2018 Project 1: Approximate Inference

Wenting Li liw14@rpi.edu

November 6, 2018

1 Project description

In this project, you will implement an approximate inference method to perform inference on the CHILD Bayesian Network below.

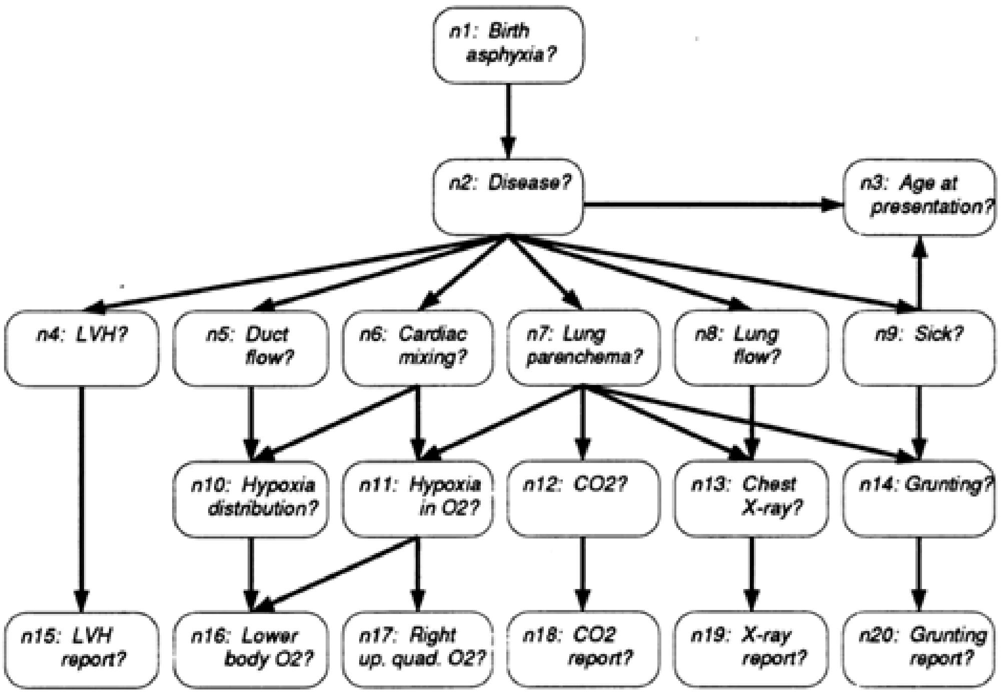

Figure 1: CHILD Bayesian Network

The CHILD network is for diagnosing congenital heart disease in a new born blue baby. It provides a mechanism so that both clinical expertise and available data can be exploited to generate diagnostic aid. Further information on the network can be found in [1]. Figure 1 shows the CHILD Bayesian Network. The basic information of the network is as follows: Number of nodes: 20; Number of arcs: 25; Number of parameters: 230; Size of node: 2,3,4,5 or 6.

Each node is discrete, with different number of states. Possible states for each variable are: Birth Asphyxia:{“yes”, “no”} ;
Disease: {“PFC”, “TGA”, “Fallot”, “PAIVS”, “TAPVD”, “Lung”} ;
Age:{“0-3 days”, “4-10 daysv11-30 days”} ;
LVH:{“yes”, “no”} ;
Duct flow: {“Lt to Rt”, “None”, “Rt to Lt”} ;
Cardiac mixing: {“None”, “Mild”, “Complete”, “Transparent”} ;

Lung parenchema: {“Normal”, “Oedema”, “Abnormal”} ;
Lung flow: {“Normal”, “Low”, “High”} ;
Sick: {“yes”, “no”} ;
Hypoxia distribution: {“equal”, “unequal”};
Hypoxia in O2: {“None”, “Moderate”, “Severe ”};
CO2: {“Normal”, “Low”, “High”};
Chest X-ray: {“Normal”, “Oligaemic”, “Plethoric”, “Grd.Glass”, “Asy/Patchy”};
Grunting: {“yes”, “no”};
LVH report: {“yes”, “no”};
Lower body O2: {“<5”, “5-12”, “12+”};
RUQ O2: {“<5”, “5-12”, “12+”};
CO2 report: {“<7.5”, “>=7.5”};
X-ray report: {“Normal”, “Oligaemic”, “Plethoric”, “Grd.Glass”, “Asy/Patchy”};
Grunting Report: {“yes”, “no”};

2 Introduction

Given the BN and its parameterization, **the aim of this project** is to perform the following inference tasks
1) Posterior probability inference: Compute p(Birth Asphyxia=yes | CO2report =“<7.5”, LVHReport=yes, and X-rayReport=Plethoric);
2) MAP inference: Disease*=argmax Disease p(Disease | CO2report =“<7.5”, LVHReport=yes, and X-rayReport=Plethoric).

There are generally four methods to approximate the inference: loopy belief, sampling, variational, and model-simulation methods. The sampling methods, including logic sampling, likelihood weighted and Markov Chain Monte Carlo (MCMC), are one of the most popular ones, and the “Gibbs Sampling Method” (GSM) is the simplest one of MCMC methods. Thus we select GSM due to its simplicity, efficiency and theoretical guarantees to achieve the above tasks. As comparison, one variational method called “mean field method” will also be derived, implemented and discussed.

3 Theoretical Analysis

In this section, the theoretical analysis of GSM will be introduced.

3.1 The theory of Gibbs Sampling and the Algorithm

The basic idea of GSM is to generate a new sample based on the values of the previous sample, and any new sample only has one different variable with the previous sample. In this way, this sampling method can estimate the inference of Bayesian networks with a large number of variables. Let $X_i, i = 1, \cdots, 17$ be , $e_i, i = 1, \cdots, 3$ be CO2report, LVHReport, X-rayReport.
The Gibbs sampling

The main idea of this method is to sample from stochastic process. According to the Markov Chain theory, as long as the samples follow the Markov Chain [1] and ergodic [2], then the limit of all the states exist, and these limits are exactly the stationary distribution. Therefore, this method need to sample after the

[1]means the current state only depends on the last state but has nothing to do with the other previous states.
[2]means that a state or all states of a chain are recurrent but not periodic.

Algorithm 1 The Single Chain Gibbs Sampling Method

```
1: Input: burn in period $t$, skip steps $k$, and iteration times $N$.
2: Initialize all the states $\mathbf{X} = \{X_1, \cdots, X_{17}\}$ by some random binary numbers according to their total
   number of states, and $\mathbf{e} = e_1, e_2, e_3$. Let $n = 0$;
3: for $i = 1, \cdots, N$ do
4:     Random pick the $j$th ($j \in [1, 17]$) state of $X$;
5:     Update the value of the state $X_j$ by obtaining the sample $x_j^{i+1}$ following the distribution
   $p(X_j|x_{-j}^i, \mathbf{e}) = p(X_j|MB(X_j))$ of (1);
6:     while other states are kept the same $x_k^{i+1} = x_k^i, k \neq j$;
7:     Form a sample of $\mathbf{x}^{i+1}$;
8:     if $i = t + nk$ then
9:         Return the sample $\mathbf{x}^{t+nk}$                    ▷ Collecting Sampling Results
10:        $n = n + 1$;
```

stochastic process goes to the "burn-in period" to ensure the sample follows the stationary distribution or the true distribution. In this case, the initial conditions do not influence the eventual estimation if sample number is sufficiently large to enter the burn-in period.

Another key point of this method is that the probability of one state variable X_k^t of a new sample at the time t can be determined by the transition model $p(X_k^{t+1}|\mathbf{x}^t/x_k^t, \mathbf{e})$, which can be locally computed based on the Markov Blanket method. Specifically, the transition model $p(X_j|\mathbf{x}_{-j}^i, \mathbf{e})$ satisfies

$$p(X_j|x_{-j}^i, \mathbf{e}) = p(X_j|MB(X_j)) \tag{1}$$

$$= \frac{p(X_j|\pi(X_j))\Pi_{i=1}^k p(Y_i|\pi(Y_i))}{\Sigma_{x_j} p(x_j|\pi(x_j))\Pi_{i=1}^k p(Y_i|\pi(Y_i))} \tag{2}$$

where $\pi(X)$ denotes the parents of X, Y_i is the ith child of X_j's kth children.

3.2 The discussion of Gibbs sampling method

Although there are rigorous theoretical guarantees that the GMS can eventually be sufficiently close to the true value, there are some open issues of implementation.

1. The burn-in period. There is no theoretical guarantees about how to set the burn-in period to ensure the exactness of the inference, and the most practical way is by trials and errors.

2. Whether a long chain or multiple chain is better to sample the results. A long chain only requires one burn-in period and skip time, but the samples are more likely to be correlated than multiple chain, but many different burn-in periods need to be identified if multiple chains are employed simultaneously. The common way in reality is to combine these two method.

3. The correlations between samples need to be reduced, thus a skip time need to be defined between any two samples, but the specific way of selecting a reasonable skip time can not be clearly specified. Due to these open questions, the effects of these parameters will be tested and discussed in our problem numerically.

3.3 The theory and algorithms of mean field method

Derivations of mean field method

The main idea of mean field method is to compute a surrogate distribution $q(\mathbf{x}|\beta), \mathbf{x}, \beta \in R^n$ to approximate the original complex distribution $p(\mathbf{X}|\mathbf{E})$, where $\mathbf{X}$ are the unknown variables, $\mathbf{E}$ are evidence, and the choice of this surrogate distribution is by assuming that all the variables are independent, that is,

$$q(\mathbf{x}|\beta) = \Pi_{i=1}^{n} q(x_i|\beta_i) \tag{3}$$

Let $ELBO(x, \beta) = f(x, \beta) = -\Sigma_{\mathbf{x}} q(\mathbf{x}|\beta) \log(q(\mathbf{x}|\beta)) + \Sigma_{\mathbf{x}} q(\mathbf{x}|\beta) \log(p(\mathbf{X}, \mathbf{E}))$. To approximate the unknown distribution $P(\mathbf{X}|\mathbf{E})$, we can obtain the parameters β by maximizing $f(x, \beta)$.

$$\begin{aligned}
f(x,\beta) &= -\Sigma_{\mathbf{x}} q(\mathbf{x}|\beta) \log(q(\mathbf{x}|\beta)) + \Sigma_{\mathbf{x}} q(\mathbf{x}|\beta) \log(p(\mathbf{X}, \mathbf{E})) && (4)\\
&= -\Sigma_{i=1}^{n} \Sigma_{x_i} q(x_i|\beta_i) \log(q(x_i|\beta_i)) + \Sigma_{\mathbf{x}} q(\mathbf{x}|\beta) \log(p(\mathbf{X}, \mathbf{E})) && (5)\\
&= -\Sigma_{i=1}^{n} \Sigma_{x_i} q(x_i|\beta_i) \log(q(x_i|\beta_i)) + \Sigma_{i=1}^{n} q(x_i|\beta_i) \Sigma_{\mathbf{X}_{-i}} \Pi_{x_j \in \mathbf{X}_{-i}} q(x_j|\beta_j)) \log(p(\mathbf{X}_{-i}, x_i, \mathbf{E}) && (6)\\
&= -\Sigma_{i=1}^{n} \Sigma_{x_i} q(x_i|\beta_i) \log(q(x_i|\beta_i)) + E_{q(\mathbf{X}_{-i})}(\log(p(\mathbf{X}_{-i}, x_i, \mathbf{E})) && (7)\\
& && (8)
\end{aligned}$$

where $E_{q(\mathbf{X}_{-i})}(\log(p(\mathbf{X}_{-i}, x_i, \mathbf{E}))$ denotes the expectation of $\log(p(\mathbf{X}_{-i}, x_i, \mathbf{E})$ based on the probability $\Pi_{x_j \in \mathbf{X}_{-i}} q(x_j|\beta_j))$. To satisfy the first order optimal condition, we set the derivative of (4) to be 0.

For our discrete BN, we can derive the gradient of $\beta_{ik}, i = 1, \cdots, n, k = 1, \cdots, K_i$, where K_i is the number of states of the ith node.

$$\frac{dq(x_i|\beta_i)}{d\beta_{ik}} = \begin{cases} 1 & \text{if } x_i = k, k < K_i \\ 0 & \text{if } x_i \neq k, k < K_i \\ -1 & \text{if } x_i = K_i \end{cases} \tag{9}$$

Then we obtain the gradient of $\frac{\partial(f)}{\partial(\beta_{ik})}$,

$$\frac{\partial(f)}{\partial(\beta_{ik})} = -\Sigma_{x_i}(1 + \log(q(x_i|\beta_i)) \frac{dq(x_i|\beta_i)}{d\beta_{ik}}) + \Sigma_{x_i} \frac{dq(x_i|\beta_i)}{d\beta_{ik}} E_{q(\mathbf{X}_{-i})} \log(p(\mathbf{X}_{-i}, x_i, \mathbf{E}) = 0 \tag{10}$$

By plugging (9) to (10), we acquire the equation of

$$\log(q(x_i = k|\beta_i) = E_{q(\mathbf{X}_{-i})} \log(p(\mathbf{X}_{-i}, x_i = k, \mathbf{E}) + log(q(x_i = K_i|\beta_i) - E_{q(\mathbf{X}_{-i})} \log(p(\mathbf{X}_{-i}, x_i = K_i, \mathbf{E}) \tag{11}$$

Take the exponential operator on both side, then we obtain the closed form expression of β_{ik},

$$\beta_{ik} = \frac{E_{q(\mathbf{X}_{-i})} \log(p(\mathbf{X}_{-i}, x_i = k, \mathbf{E})}{\Sigma_{l=1}^{K_i} E_{q(\mathbf{X}_{-i})} \log(p(\mathbf{X}_{-i}, x_i = l, \mathbf{E})} \tag{12}$$

Thus we can update all the $\beta_{ik}, i = 1, \cdots, n, k = 1, \cdots, K_i$ until converge.

Furthermore, we can simplify the $E_{q(\mathbf{X}_{-i})} \log(p(\mathbf{X}_{-i}, x_i = l, \mathbf{E}))$ by the Bayesian chain rule, which represents the joint probability $p(\mathbf{X}_{-i}, x_i = l, \mathbf{E})$ by the product of condition probability $\Pi_{m=1}^{n} p(x_m|\pi(x_m))$.

$$\begin{aligned}
E_{q(\mathbf{X}_{-i})} \log(p(\mathbf{X}_{-i}, x_i = k, \mathbf{E})) &= E_{q(\mathbf{X}_{-i})} \log(\Pi_{m=1}^{n} p(x_m|\pi(x_m)) && (13)\\
&= \Sigma_{m=1}^{n} E_{q(\mathbf{X}_{-i})} \log(p(x_m|\pi(x_m)) && (14)
\end{aligned}$$

where $x_m = k$ if $m = i$, and $x_m = e$ if $m \in \mathbf{E}$.

Algorithm 2 The mean field method 2

```
1: Input: The evidence $\mathbf{e}$ and unknown variables X of BN, and the convergence threshold $\sigma$.
2: Output: The parameters $\beta_{ik}, i = 1, \cdots, n, k = 1, \cdots, K_i$, where $n$ is the number of nodes and $K_i$ is the number of states of the $i$th node.
3: Initialize $\beta_{ik}$ by some random values.
4: while $\beta_{ik}$ not converge do
5:     Compute the $\beta_{ik}$ by (9), (10) and (12).
6:     Return $\beta_i, i = 1, \cdots, n$
```

Discussion of the mean field method The primary issue of the mean field method is the accuracy. There always exist some difference between $q(\mathbf{X}|\beta)$ with $p(\mathbf{X}, |\mathbf{E})$unless all the nodes of the BN are independent. Moreover, the mean field method also relies on the initialization. The advantage is that the close formed solution of β_{ik} id deduced without employing the gradient decent to estimate the parameters.

4 Numerical Results

4.1 Task 1: Inference result

P(Birth Asphyxia=yes | CO2report :"<7.5", LVHReport:yes, X-rayReport:Plethoric) = 0.0813.

The basic parameters are summarized as follows. Sample number is 100000, burn-in period is 10000 , skip time is 100. The algorithm is implemented by MATLAB, and the codes are attached.

4.2 Task 2: MAP inference

Disease* = $\arg\max$ p(Disease | CO2report :"<7.5", LVHReport:yes, X-rayReport:Plethoric)= "PAIVS" with the probability 0.6873.

As the node "Disease" has 6 states, {"PFC", "TGA", "Fallot", "PAIVS", "TAPVD", "Lung"}. The conditional probability of all these states are also computed: p(Disease | CO2report :"<7.5", LVHReport:yes, X-rayReport:Plethoric) = [0.0237 0.1028 0.1428 0.6873 0.0109 0.0325]. Thus p(Disease = TGA | CO2report :"<7.5", LVHReport:yes, X-rayReport:Plethoric) is the maximum.

Evaluation of Gibbs Sampling Algorithm

The advantages of Gibbs Sampling Algorithm 1 have four aspects: 1) Simplicity of implementation; 2) Theoretical guarantees; 3) Parallelizable and hardware implementation; 4) Suitable for a large number of network sizes. On the other hand, the disadvantages include: 1) Can be slow to converge, and it is difficult to determine the convergence especially there is some extreme probability or the evidence probability is very low; 2) The determination of some parameters has no theoretical guarantees, such as "burn-in period", "skip steps", and how to avoid the data correlations; 3) No rigorous criterion to illustrate whether the chain works or not.

4.3 The Effect of Sampling Times on the Inference

We keep the same setup of skip steps being 100, , the same initialization,burn-in time being 10000, and then change the sample numbers from 10 to 100000. From the results we can observe that after 10000 sample times, the inference result converges, thus for the latter results the sample times is set to be 10000, but when the sample number is too small (less than 500 in this project), for example, when sample times

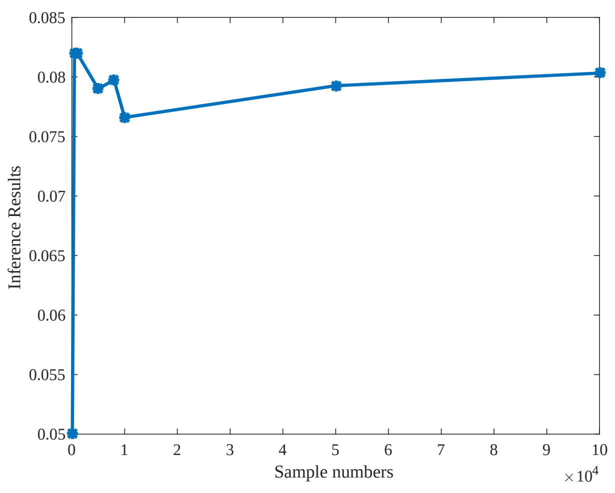

Figure 2: The effects of sampling time with inference results

Table 1: The effects of sampling time with inference results

Sample Times	50	100	500	1000	5000	8000	10000	50000	100000
Inference	0.1000	0.0500	0.0820	0.0820	0.0790	0.0798	0.0766	0.0793	0.0813

are 100, the inference is 0.05 far from the true value. Thus if not sufficient number of samples are provided, the inference results have relatively large errors.

4.4 The Effect of Initialization on the Inference

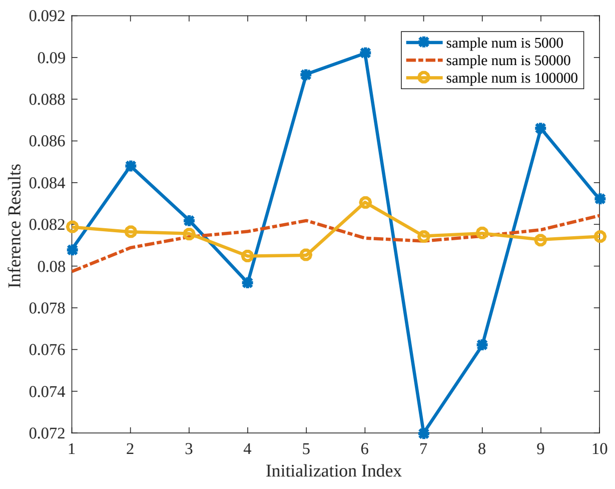

Figure 3: The effects of different initialization with inference results for various sample numbers

Table 2: The inference results when using different initialization with 100000 samples

Inference	0.0818	0.0816	0.0815	0.0804	0.0805	0.0830	0.08144	0.0815	0.0812	0.0814

4.5 The Effect of Burn-in Time on the Inference

By fixing burn-in period to be 10000, skip time to be 100. Considering the significance of sample numbers, we analyze the relations between different initial conditions to the inference results when the sample number is 5000, 50000 and 100000 respectively. From the results shown in the Table 2, we can observe that when the samples are relatively small, like 5000 samples, different initial conditions may cause the inference results to be various, but when sufficient number of samples are generated, like the orange line when sample number is 100000, the influence of different initial conditions is almost invisible. Therefore, the inference results are not changed by different initialization. We keep the same setup of skip

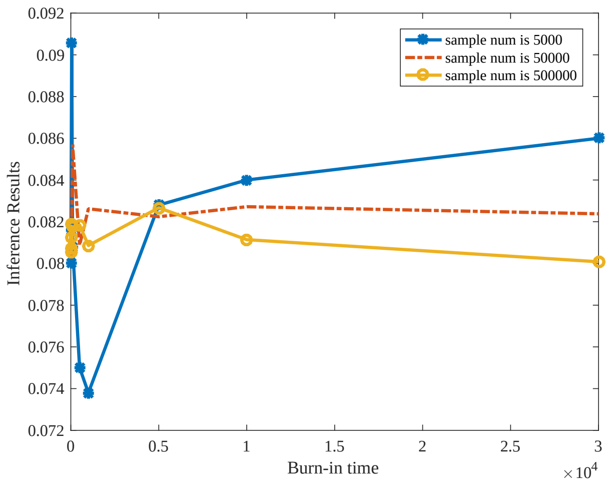

Figure 4: (The effects of burn-in period with inference results when different samples

Table 3: The effects of burn-in period with inference results when sample number is 100000

Burn-in Period	10	50	100	500	1000	5000	10000	30000
Inference	0.0812	0.0805	0.0807	0.0817	0.0818	0.0808	0.0826	0.0811

steps being 100, the same initialization, and change the burn-in period from 1 to 50000 to study the relations between inference and burn-in period. These relations are illustrated when sample numbers are small and large respectively in the Figure 4. When only generate 5000 samples, the inference results can be different for different initial conditions, but when sample numbers increase to 50000 and even 500000, the results are almost not influenced by various initial conditions. Thus if sample numbers are small, like the blue curves, a larger burn-in period is more likely to converge to the true value but if the burn-in period is too small (less than 2000), the inference can have a relatively large error. Compare the yellow, blue and red curves, we can find that when the sample numbers are larger enough (more than 500000), even a relatively smaller burn-in time still can be close to the true inference. For example, in Table 3, when the sample number is larger (50000 in this case), the inference is close to the true value even when the burn-in period is smaller than 100. Therefore, if the sample numbers are small, a larger burn-in period is more likely to converge to the true value, and if the sample numbers are sufficiently larger, the influence of various burn-in period becomes not obvious.

4.6 The Effect of Skip Time on the Inference

We keep the same setup of burn-in time being 10000, the same initialization, and then change the skip time from 1 to 600 when the sample number is 5000, 50000 and 500000 respectively, as shown on the left

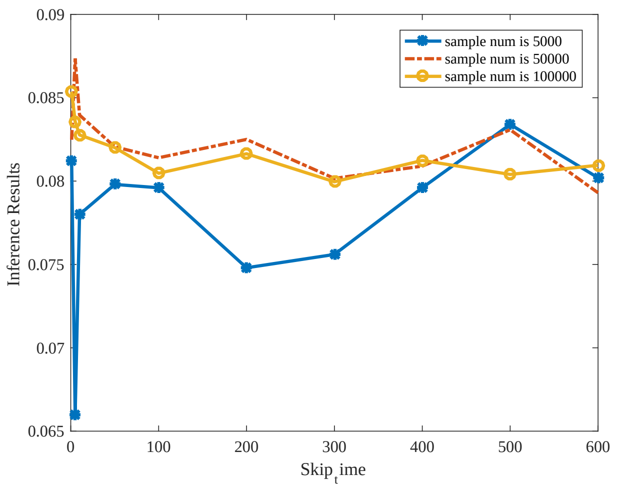

Figure 5: The effects of skip time with inference results when sample number changes from 1000 to 100000

Table 4: The effects of skip time with inference results when sample number is 100000

Skip Times (steps)	5	10	50	100	200	300	400	500	600
Inference	0.0853	0.0835	0.0827	0.0820	0.0804	0.0816	0.0799	0.0812	0.0804

figure of Figure 5. When the samples are sufficiently large, like the yellow and red lines, we can observe that when skip time larger than 50, the inference results converge to the same value. When the samples are relatively small, like the blue line, a too large or too small skip can both produce an inference with larger errors and the results are not stable. Thus to reach a reliable inference result, the sample numbers should large enough, here larger than 50000 is suggested, and then a large skip time can ensure the convergence of the inference.

In generally, the influences of initial conditions, skip time, burn in period all can be largely reduced when sample number is sufficiently large. This is a beneficial for the complication of choosing proper parameters. As long as the computer has enough computation capability, there is no need to tune the best parameters but only need to generate enough samples. This property enable Gibbs sampling method has the potential to be widely applied to more complicated and large scale BN and may solve some more challenging problem in the future.

4.7 The Results of Mean Field Method

The mean field method is also implemented, and the results of the two tasks are summarized as follows:

4.7.1 Task 1: Inference result

P(Birth Asphyxia=yes | CO2report :“<7.5”, LVHReport:yes, X-rayReport:Plethoric) = 0.0673.

The speed of convergence is very fast, which only need about 6 iterations although more than 1000 iterations are tested. As we deduced theoretically, there is still a gap between the true inference and the estimated result since there are some independence of the BN ignored in this algorithm. Thus mean field method can provide a lower bound of the true inference.

4.7.2 Task 2: MAP inference

Disease* = arg max p(Disease | CO2report :“<7.5”, LVHReport:yes, X-rayReport:Plethoric) = “PAIVS” with the probability 0.9437.

The estimated inference by mean field method of “Disease = PAIVS” has some difference with that of Gibbs sampling method, but the state of “PAIVS” is correctly determined as the one with the highest probability.

5 Conclusions and Discussion

Both Gibbs sampling and mean field methods can approximately estimate the inference. Compared with mean field method, Gibbs sampling method is much simpler and accurate. One practical tip of setting the parameters is that first ensure the sample numbers are larger enough, and then select relative larger skip time, burn-in period and random initialization can usually converge to close to the true value, and as long as the samples are sufficiently larger, the influence of different parameters can be largely reduced. Therefore, Gibbs sampling methods are easily applied widely, especially when the capacity of computer is enhanced significantly in the future. For the mean field method, the accuracy is difficult to be ensured, especially when the BN has dense links or dependence, and the computation complexity increases when BN are densely connected.

Appendix: Matlab code

Instructions of running the codes of Gibbs Sampling Method This function is to compute the posterior probability inference “Inference” and the MAP inference “MAP” of this project through Gibbs sampling method of multiple chains. The input parameters include burn in period, skip time, sample numbers and the number of chains. Notice that the absolute path of the parameter and structure files “parameter.mat” and “structure.mat” should be modified according to location in your computer.

The Gibbs sampling of Multiple Chains

```

% Gibbs Sampling Method to compute P(X|E)
function [ Inference , MAP] = Gibbs_sample(burn_in , skip , sample_num  )
%input:
        %burn_in -- The burn-in period;
        %skip -- The skip time;
        %sample_num -- The number of samples;
        %num_chain -- The number of Markov chains that user want to use
           to generate the samples
%output:
        %Inference -- The posterier inference p(Birth Asphyxia=yes  |
           CO2report :`` < 7.5'', LVHReport:yes, %X-rayReport:Plethoric)
        % MAP -- The MAP inference of Disease
        % load data
        load('.\parameter.mat');
        load('.\structure.mat');
        %% parameters
        total_num = 20;
```

```
    unknown_num = 17;
    times = 1;
    unknown_ind = [1:14, 16:17, 20];
    variable_names = {BirthAsphyxia, Disease, Age, LVH, DuctFlow, CardiacMixing, LungParench,...
        LungFlow, Sick, HypDistrib, HypoxiaInO2, CO2, ChestXray, Grunting, LVHreport, LowerBodyO2,...
        RUQO2, CO2Report, XrayReport, GruntingReport  };
    iter_num = 100000000;
    %% initialize
    for k = 1: total_num
        X(k) = randi(domain_counts(k));
    end
    X(18) = 1; X(15) = 1; X(19) = 3;
    samples = [];
    for i = 1:iter_num
        %% randomly pick one state
        j = randi(unknown_num);
        index = unknown_ind(j);
        %% update X(index)
        prob = zeros(domain_counts(index),1);
        children = find (dag( index,:) == 1);
        child_num = sum(dag( index,:));
        for s = 1:domain_counts(index)
            X(1,index) = s;
            [ P_index ] = P_node(variable_names,domain_counts, index, dag,  X  );
            temp =1;
            for l=1:child_num
                [ P_child] = P_node(variable_names,domain_counts, children(l) , dag,    X );
                temp = temp * P_child;
            end
            prob(s) = P_index * temp;
        end
        prob = prob./sum(prob);
        X(1,index) = sample_k(prob);

        %% sample after t burn-in period with skip time k
        if i == burn_in+skip*times
            samples = [samples; X];
            times = times + 1;
        end
        if size(samples, 1) == sample_num
            display('Samples are enough!')
            break
        end
    end
    Inference = numel(find(samples(:,1) == 1))/ size(samples,1)
    MAP_Infer = zeros(1,6);
```

```
        for k =1:6
                MAP_Infer(k) = numel(find(samples(:,2) == k))/ size(
                    samples,1);
        end
        MAP = find(MAP_Infer == max(MAP_Infer)) ;
end

% This sub-function is to generate a new sample given the current states
    X and some parameters.
function [X ] = gener_sample(unknown_num,unknown_ind,variable_names,dag,
   X)
% input:
        % unknown_num -- The number of variables that are unknown;
        % unknown_ind -- The index of the unknown variables;
        % variable_names -- The variable names;
        % dag -- The relations between the parents and children
        % X -- The current states of all the nodes
% output:
        % X -- The updated states of all the nodes
        %% randomly pick one state
    j = randi(unknown_num);
    index = unknown_ind(j);
 %% update X(index)
    prob = zeros(domain_counts(index),1);
    children = find (dag( index,:) == 1);
    child_num = sum(dag( index,:));
    for s = 1:domain_counts(index)
        X(1,index) = s;
        [ P_index ] = P_node(variable_names,domain_counts, index, dag,
           X  );
        temp =1;
        for l=1:child_num
            [ P_child] = P_node(variable_names,domain_counts,  children(
               l) , dag,    X );
            temp = temp * P_child;
        end
        prob(s) = P_index * temp;
    end
    prob = prob./sum(prob);
    X(1,index) = sample_k(prob);
end

% This sub-function is to compute the probability of the node 'index'
   given its Markov Blanket.
%input:
        % variable_names --  The conditional probability;
        % domain_counts -- all the number of states of nodes;
        % index -- The index of the node
        % dag -- The relations of the parents of nodes
        % X -- The current states of all nodes
% output:
        % P_index -- Conditional probability of node index
function [ P_index ] = P_node( variable_names, domain_counts,index, dag,
    X  )
    CPT = variable_names{1,index};
    Pa_num = sum(dag(:,index));
    if Pa_num == 0
        prob = CPT;
    elseif Pa_num == 1
        Pa = find(dag(:, index)==1);
        Pa_state = X(Pa);
        prob = CPT(:,Pa_state);
    elseif Pa_num == 2
        all_pa = find(dag(:,index)==1);
        Pa1 = all_pa(1); Pa2 = all_pa(2);
        Pa_state = (X(Pa1)-1)*(domain_counts(Pa2) ) + X(Pa2);
        prob = CPT(:,Pa_state);
    end
    P_index = prob(X(index));

end


% This sub-function is to generate one new state of one node given the
    probability 'prob'
function x = sample_k(prob)
        u = rand;
        prob0 = [0; prob];
        for i = 1:numel(prob0)
            if sum(prob0(1:i )) > u
                x = i-1 ;
                break
            end
        end
end
```

Instructions of running the codes of Mean Field Method This function is to compute the posterior probability inference “Inference” and the MAP inference “MAP” of this project through Mean Field Method. The input parameters include the absolute path ”path parameters” and ”path structure”of files of “parameter.mat” and “structure.mat” should be modified according to location in your computer.

The Mean Field Method

```
% mean field method
function [ Inference, MAP] = mean_field(path_parameters, path_structure
  )
% input:
        % path_parameters -- the path of the parameter.mat file;
        % path_structure -- the path of the structure.mat file;
```

```matlab
%output:
        %Inference -- The posterier inference p(Birth Asphyxia=yes  | CO2report :`` < 7.5'', LVHReport:yes, %X-rayReport:Plethoric)
        % MAP -- The MAP inference of Disease
        % load datasets
        load(path_parameters);
        load(path_structure);
        %% parameters
        thres = 1e-7;
        iter_num = 10000000;
        total_num = 20;
        unknown_num = 17;
        unknown_ind = [1:14, 16:17, 20];
        variable_names = {BirthAsphyxia, Disease, Age, LVH, DuctFlow, CardiacMixing, LungParench,...
            LungFlow, Sick, HypDistrib, HypoxiaInO2, CO2, ChestXray, Grunting, LVHreport, LowerBodyO2,...
            RUQO2, CO2Report, XrayReport, GruntingReport  };
        %% initialize
        beta = zeros(total_num, 6);
        err = zeros(total_num, 6);
        all_results = [];
        for k = 1: total_num
            for i = 1:domain_counts(k)
                    beta(k,i) = rand;
            end
            beta(k,:) = beta(k,:)./sum(beta(k,:));
        end
        all_results = zeros(1,iter_num);
        for iter =1:iter_num
            beta_old = beta;
            for i =  1:total_num %update the ith node with the state of s
                Eqi = zeros(1,domain_counts(i));
                for s = 1:domain_counts(i)
                    [ Eqi(s)  ] = Eq_i(total_num, dag, i , s, domain_counts, beta,variable_names  );
                end
                for k = 1: domain_counts(i)
                    beta(i,k) = exp(Eqi(k))/sum(exp(Eqi)) ;
                    err(i,k) = norm(beta(i,k) - beta_old(i,k));
                end
            end
            if min(min(err)) < thres
                break
            end
            all_results(iter) = beta(1,1) ;
            if mod(iter, 1) ==1
                fprintf('The result is %.4f \n', beta(1,1));
            end
        end
        Pl_inf  = beta(1,1)
        MAP_inference = find(beta(2,:) == max(beta(2,:)))
end

% This sub-function is to compute the expectation E_q(X_{-i})(p(X, x_i = s, E))
function [ results ] = Eq_i(total_num, dag, index, state, domain_counts, beta,variable_names  )
%input:
        % i -- the node i;
        % s -- the state of node i;
% output:
        % Eq -- E_{q(\X_{-i})} (\log(p(\X_{-i}, x_i, \E))
    results = 0;
    for m =   1: total_num
        all_pa = find(dag(:,m)==1);
        all_node = [all_pa;  m];
        [all_x] = list_all_x(index, state, all_node,domain_counts,[],  []);
        Eq = 0;
        for i = 1: size(all_x,1)
            X = all_x(i,:);
            %% compute q
            q_no_i = 1;
            for n = 1: numel(all_node)
                if all_node(n) ~= index  && X(n) < domain_counts(all_node(n))
                   q_no_i = q_no_i * beta(all_node(n), X(n));
                elseif all_node(n) ~= index  && X(n) == domain_counts(all_node(n))
                   K = domain_counts(all_node(n))-1;
                   q_no_i = q_no_i * (1-sum(beta(all_node(n),1:K )));
                end
            end
            %% Pm
            CPT = variable_names{1,m};
            Pa_num = sum(dag(:,m));
            if Pa_num == 0
                prob = CPT;
            elseif Pa_num == 1
                Pa_state  = X(1);
                prob = CPT(:,Pa_state);
            elseif Pa_num == 2
                Pa_state  = ( X(1)-1)*(domain_counts(all_pa(2)) ) + X(2);
                prob = CPT(:,Pa_state);
            end
            Pm = prob(X(numel(X)));
            if Pm ==0
```

```matlab
                break
            end
            Eq = Eq + log(Pm) * q_no_i;
        end
        results = results + Eq;
    end
end

% This sub-function is to compute the possible x values
%input:
        % index -- The node i
        % state -- The state of node i
        % all_pa -- all the nodes that are considered;
        % domian_counts -- The number of states
        % sample -- The one sample
        % all_x -- The collection of samples
% output:
        % all_x -- all possible results
function [all_x] = list_all_x( index, state, all_pa,domain_counts,sample, all_x )
    if size(sample,2 ) == numel(all_pa)
        all_x =[all_x; sample] ;
        return
    end
    domain_counts(index) = 1;
    domain_counts(15) = 1;
    domain_counts(18) = 1;
    domain_counts(19) = 1;
    for j=1:domain_counts(all_pa(size(sample,2)+1))
            if all_pa(size(sample,2)+1) == 15 || all_pa(size(sample,2)+1) == 18
                sample = [sample 1];
            elseif all_pa(size(sample,2)+1) == 19
                sample = [sample 3];
            elseif all_pa(size(sample,2)+1) == index
                sample = [sample state];
            else
                sample = [sample j];
            end
            if size(sample,2 ) <= numel(all_pa)
                [all_x] = list_all_x(index, state, all_pa,domain_counts, sample, all_x );
                sample(end) = [];
            end
    end
end
```