_T = Max_Marginalization(temp_2, [Let_to_Num('E')]);
LAMBDA.T = LAMBDA_E_TO_T;

% Computing LAMBDA(S)
temp = Product(probab.input(7), LAMBDA.B);
LAMBDA_B_TO_S = Max_Marginalization(temp, [Let_to_Num('B')]);
LAMBDA.S = LAMBDA_B_TO_S;

% Computing LAMBDA(L)
temp = Product(probab.input(5), PI.T);
temp_1 = Max_Marginalization(temp, [Let_to_Num('T')]);
temp_2 = Product(temp_1, LAMBDA.E);
LAMBDA_T_TO_E = Max_Marginalization(temp_2, [Let_to_Num('E')]);
LAMBDA.L = LAMBDA_T_TO_E;

% Computing LAMBDA(A)
temp = Product(probab.input(4), LAMBDA.T);
LAMBDA_T_TO_A = Max_Marginalization(temp, [Let_to_Num('T')]);
LAMBDA.A = LAMBDA_T_TO_A;

% Calculating Beliefs;
most_probable_assignment = [];

temp = Product(LAMBDA.A, PI.A);
A_1 = normalization(temp.Probability_Values);
[value, index] = max(A_1);
display(strcat('Random Variable '," ", Num_to_Let(1), ' takes MPE assignment ', " ",  string(rem(index, 2))))

temp = Product(LAMBDA.S, PI.S);
S_2 = normalization(temp.Probability_Values);
[value, index] = max(S_2);
display(strcat('Random Variable '," ", Num_to_Let(2), ' takes MPE assignment'," ", string(rem(index, 2))))

temp = Product(LAMBDA.L, PI.L);
L_3 = normalization(temp.Probability_Values);
```

```
[value, index] = max(L_3);
display(strcat('Random Variable '," ", Num_to_Let(3), ' takes MPE assignment'," ", string(rem(index, 2))))

temp = Product(LAMBDA.T, PI.T);
T_4 = normalization(temp.Probability_Values);
[value, index] = max(T_4);
display(strcat('Random Variable '," ", Num_to_Let(4), ' takes MPE assignment'," ", string(rem(index, 2))))

temp = Product(LAMBDA.E, PI.E);
E_5 = normalization(temp.Probability_Values);
[value, index] = max(E_5);
display(strcat('Random Variable', " ", Num_to_Let(5), ' takes MPE assignment'," ", string(rem(index, 2))))

temp = Product(LAMBDA.B, PI.B);
B_7 = normalization(temp.Probability_Values);
[value, index] = max(B_7);
display(strcat('Random Variable', " ", Num_to_Let(7), ' takes MPE assignment'," ", string(rem(index, 2))))
```

This function just carries out the max marginalization of the variable V over the input factor A.

Listing 11: Function for Max Marginalization

```
function B = Max_Marginalization(A, V)

[B.Var_Name, mapB] = setdiff(A.Var_Name, V);
B.Cardinality = A.Cardinality(mapB);
B.Probability_Values = zeros(1, prod(B.Cardinality));

assignments = conv_to_assignment(1:length(A.Probability_Values), prod(A.Cardinality));
indxB = conv_to_index(assignments(:, mapB), B.Cardinality);

for i=1:prod(B.Cardinality)
for j=1:prod(A.Cardinality)
if indxB(j)==i
B.Probability_Values(i)=max(B.Probability_Values(i), A.Probability_Values(j));
end
end
end

end
```

Rest all of the function for Max Product inference is that same as Sum product.

References

[1] Q. Ji, "3 - directed probabilistic graphical models," in *Probabilistic Graphical Models for Computer Vision.*, ser. Computer Vision and Pattern Recognition, Q. Ji, Ed. Oxford: Academic Press, 2020, pp. 31 – 129. [Online]. Available: http://www.sciencedirect.com/science/article/pii/B9780128034675000083