

2018 Project 1: Approximate Inference

Wenting Li

liw14@rpi.edu

February 10, 2019

1 Project description

This project is to implement an approximate inference method to perform inference on the CHILD Bayesian Network below.

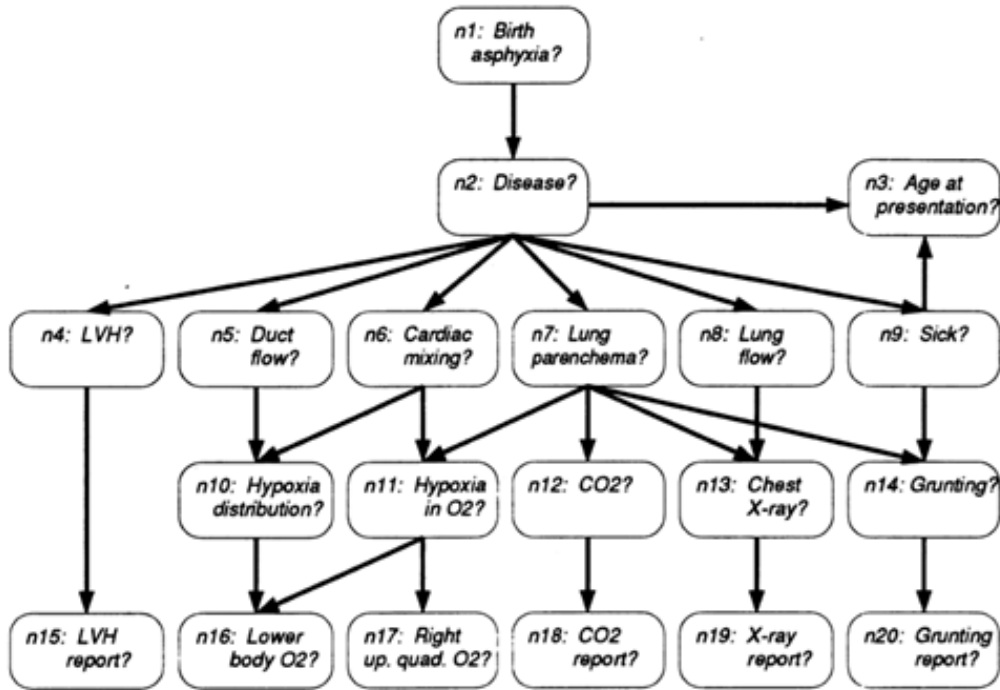


Figure 1: CHILD Bayesian Network

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 days", "11-30 days"} ;
LVH: {"yes", "no"} ;
Duct flow: {"Lt to Rt", "None", "Rt to Lt"} ;
Cardiac mixing: {"None", "Mild", "Complete", "Transparent"} ;

Lung parenchyma: {"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 $p(\text{Birth Asphyxia}=\text{yes} \mid \text{CO2report} = "<7.5", \text{LVHReport}=\text{yes}, \text{and X-rayReport}=\text{Plethoric})$;
- 2) MAP inference: $\text{Disease}^* = \arg\max \text{Disease } p(\text{Disease} \mid \text{CO2report} = "<7.5", \text{LVHReport}=\text{yes}, \text{and X-rayReport}=\text{Plethoric})$.

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 $X_i, i = 1, \dots, 17$ be , $e_i, i = 1, \dots, 3$ 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 Chain¹ and ergodic², then the limit of all the states exist, and these limits are exactly the stationary distribution. Therefore, this method need to sample after the

¹means the current state only depends on the last state but has nothing to do with the other previous states.

²means 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 $\mathbf{X} = \{X_1, \dots, X_{17}\}$ 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** $i = 1, \dots, N$ **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

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

$$= \frac{p(X_j|\pi(X_j))\prod_{i=1}^k p(Y_i|\pi(Y_i))}{\sum_{x_j} p(x_j|\pi(x_j))\prod_{i=1}^k p(Y_i|\pi(Y_i))} \quad (2)$$

where $\pi(X)$ denotes the parents of X , Y_i is the i th child of X_j 's k th 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,

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

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

$$f(x, \beta) = -\sum_{\mathbf{x}} q(\mathbf{x}|\beta) \log(q(\mathbf{x}|\beta)) + \sum_{\mathbf{x}} q(\mathbf{x}|\beta) \log(p(\mathbf{X}, \mathbf{E})) \quad (4)$$

$$= -\sum_{i=1}^n \sum_{x_i} q(x_i|\beta_i) \log(q(x_i|\beta_i)) + \sum_{\mathbf{x}} q(\mathbf{x}|\beta) \log(p(\mathbf{X}, \mathbf{E})) \quad (5)$$

$$= -\sum_{i=1}^n \sum_{x_i} q(x_i|\beta_i) \log(q(x_i|\beta_i)) + \sum_{i=1}^n q(x_i|\beta_i) \sum_{\mathbf{x}_{-i}} \prod_{x_j \in \mathbf{x}_{-i}} q(x_j|\beta_j) \log(p(\mathbf{X}_{-i}, x_i, \mathbf{E})) \quad (6)$$

$$= -\sum_{i=1}^n \sum_{x_i} q(x_i|\beta_i) \log(q(x_i|\beta_i)) + E_{q(\mathbf{x}_{-i})}(\log(p(\mathbf{X}_{-i}, x_i, \mathbf{E}))) \quad (7)$$

$$(8)$$

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 $\prod_{x_j \in \mathbf{x}_{-i}} q(x_j|\beta_j)$. 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 $\beta_{ik}, i = 1, \dots, n, k = 1, \dots, K_i$, where K_i is the number of states of the i th node.

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

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

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

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

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

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

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

Thus we can update all the $\beta_{ik}, i = 1, \dots, n, k = 1, \dots, K_i$ 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 $\prod_{m=1}^n p(x_m|\pi(x_m))$.

$$E_{q(\mathbf{x}_{-i})} \log(p(\mathbf{X}_{-i}, x_i = k, \mathbf{E})) = E_{q(\mathbf{x}_{-i})} \log(\prod_{m=1}^n p(x_m|\pi(x_m))) \quad (13)$$

$$= \sum_{m=1}^n E_{q(\mathbf{x}_{-i})} \log(p(x_m|\pi(x_m))) \quad (14)$$

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

Discussion of the mean field method The primary issue of the mean field method is the accuracy. There always exist some difference between $q(X|\beta)$ with $p(X, |E)$ 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} is deduced without employing the gradient decent to estimate the parameters.

4 Numerical Results

4.1 Task 1: Inference result

$P(\text{Birth Asphyxia}=\text{yes} \mid \text{CO2report} : "<7.5", \text{LVHReport}:\text{yes}, \text{X-rayReport}:\text{Plethoric}) = 0.0813$.

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

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

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

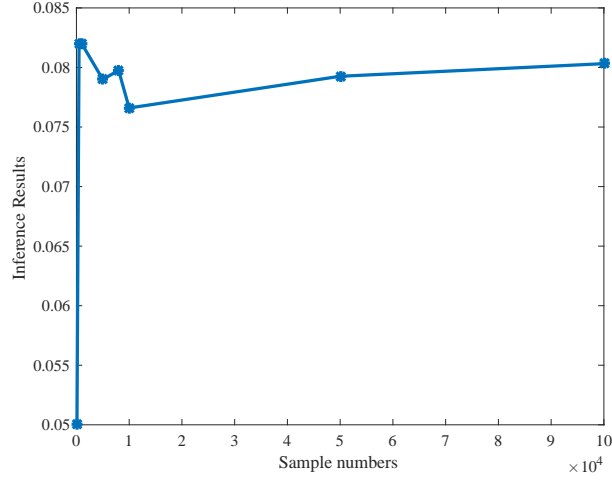


Figure 2: The effects of sampling time with inference results

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

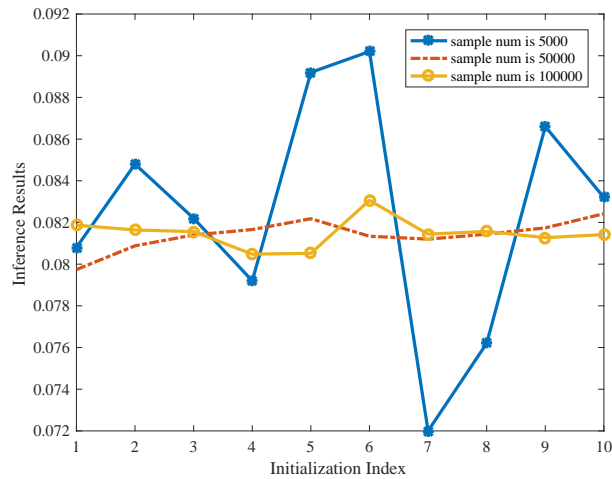


Figure 3: The effects of different initialization with inference results for various sample numbers

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

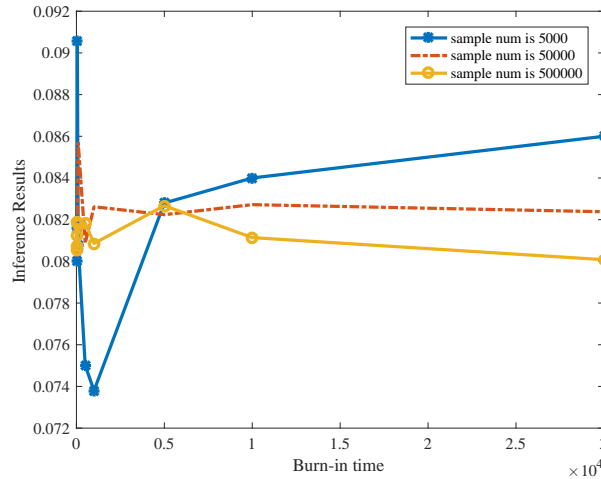


Figure 4: (The effects of burn-in period with inference results when different samples

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

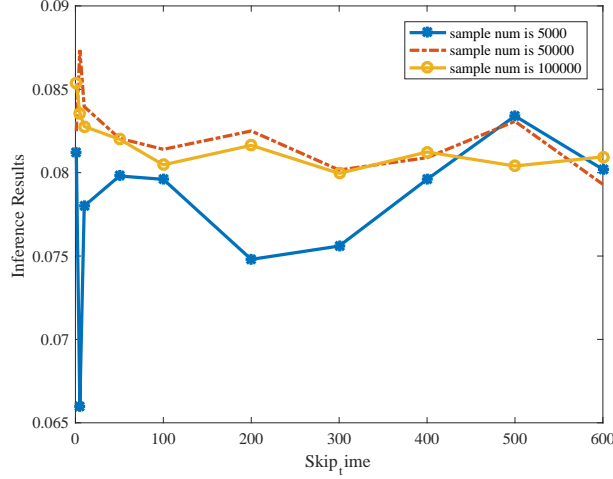


Figure 5: The effects of skip time with inference results when sample number changes from 1000 to 100000

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

$$P(\text{Birth Asphyxia}=\text{yes} \mid \text{CO2report} : "<7.5", \text{LVHReport}:\text{yes}, \text{X-rayReport}:\text{Plethoric}) = 0.0673.$$

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

$\text{Disease}^* = \arg \max p(\text{Disease} \mid \text{CO2report} : "<7.5", \text{LVHReport}:\text{yes}, \text{X-rayReport}:\text{Plethoric}) = \text{"PAIVS"}$ 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

```
1
2 % Gibbs Sampling Method to compute P(X|E)
3 function [ Inference , MAP] = Gibbs_sample(burn_in , skip , sample_num )
4 %input:
5     %burn_in — The burn-in period;
6     %skip — The skip time;
7     %sample_num — The number of samples;
8     %num_chain — The number of Markov chains that user want to use
9                 to generate the samples
10 %output:
11 %Inference — The posterier inference p(Birth Asphyxia=yes |
12             CO2report : `` < 7.5 ' ', LVHReport:yes , %X-rayReport:Plethoric)
13 % MAP — The MAP inference of Disease
14 % load data
15 load( '.\parameter.mat' );
16 load( '.\structure.mat' );
17 %% parameters
18 total_num = 20;
```

```

17 unknown_num = 17;
18 times = 1;
19 unknown_ind = [1:14, 16:17, 20];
20 variable_names = {BirthAsphyxia, Disease, Age, LVH, DuctFlow,
21   CardiacMixing, LungParench,...
22   LungFlow, Sick, HypDistrib, HypoxiaInO2, CO2, ChestXray,
23   Grunting, LVHreport, LowerBodyO2,...
24   RUQO2, CO2Report, XrayReport, GruntingReport };
25 iter_num = 100000000;
26 %% initialize
27 for k = 1: total_num
28   X(k) = randi(domain_counts(k));
29 end
30 X(18) = 1; X(15) = 1; X(19) = 3;
31 samples = [];
32 for i = 1:iter_num
33   %% randomly pick one state
34   j = randi(unknown_num);
35   index = unknown_ind(j);
36   %% update X(index)
37   prob = zeros(domain_counts(index),1);
38   children = find (dag( index,:) == 1);
39   child_num = sum(dag( index,:));
40   for s = 1:domain_counts(index)
41     X(1,index) = s;
42     [ P_index ] = P_node(variable_names, domain_counts, index
43       , dag, X );
44     temp =1;
45     for l=1:child_num
46       [ P_child ] = P_node(variable_names, domain_counts,
47         children(l) , dag, X );
48       temp = temp * P_child;
49     end
50     prob(s) = P_index * temp;
51   end
52   prob = prob./sum(prob);
53   X(1,index) = sample_k(prob);
54
55   %% sample after t burn-in period with skip time k
56   if i == burn_in+skip*times
57     samples = [samples; X];
58     times = times + 1;
59   end
60   if size(samples, 1) == sample_num
61     display('Samples are enough!')
62     break
63   end
64 end
65 Inference = numel(find(samples(:,1) == 1))/ size(samples,1)
66 MAP_Infer = zeros(1,6);

```

```

63     for k = 1:6
64         MAP_Infer(k) = numel(find(samples(:,2) == k))/ size(
            samples,1);
65     end
66     MAP = find(MAP_Infer == max(MAP_Infer)) ;
67 end
68
69 % This sub-function is to generate a new sample given the current states
    X and some parameters.
70 function [X ] = gener_sample(unknown_num,unknown_ind,variable_names,dag,
    X)
71 % input:
72     % unknown_num — The number of variables that are unknown;
73     % unknown_ind — The index of the unknown variables;
74     % variable_names — The variable names;
75     % dag — The relations between the parents and children
76     % X — The current states of all the nodes
77 % output:
78     % X — The updated states of all the nodes
79     %% randomly pick one state
80     j = randi(unknown_num);
81     index = unknown_ind(j);
82     %% update X(index)
83     prob = zeros(domain_counts(index),1);
84     children = find (dag( index,:) == 1);
85     child_num = sum(dag( index,:));
86     for s = 1:domain_counts(index)
87         X(1,index) = s;
88         [ P_index ] = P_node(variable_names,domain_counts, index, dag,
            X );
89         temp = 1;
90         for l=1:child_num
91             [ P_child] = P_node(variable_names,domain_counts, children(
                l) , dag, X );
92             temp = temp * P_child;
93         end
94         prob(s) = P_index * temp;
95     end
96     prob = prob./sum(prob);
97     X(1,index) = sample_k(prob);
98 end
99
100 % This sub-function is to compute the probability of the node 'index'
    given its Markov Blanket.
101 %input:
102     % variable_names — The conditional probability;
103     % domain_counts — all the number of states of nodes;
104     % index — The index of the node
105     % dag — The relations of the parents of nodes
106     % X — The current states of all nodes

```

```

107 % output:
108 % P_index — Conditional probability of node index
109 function [ P_index ] = P_node( variable_names , domain_counts , index , dag ,
    X )
110 CPT = variable_names{1,index};
111 Pa_num = sum(dag(:,index));
112 if Pa_num == 0
113     prob = CPT;
114 elseif Pa_num == 1
115     Pa = find(dag(:, index)==1);
116     Pa_state = X(Pa);
117     prob = CPT(:, Pa_state);
118 elseif Pa_num == 2
119     all_pa = find(dag(:,index)==1);
120     Pa1 = all_pa(1); Pa2 = all_pa(2);
121     Pa_state = (X(Pa1)-1)*(domain_counts(Pa2) ) + X(Pa2);
122     prob = CPT(:, Pa_state);
123 end
124 P_index = prob(X(index));
125
126 end
127
128
129 % This sub-function is to generate one new state of one node given the
    probability 'prob'
130 function x = sample_k(prob)
131     u = rand;
132     prob0 = [0; prob];
133     for i = 1:numel(prob0)
134         if sum(prob0(1:i) ) > u
135             x = i-1 ;
136             break
137         end
138     end
139 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

```

1 % mean field method
2 function [ Inference , MAP] = mean_field(path_parameters , path_structure
    )
3 % input:
4 % path_parameters — the path of the parameter.mat file;
5 % path_structure — the path of the structure.mat file;

```

```

6 %output:
7 %Inference — The posterior inference p(Birth Asphyxia=yes |
    CO2report :`` < 7.5'', LVHReport:yes, %X-rayReport:Plethoric)
8 % MAP — The MAP inference of Disease
9 % load datasets
10 load(path_parameters);
11 load(path_structure);
12 %% parameters
13 thres = 1e-7;
14 iter_num = 10000000;
15 total_num = 20;
16 unknown_num = 17;
17 unknown_ind = [1:14, 16:17, 20];
18 variable_names = {BirthAsphyxia, Disease, Age, LVH, DuctFlow,
    CardiacMixing, LungParench,...
19    LungFlow, Sick, HypDistrib, HypoxiaInO2, CO2, ChestXray,
    Grunting, LVHreport, LowerBodyO2,...
20    RUQO2, CO2Report, XrayReport, GruntingReport };
21 %% initialize
22 beta = zeros(total_num, 6);
23 err = zeros(total_num, 6);
24 all_results = [];
25 for k = 1: total_num
26     for i = 1:domain_counts(k)
27         beta(k,i) = rand;
28     end
29     beta(k,:) = beta(k,:) ./ sum(beta(k,:));
30 end
31 all_results = zeros(1,iter_num);
32 for iter =1:iter_num
33     beta_old = beta;
34     for i = 1:total_num %update the ith node with the state of
        s
35         Eqi = zeros(1,domain_counts(i));
36         for s = 1:domain_counts(i)
37             [ Eqi(s) ] = Eq_i(total_num, dag, i, s,
                domain_counts, beta, variable_names );
38         end
39         for k = 1: domain_counts(i)
40             beta(i,k) = exp(Eqi(k))/sum(exp(Eqi)) ;
41             err(i,k) = norm(beta(i,k) - beta_old(i,k));
42         end
43     end
44     if min(min(err)) < thres
45         break
46     end
47     all_results(iter) = beta(1,1) ;
48     if mod(iter, 1) ==1
49         fprintf('The result is %.4f \n', beta(1,1));
50 end

```

```

51         end
52         Pl_inf = beta(1,1)
53         MAP_inference = find(beta(2,:) == max(beta(2,:)))
54     end
55
56 % This sub-function is to compute the expectation  $E_q(X_{-i})(p(X, x_i =$ 
57 %  $s, E))$ 
58 function [ results ] = Eq_i(total_num, dag, index, state, domain_counts,
59 % input:
60 % i — the node i;
61 % s — the state of node i;
62 % output:
63 % Eq —  $E_{\{q(X_{-i})\}}(-\log(p(X_{-i}, x_i, E)))$ 
64 results = 0;
65 for m = 1: total_num
66     all_pa = find(dag(:,m)==1);
67     all_node = [all_pa; m];
68     [all_x] = list_all_x(index, state, all_node, domain_counts, [], []);
69     Eq = 0;
70     for i = 1: size(all_x,1)
71         X = all_x(i,:);
72         %% compute q
73         q_no_i = 1;
74         for n = 1: numel(all_node)
75             if all_node(n) ~= index && X(n) < domain_counts(
76                 all_node(n))
77                 q_no_i = q_no_i * beta(all_node(n), X(n));
78             elseif all_node(n) ~= index && X(n) == domain_counts(
79                 all_node(n))
80                 K = domain_counts(all_node(n))-1;
81                 q_no_i = q_no_i * (1-sum(beta(all_node(n), 1:K)));
82             end
83         end
84         %% Pm
85         CPT = variable_names{1,m};
86         Pa_num = sum(dag(:,m));
87         if Pa_num == 0
88             prob = CPT;
89         elseif Pa_num == 1
90             Pa_state = X(1);
91             prob = CPT(:, Pa_state);
92         elseif Pa_num == 2
93             Pa_state = (X(1)-1)*(domain_counts(all_pa(2)))+X(2);
94             ;
95             prob = CPT(:, Pa_state);
96         end
97         Pm = prob(X(numel(X)));
98         if Pm == 0

```

```

95         break
96     end
97     Eq = Eq + log(Pm) * q_no_i;
98 end
99     results = results + Eq;
100 end
101 end
102
103 % This sub-function is to compute the possible x values
104 %input:
105     % index — The node i
106     % state — The state of node i
107     % all_pa — all the nodes that are considered;
108     % domain_counts — The number of states
109     % sample — The one sample
110     % all_x — The collection of samples
111 % output:
112     % all_x — all possible results
113 function [all_x] = list_all_x( index, state, all_pa, domain_counts, sample
    , all_x )
114     if size(sample,2) == numel(all_pa)
115         all_x = [all_x; sample] ;
116         return
117     end
118     domain_counts(index) = 1;
119     domain_counts(15) = 1;
120     domain_counts(18) = 1;
121     domain_counts(19) = 1;
122     for j=1:domain_counts(all_pa(size(sample,2)+1))
123         if all_pa(size(sample,2)+1) == 15 || all_pa(size(sample,2)
            +1) == 18
124             sample = [sample 1];
125         elseif all_pa(size(sample,2)+1) == 19
126             sample = [sample 3];
127         elseif all_pa(size(sample,2)+1) == index
128             sample = [sample state];
129         else
130             sample = [sample j];
131         end
132         if size(sample,2) <= numel(all_pa)
133             [all_x] = list_all_x(index, state, all_pa, domain_counts,
                sample, all_x );
134             sample(end) = [];
135         end
136     end
137 end

```