

國立交通大學

電控工程研究所

碩士論文

大型貝氏網路推論之  
時間與準度權衡演算法

A Tractable Time-Precision Tradeoff Algorithm for  
Inference in Large-Scale Bayesian Networks

研究生：莊仲翔

指導教授：周志成 博士

中華民國九十九年七月

大型貝氏網路推論之  
時間與準度權衡演算法

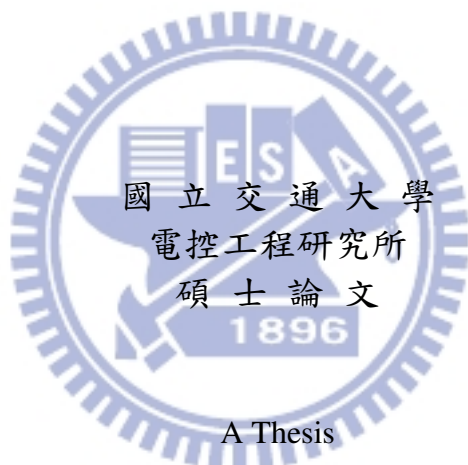
A Tractable Time-Precision Tradeoff Algorithm for  
Inference in Large-Scale Bayesian Networks

研究生：莊仲翔

Student : Chung- Hsiang Chuang

指導教授：周志成

Advisor : Chi-Cheng Jou



Submitted to Department of Electrical and Control Engineering

College of Electrical Engineering

National Chiao Tung University

in Partial Fulfillment of the Requirements

for the Degree of

Master

in

Electrical and Control Engineering

July 2010

Hsinchu, Taiwan, Republic of China

中華民國九十九年七月

# 大型貝氏網路推論之 時間與準度權衡演算法

學生：莊仲翔

指導教授：周志成

國立交通大學電控工程研究所

## 摘 要

在不同的領域中，貝氏網路是一種功能強大的分析工具。推論引擎則是貝氏網路中最重要的部分，負責處理接收到的訊息藉由機率的方式給出結論。推論引擎可以藉由不同的演算法來實現，並且加速推論的效率。由於傳統的精確推論演算法在大型貝氏網路中，會大幅降低演算效率導致無法運作。因此，我們提出了一種新的演算法，稱作 KLA 演算法，來解決在大型複雜的貝氏網路中推論的問題。KLA 演算法可透過權衡時間和精準度來提升推論的效率，而且所需要的記憶體空間會是所有推論演算法中最小的。因此，此演算法能夠更容易的將貝氏網路實現在記憶體受限制的應用中。為了評估在不同結構下 KLA 演算法的表現，我們設計了一系列的實驗來觀察準度和計算時間的結果並且和傳統聯結樹演算法來做比較。我們也將 KLA 演算法運用在現實中的案例上來觀察模擬的成果。


# A Tractable Time-Precision Tradeoff Algorithm for Inference in Large-Scale Bayesian Networks

Student : Chung- Hsiang Chuang

Advisors : Dr. Chi-Cheng Jou

Department of Electrical and Control Engineering  
National Chiao Tung University

## ABSTRACT



Bayesian networks (BNs) are powerful tools in diverse fields. The most important part is the inference engine, which draws conclusions by updating probabilities on the basis of the given knowledge. The inference engine can be implemented by using different algorithms, which help BNs draw various conclusions efficiently. Since traditional exact methods collapse when applied to large-scale methods, we propose a new algorithm, called KLA-algorithm, to solve the problem of drawing an inference in a large and complex system. The KLA-algorithm always has tractable computational time and a trade-off with precision. The required memory in KLA-algorithm is the minimum as compared to the other inference algorithms. This advantage extends large-scale BNs to some limited resource applications. In order to verify the KLA-algorithm in a different graph structure, we design a series experiment to compare with the junction tree algorithm and discuss the performance of precision and computation time. We also apply the KLA-algorithm to real-world data in order to carry out some simulations.

## 誌 謝

研究這條路，一路走來一直都是跌跌撞撞的，處在迷霧中永遠比撥雲見日的時間還要永久許多。能夠毫無畏懼的一直走到最後是因為有大家一直在旁邊的陪伴。

謝謝我的指道教授可以這麼放任的讓我為所欲為，讓我可以做我自己的研究，朝我想走的方向走；即使由於我的固執而走入死路，也會即時指引出另一條可行的道路。然而跟您相處兩年的點點滴滴，您的行事風格，特立獨行的品味對我的影響更甚於課業上的教導。

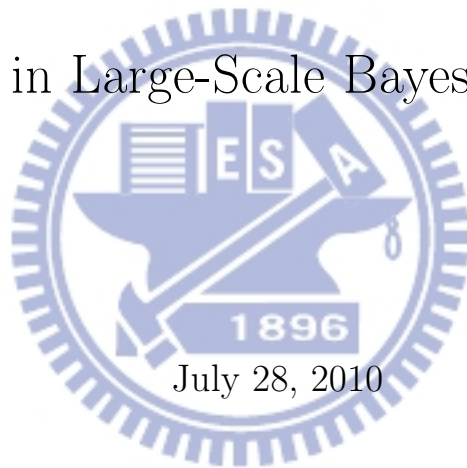
謝謝實驗室的研究同伴們，在苦悶時總是可以一起聊天、吃飯，這一直是我堅持到最後的動力。雖然有時覺得實驗室很吵，但是當實驗室空無一人時，才深深體會到那份吵鬧的珍貴。

在我身邊的朋友們，感謝你們的陪伴，大家研究的路都各不相同，但目標都是一致的。彼此的扶持與鼓勵讓我覺得這條旅途並不孤獨。

感謝口試委員對論文方面的建議以及提點，與你們討論令我受益良多也讓這份作品能更加完整。

最後，謝謝我的家人，總是受到你們的呵護與照顧。也在此將這份論文獻給你們。

A Tractable Time-Precision Tradeoff Algorithm for  
Inference in Large-Scale Bayesian Networks



July 28, 2010

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Motivation . . . . .	1
1.2	Organization . . . . .	4
<b>2</b>	<b>Large Bayesian Network Structure Learning</b>	<b>5</b>
2.1	Introduction . . . . .	5
2.2	Basic Concept of Bayesian Networks . . . . .	7
2.2.1	Independence and Conditional Independence . . . . .	7
2.2.2	BN Factorization . . . . .	7
2.2.3	D-Separation . . . . .	9
2.3	BN Power Constructor . . . . .	10
2.4	Issues Related to BN Power Constructor . . . . .	13
2.4.1	Threshold Selection . . . . .	13
2.4.2	Orientation Problem . . . . .	14
<b>3</b>	<b>Bayesian Network Inference</b>	<b>15</b>
3.1	Introduction . . . . .	15
3.2	Parameter Learning . . . . .	16
3.2.1	Bayesian Networks Model Parameterization . . . . .	17
3.2.2	Maximum Likelihood with Complete Data . . . . .	18
3.3	Inference engine . . . . .	20

3.3.1	Junction Tree Algorithm . . . . .	21
3.3.2	Conditioning Algorithm . . . . .	25
3.4	Why New Algorithm? . . . . .	29
3.4.1	Exact Algorithm Problem . . . . .	29
3.4.2	Some Approaches for Large-Scale Inference . . . . .	30
3.4.3	Why New Algorithm? . . . . .	34
<b>4</b>	<b>New Inference Approach</b>	<b>35</b>
4.1	Introduction . . . . .	35
4.2	KLA-Algorithm . . . . .	36
4.2.1	Structure Construction . . . . .	36
4.2.2	Initializing Potentials . . . . .	38
4.2.3	Propagation . . . . .	38
4.2.4	Joint Probabilities . . . . .	39
4.2.5	Conditional Probabilities . . . . .	41
4.3	Inference in Large System . . . . .	43
4.3.1	Complexity of Graph vs. Computing Time . . . . .	44
4.3.2	Good Approximation Method . . . . .	45
4.4	Complexity of KLA-Algorithm . . . . .	50
<b>5</b>	<b>Experiments</b>	<b>53</b>
5.1	Verification of KLA-Algorithm . . . . .	53
5.2	Application to Ozone Level Detection Data Set . . . . .	64
5.2.1	Problem and Data Introduction . . . . .	64
5.2.2	Bayesian network construction . . . . .	65
5.2.3	Inference simulation . . . . .	71
<b>6</b>	<b>Conclusions</b>	<b>75</b>



# List of Figures

1.1	Bayesian network modeling a meadow. The structure (nodes and arcs) and parameters (conditional probabilities table) are shown in the graph. . . . .	2
2.1	The joint probability $P(A, B, C, D, E)$ can be factorized as the product of all conditional probabilities, $\prod P(X \Pi_X)$ . . . . .	8
2.2	(a) Original graph of a multi-connected network. (b) Result draft of original data output from Phase I and input to Phase II. (c) Result graph after Phase II. The remaining edges are passed from the heuristic (in)dependent test. (d) Result graph output from Phase III. The remaining edges are passed from the exact (in)dependent test, and part of them will be oriented. The result structure is similar to the original structure. . . . .	12
3.1	Bayesian network model, structure, and CPTs; here, we parameterize all the entries in CPT. . . . .	18
3.2	(a) Original Bayesian network. (b) Corresponding moral graph. The newly added arcs are shown with dashed lines in the moralized graph. (c) Corresponding triangulated graph with two added chords (dashed). (d) Junction tree structure with corresponding cliques and separators. During the message propagation, messages are passed away from clique $ACE$ , beginning with $ACE$ ; these message passing route is indicated by the arrows. The numbers indicate a possible message passing order. . . . .	22

3.3	Initialization of clique $ACE$ by multiplying the CPT of node $C$ and node $E$ . The separator $CE$ is initialized to be unity. . . . .	24
3.4	Part of poly-tree network for each node $X$ with parents $U_1, \dots, U_M$ and children $Y_1, \dots, Y_N$ . . . . .	26
3.5	(a) $G$ is a moral graph of the Bayesian network and is a union graph in (b). (b) $G$ is sectioned into four graphs. (c) Junction forest of $G$ , each square represents a sub-graph in (b) and is called an agent. . . . .	32
4.1	(a) Simple Bayesian network with a loop. (b) Clique graph from (a); the parents of node $X_i$ and $X_i$ can be contained in corresponding $C_i$ . (c) Initial clique potential $\varphi(C_i)$ and separator potential $\phi(S_j)$ . . . . .	37
4.2	Message propagation in the nodes. (a) Update $\varphi(AEB)$ and $\varphi(AC)$ . (b) Since there is a loop contain node $D$ and $D$ is the bottom node, node $A$ will be instantiated to 0 and 1. The correct potential of $\varphi(BCD)$ will be obtained by $\varphi_1(BCD) \times P(A = 0) + \varphi_2(BCD) \times P(A = 1)$ . (c) Top node and bottom node in a loop. We select top node $A$ in the local loop cut-set of node $D$ . . . . .	40
4.3	(a) Addition of a virtual node $H$ , and assigning the interesting set of nodes to be its parents. (b) Corresponding clique-node $EDH$ . The blue line indicates the message propagation path, and $\varphi(EDH)$ can be obtained. . . . .	41
4.4	Propagation path of loopy belief propagation. Blue line represents the backward propagation path. After clique-nodes $B, C, D$ and $E$ have propagated the message to clique-node $A$ , return again until all of clique potentials are consistence. . . . .	43

4.5	(a) Maximum eigenvalue of a structure with ten nodes with different numbers of edges. A greater number of edges will lead to more loops in the structure and a large maximum eigenvalue. The maximum eigenvalue of all the ten nodes is between that of the line structure (the left) and the that of the fully connected structure (the right). (b) Maximum eigenvalue of nine edges with different nodes. A greater number of nodes will lead to simple structures and the maximum eigenvalue will decrease. . . . .	46
4.6	Computation time increase exponentially with an increase in the graph complexity. The value on the y-axis is the natural logarithm. . . . .	47
4.7	(a) Distance level from parents of bottom node $X_4$ to top node $X_1$ of the loop. (b) KL-divergence between real joint probability and estimated joint probability of the parents of node $X_4$ in the loop. The estimated joint probability assumes that the nodes $X_2$ and $X_3$ are independent of each other. The low value KL-divergence implies that the parents of node $X_4$ are closer to the independent node. The blue line indicates a loop structure that does not have an outside node, see (a). The green line indicates an outside node added to parent $X_3$ (shown in (c)) and has less KL-divergence than the blue line. (c) Addition of an outside node on the left path. . . . .	49
4.8	Yellow nodes $\{X_1, X_2, X_3\}$ are the local loop cut-set of node $X_{12}$ . $X_1$ is three levels from $X_{12}$ , and $X_2$ and $X_3$ are two levels from $X_{12}$ . We can just keep two levels for approximation, and the reduced local loop cut-set has only two nodes, $\{X_2, X_3\}$ . . . . .	50
4.9	(a) Poly-tree structure. (b) Multiply networks with few loops. (c) Multiply networks with many long loops. (d) Multiply networks with many short loops.	52
5.1	Number of nodes vs. graph complexity. . . . .	54

5.2	Graph complexity vs. maximum clique size. Size of the clique is represented by the number of nodes in the clique. The maximum clique size grows exponentially in junction tree algorithm but fixed in the KLA-algorithm when the graph complexity increases. . . . .	55
5.3	Computation time (seconds) of VN-method with different numbers of level approximations and junction tree algorithm. The computation time is represented in a logarithmic form. Junction tree algorithm lacks one point because the algorithm can not work in the most complex structure (The memory is not sufficient). The VN-method with all levels only has 4 points because the computation time is intractable in the last three structures. . . . .	57
5.4	K-L divergence between approximate value and exact value. There is no clear relation between KL-divergence value and graph complexity. Most K-L divergence values are less than $10^{-2}$ , which means that we can obtain a good approximation by the VN-method of the KLA-algorithm. . . . .	58
5.5	Computation time (seconds) of FB-method with different numbers of level approximations and junction tree algorithm. The computation time is represented in a logarithmic form. The FB-method when all levels are kept is replaced by that when seven levels are kept here, since the computation time of keeping all values is always intractable. . . . .	60
5.6	K-L divergence between approximate value and exact value in FB-method. There is no clear relation between KL-divergence value and graph complexity, and the K-L divergence in different level approximations are similar. All of the K-L divergence value are less than $10^{-2}$ , which means that we can obtain a good approximation by FB-method of KLA-algorithm. . . . .	62

5.7	Computational time of different number of evidence nodes in two different methods. The FB-method is presented as diamond with thick line, and the VN-method is circle with thin line. The FB-method has more stable computing time than VN-method. The y-axis represents the logarithm. . . . .	63
5.8	Maximum clique of JT and KLA algorithm in different value of threshold $\varepsilon_2$ . (a) In 2-bins discretization situation. (b) In 3-bins discretization situation. Both two situation indicate the JT-algorithm need more large space than KLA-algorithm. . . . .	68
5.9	BIC value of the model with different threshold and bins. The blue line is two bins discretization and the green line is three. The two bins discretization is always better than three bins. The best network is occur at $\varepsilon_2 = 0.04$ , with two bins discretization. . . . .	69
5.10	BIC value of the model with different threshold $\varepsilon_1$ . The $\varepsilon_1$ value smaller than 0.0016 would have the best structure. . . . .	70
5.11	Final networks of ozone detection problem. Each circle corresponds a attributes according to the number. The target node is No.73 and only be connected by Node12 which is wind speed resultant at 11 am. . . . .	71
5.12	The corresponding precision with different value of $v_E$ . The highest precision is 0.76 and $v_E$ is between 0.04 and 0.16. . . . .	72
5.13	Probability of ozone day on random six days from test-data. The left column is the normal day case and right column is the ozone day case. The dashed line is the threshold $v_E$ . . . . .	74

# List of Tables

3.1	X-ray noisy-OR example. The chance that an X-ray (E) will fail to detect two medical conditions $X$ and $Y$ is just the product of the individual failure chances.	33
5.1	Computation time (seconds) of VN-method with different numbers of level approximations and junction tree algorithm. The corresponding graph is shown in Figure 5.3. . . . . .	56
5.2	K-L divergence between approximate value and exact value. NaN represents that we do not have approximate value. The corresponding graph is shown in Figure 5.4. . . . . .	58
5.3	Computation time (seconds) of FB-method with different numbers of level approximations and junction tree algorithm. The corresponding graph is shown in Figure 5.5. . . . . .	60
5.4	K-L divergence between approximate value and exact value in FB-method. NaN represents that we do not have approximate value. The corresponding graph is shown in Figure 5.6. . . . . .	61
5.5	Computational time (sec) of different number of evidence nodes in two different methods of KLA-Algorithm. The corresponding graph is shown in Figure 5.7.	63
5.6	Seventy-two continuous attributes and one target variable in the data file. . . . . .	66
5.7	Precision and cross entropy error of different level approximation. All level is the exact result. . . . . .	73

# Chapter 1

## Introduction

### 1.1 Motivation

Bayesian networks are powerful tools in diverse fields such as medical diagnosis [24], image recognition [1], language comprehension [2], and search algorithms [12]. Bayesian networks are very effective in modeling situations where some information is already known and the incoming data are uncertain or partially unavailable (unlike rule-based or other expert systems, where uncertain or unavailable data lead to ineffective or inaccurate reasoning). These networks also provide consistent semantics for representing causes and effects through an intuitive graphical representation. Because of all of these capabilities, Bayesian networks are being increasingly used in a wide variety of domains where automated reasoning is needed.

The Bayesian network systems can be deconstructed into three components. First is the structure, which represents a set of random variables and their conditional independence. Second is the parameter, which uses a conditional probabilities table to quantify the relationship between variables. Third and the most important part is the inference engine, which draws conclusions by updating probabilities on the basis of the given knowledge. The previous two parts can be attained from domain knowledge or by learning from the data. The inference engine can be implemented by using different algorithms, which help Bayesian

networks draw various conclusions efficiently.

For example, Figure 1.1 shows the simple case for modeling a meadow. The structure, composed of arcs and nodes, indicates the relationship between cloudy weather, sprinkler, rain, and wet grass conditions. The wet grass condition is caused by the rain or the sprinkler condition, and the cloudy weather condition can affect the rain and decide whether the sprinkler should be turned on. The strength of a relationship depends on the conditional probabilities table; these probabilities are deduced on the basis of data or experience. Suppose we observe that the grass is wet in the morning, and wish to know whether it rained or whether the water sprinkler was turned on the previous night. By using the inference engine, the Bayesian network would update the probabilities on the basis of the given information (e.g. the grass is wet.).

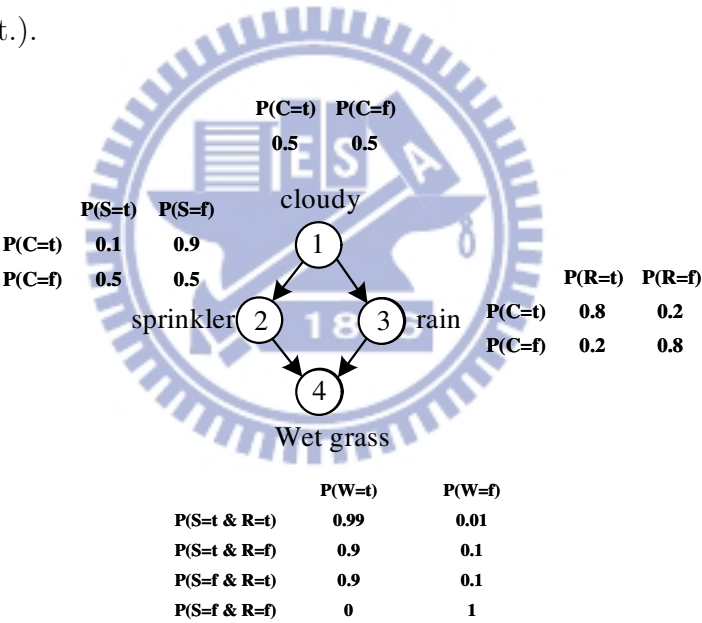


Figure 1.1: Bayesian network modeling a meadow. The structure (nodes and arcs) and parameters (conditional probabilities table) are shown in the graph.

To be considered an expert system, it is important for the system to draw efficient inference. In recent years, several efficient exact inference algorithms have been proposed, and successfully used in different network structures. However, when a system becomes complicated and the scale of the network increases, the inference engine requires a large number of computations or a huge amount of memory, and becomes inefficient. In order to solve



this problem, some people use approximate methods, and others keep improving the exact methods. The approximate methods have a trade-off between the precision and the computational time as well as the memory space. In large-scale networks, the precision is not sufficiently good enough for some applications. On the other hand, there has not been any significant improvement of the exact methods. The most famous method is the multiply sectioned Bayesian network, which divides a network into small sub-networks, and obtains the result by combining the inference in these sub-networks. However, even this method has not efficiently decreased the computation time and memory space required.

In the present exercise, we focus on developing a new algorithm to make the Bayesian networks draw inferences efficiently from complex, large-scale networks. There are three reasons why we pay attention to the problem:

1. A complex large-scale system is commonly used in a variety of fields such as industry, meteorology, and genetics. A Bayesian network is a powerful tool for predictive reasoning in these fields. Therefore, the above-mentioned problem is very practical in real life.
2. While modeling the large-scale problem, feature selection can be used for simplifying the problem. However, feature selection focuses on the correlation between an input variable and an output variable, but not on the input variables themselves. Therefore, we may lose some information related to the system. Large-scale Bayesian networks can store more information about a system.
3. The memory space is limited in some applications such as embedded system. Therefore, solving this problem is helpful in extending the applications of Bayesian networks to a large scale.

## 1.2 Organization

In Chapter 2, we first describe the Bayesian networks, and then present a construction algorithm for large-scale Bayesian networks. In large-scale networks, it is difficult to attain the structure from domain knowledge. Therefore, we briefly discuss the algorithms for obtaining a structure from the data and select the BNPC algorithm [5] to construct our model in Chapter 5.

Chapter 3 presents the parameter learning and inference engine of a Bayesian network. The maximum likelihood method is introduced for estimating the parameters of the model. Two exact inference engines - junction tree algorithm [14] and conditioning methods [27] - are described; these are helpful for the development of our new algorithm in Chapter 4. We also examine the other methods proposed to solve the large-scale networks at the end.

Chapter 4 presents a new algorithm devised by using combination of the two exact inference engines mentioned in Chapter 3. The approximate working of new algorithm is discussed with respect to obtaining a relatively fast inference. We discuss the complexity of this algorithm and compare this algorithm to other exact inference engines.

In Chapter 5, we first carry out some simulations in order to compare the performance of the new algorithm and the junction tree algorithm for different complexity values of the networks. Next, we apply the new algorithm to real-world data and discuss the result.

Chapter 6 summarizes the present exercise and outlines some future work.

# Chapter 2

## Large Bayesian Network Structure Learning

### 2.1 Introduction

A Bayesian network, also called a *belief network*, is a graphical structure that allows us to represent and reason an uncertain domain. Formally, Bayesian networks represent a set of random variables and their conditional independence via a *directed acyclic graph* (DAG). Each node in the graph is associated with a random variable, while the arcs, or directed edges, between the nodes indicate the probabilistic dependencies among the corresponding random variables. For example, a Bayesian network can represent the probabilistic relationships in medical diagnosis. Given the symptoms, the network can be used for computing the probabilities of the presence of various diseases.

The construction of a network structure for a given application by domain experts is a time-consuming task. It is difficult to build a medium-sized network manually. Therefore, one may learn the dependency structure directly from the data via computational methods. Over the last decade, considerable progress has been made with respect to structural learning in Bayesian networks. Two important classes of such algorithms have been defined as the

*score-metric-based* and the *constraint-based* learning methods [13, 19, 25]. The score-metric-based methods deduce structures by optimizing a scoring function, and the constraint-based methods infer structures through conditional independence tests. Since the number of possible network structures grows exponentially with the number of nodes, both methods apply heuristic search in a certain manner.

In a comparative study [17], some currently used structure learning algorithms are identified. One is a method based on a scoring criterion, such as K2 [6] (maximization of the structure probability for the given data), Greedy search [3] (finding the best score of the neighbor and iterate) or SEM [8] (greedy search dealing with missing values). The other is a method based on a constraint-based criterion, such as PC [19] or IC [22] (searching causality using statistical tests to evaluate conditional independence), or BN Power Constructor (BNPC) [5] (using conditional independence tests based on an information theory). However, the problem of learning an optimal Bayesian network from a given data-set is NP-hard [4]. In the present exercise, we have focused on a large, complex system. Therefore, we need a method that can achieve structure learning with limited memory space and tractable time. The best method is BN Power Constructor; however, we still need to modify this method for the construction of our model.

This chapter is organized as follows: Section 2.2 presents the basic concept of Bayesian networks. In Section 2.3, we describe the BNPC structure learning algorithm. Section 2.4 discusses some issues that arise when this algorithm is applied to the construction of a large variable network.

## 2.2 Basic Concept of Bayesian Networks

### 2.2.1 Independence and Conditional Independence

In probability theory, two random variables,  $X$  and  $Y$ , are said to be independent if

$$P(X = x, Y = y) = P(X = x)P(Y = y), \quad (2.1)$$

or equivalently, if

$$P(X = x|Y = y) = P(X = x), \quad (2.2)$$

where  $P(X = x|Y = y)$  is a conditional probability.

By combining the concept of independent and conditional probability, we can represent the conditional independence as

$$P(X = x, Y = y|Z = z) = P(X = x|Z = z)P(Y = y|Z = z), \quad (2.3)$$

or equivalently<sup>1</sup>,

$$P(X = x|Y = y, Z = z) = P(X = x|Z = z). \quad (2.4)$$

In other words, if we know about  $Z$ , then  $X$  and  $Y$  are independent or the knowledge of  $Y$  does not help us to guess  $X$  when  $Z$  is known. We say that  $X$  and  $Y$  are conditionally independent given  $Z$ , and we write this as  $X \perp Y | Z$ .

### 2.2.2 BN Factorization

Suppose that we have a directed acyclic graph (DAG)  $D$  with nodes  $X_i$ , where  $i = 1, 2, \dots, p$ . If there is an arc from node  $X_i$  to another node  $X_j$ ,  $X_i$  is called the parent of  $X_j$ , and  $X_j$  is a child of  $X_i$ . The set of parent nodes of a node  $X_i$  is denoted by  $\Pi_{X_i}$ . A directed acyclic graph is a Bayesian network if the joint distribution of the node values can be written as the

---

<sup>1</sup> $P(X = x, Y = y|Z = z) = P(X = x|Y = y, Z = z) \cdot P(Y = y|Z = z)$ .

product of the individual density functions, conditional on their parent variables:

$$P(X_1 = x_1, X_2 = x_2, \dots, X_p = x_p) = \prod_i P(X_i = x_i | \Pi_{X_i} = \Pi_{x_i}). \quad (2.5)$$

where  $\Pi_{x_i}$  is the corresponding value of the parent nodes of node  $X_i$ .

An example of BN factorization is shown in Figure 2.1. The joint probability  $P(A, B, C, D, E)$  is factorized as the product of five conditional probabilities as follows:  $P(A) \cdot P(B) \cdot P(C|B) \cdot P(D|A, B) \cdot P(D|E)$ .

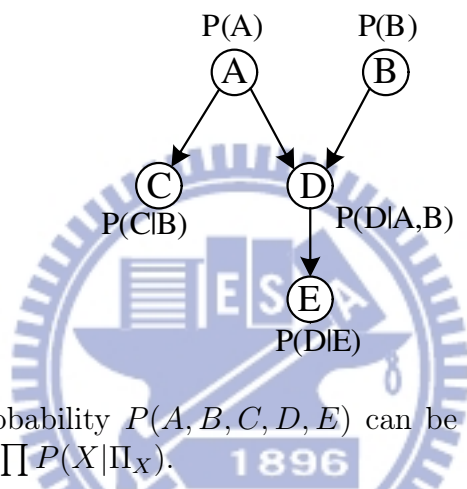


Figure 2.1: The joint probability  $P(A, B, C, D, E)$  can be factorized as the product of all conditional probabilities,  $\prod P(X|\Pi_X)$ .

We order all of variables  $X$ , such that for any two nodes  $X_i$  and  $X_j$ , when  $i < j$ , there is a directed path from  $X_i$  to  $X_j$ . When  $i < j$ ,  $X_i$  is called an *ancestor* of  $X_j$ . The set of all ancestors of  $X_j$  is denoted by  $an(X_j)$ . On the other hand,  $X_j$  is called a *descendant* of  $X_i$ . The set of all descendants of  $X_i$  is denoted by  $de(X_i)$ . The BN factorization implies the *local Markov property*: each variable is conditionally independent of its ancestors given its parent variables,

$$X_i \perp an(X_i) | \Pi_{X_i}. \quad (2.6)$$

Recall that the joint distribution can also be calculated from the conditional probabilities using the chain rule as follows:

$$P(X_1 = x_1, X_2 = x_2, \dots, X_p = x_p) = \prod_i P(X_i = x_i | X_{i-1} = x_{i-1}, \dots, X_1 = x_1). \quad (2.7)$$

By comparing (2.5) and (2.7), we can obtain the following formula:

$$P(X_i = x_i | \Pi_{X_i} = \Pi_{x_i}) = P(X_i = x_i | X_{i-1} = x_{i-1}, \dots, X_1 = x_1). \quad (2.8)$$

Because the graph is acyclic, the set of parents is a subset of the set of ancestors, thereby implying the local Markov property.

### 2.2.3 D-Separation

*D-separation* provides considerably strong criterion for independence and plays an important role in Bayesian network learning and inference. A path  $p$  is said to be d-separated (or blocked) by a set of nodes  $Z$  if and only if (at least) one of the following holds:

1.  $p$  contains a chain,  $i \rightarrow m \rightarrow j$  or  $i \leftarrow m \leftarrow j$ , such that the middle node  $m$  is in  $Z$ .
2.  $p$  contains a fork,  $i \leftarrow m \rightarrow j$ , such that the middle node  $m$  is in  $Z$ .
3.  $p$  contains an inverted fork (or *collider*),  $i \rightarrow m \leftarrow j$ , such that the middle node  $m$  is **not** in  $Z$  and no descendant of  $m$  is in  $Z$ .

If every undirected path from a node in  $X_i$  to a node in  $X_j$  is d-separated by  $Z$ , then  $X_i$  and  $X_j$  are conditionally independent given  $Z$ , in formula form  $X_i \perp X_j | Z$ .

For example, in Figure 2.1 we illustrate the following conditional independent relationship using d-separation.

- $(C \perp B, D, E | A)$ :  $C \leftarrow A \rightarrow D$  is a fork path that is closed since we condition on  $A$ .

- $(B \perp A, C|\phi)^2$ :  $B \rightarrow D \leftarrow A$  is a closed inverted fork path since we do not condition on  $D$  or it's descendant  $E$ .
- $(B \perp E|D)$ :  $B \rightarrow D \rightarrow E$  is a closed chain path since we condition on  $D$ .

## 2.3 BN Power Constructor

The BNPC algorithm was proposed in [5] . The algorithm constructs Bayesian networks by analyzing conditional independence relationships among nodes on the basis of information theory. In information theory, the *mutual information* (MI) of two random variables is a quantity that measures the mutual dependence of the two variables, defined as

$$I(X_i, X_j) = \sum_{x_i, x_j} P(x_i, x_j) \log \frac{P(x_i, x_j)}{P(x_i)p(x_j)}; \quad (2.9)$$

and *conditional mutual information* (CMI) is the expected value of the mutual information of two random variables given the value of a third, defined as

$$I(X_i, X_j|C) = \sum_{x_i, x_j, c} P(x_i, x_j, c) \log \frac{P(x_i, x_j|c)}{P(x_i|c)p(x_j|c)}. \quad (2.10)$$

In this algorithm, the conditional mutual information is used as conditional independent tests. When  $I(X_i, X_j|C)$  is smaller than a certain threshold value  $\varepsilon$ , we say that  $X_i$ , and  $X_j$  are d-separated by the condition-set  $C$ , and are conditionally independent.

The entire procedure of this algorithm can be decomposed into three phases as follows:

### **PhaseI:** (Drafting)

In the first phase, the algorithm computes the mutual information of each pair of nodes as a measure of closeness, and creates a draft on the basis of this information. We select the MI of the pairs of nodes that are greater than a threshold  $\varepsilon$ , and order them from large to

---

<sup>2</sup> $\phi$  denotes an empty set.



small. Then beginning from the large, we place an edge between the corresponding pair of nodes if there is no path between these nodes. After at most  $(n - 1)$  edges are drafted ( $n$  is the number of nodes in the graph), we obtain a draft of the structure. This phase ensures that the output draft is a single-connected structure. In fact, if the original graph is just a single-connected graph, the draft would be the same as the original one. The draft created in this phase is the base for the next phase.

We use the same simple multi-connected network example as that in [5]. Suppose we have a database of the original Bayesian networks, as shown in Figure 2.2(a). The task is to recover the original networks from the data. Suppose we have  $I(B, D) \geq I(C, E) \geq I(B, E) \geq I(A, B) \geq I(B, C) \geq I(C, D) \geq I(D, E) \geq I(A, D) \geq I(A, E) \geq I(A, C)$ , and all mutual information is greater than  $\epsilon$ , the draft obtained from Phase I is shown in Figure 2.2(b).

**PhaseII:** (Thickening)

In the second phase, the algorithm adds edges when the pairs of nodes are not conditionally independent of a certain conditioning set. We examine all pairs of nodes that have mutual information greater than  $\epsilon$  and are not directly connected. An edge is added only when two nodes are not independent given a certain condition-set. The condition-set can be found by using a heuristic method, and hence, it may not be able to find the correct condition-set for a special group of structures and has some edges that are wrongly added in this phase.

For example, the graph after Phase II is shown in 2.2(c). An edge such as  $[AC]$ ,  $[AD]$ ,  $[CD]$ , or  $[AE]$ , is not added because the CI test can reveal that these pairs of nodes are independent of the given corresponding condition-set.

**PhaseIII:** (Thinning)

In the final phase, each edge is examined using conditional independence tests and will be removed if the two nodes of the edge are conditionally independent. Since Phase II has some incorrectly added edges, we check the edges again in this phase. In order to guarantee a

correct structure, we use a correct procedure to find the exact condition-set. After obtaining the correct structure, we use the algorithm to orient some of the edges.

The correct procedure for finding the condition-set can replace the heuristic procedure in Phase II. However, in practice, the heuristic procedure usually uses relatively few conditional independent tests and requires a relatively small condition-set. Therefore, in order to speed up the algorithm, the correct procedure is used only in the final phase.

The graph after the completion of Phase III in our example is shown in Figure 2.2(d); this graph has the almost same structure as that of the original graph. Edge  $[BE]$  is removed since  $B$  and  $E$  are independent given the condition-set  $\{C, D\}$ . In this graph, we can only orient two edges  $[CE]$  and  $[DE]$ ; the reason for this will be described in Section 2.4.2.

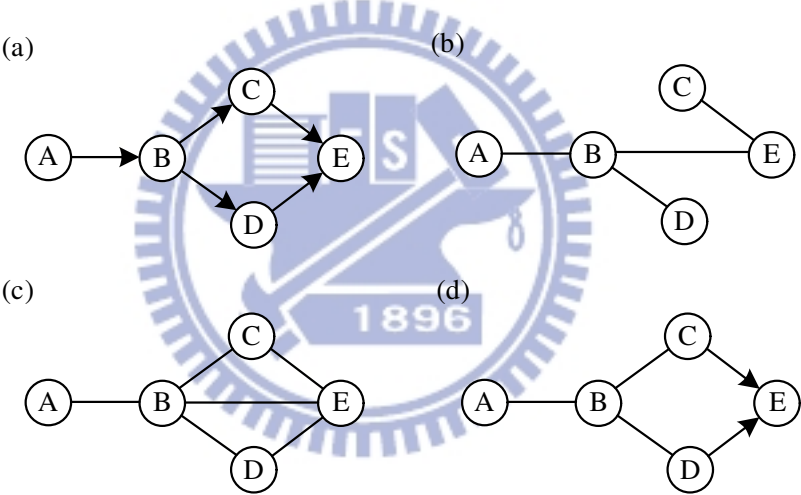


Figure 2.2: (a) Original graph of a multi-connected network. (b) Result draft of original data output from Phase I and input to Phase II. (c) Result graph after Phase II. The remaining edges are passed from the heuristic (in)dependent test. (d) Result graph output from Phase III. The remaining edges are passed from the exact (in)dependent test, and part of them will be oriented. The result structure is similar to the original structure.

## 2.4 Issues Related to BN Power Constructor

### 2.4.1 Threshold Selection

The threshold has an effect on two places in the algorithm. One is in phase I, when we choose the MI of the pairs of nodes which are greater than a threshold  $\varepsilon_1$ . The other is in Phase II and III, the procedure of testing the conditional independence of the pairs of nodes. If  $I(X_i, X_j|C)$  is smaller than a threshold value  $\varepsilon_2$ , then  $X_i$  and  $X_j$  are conditionally independent. When  $\varepsilon_1$  is large, the small edges would be considered; the small edges are set for the conditional independent test. However, the graph would be too simple to explain the data. On the other hand, if  $\varepsilon_1$  is small, many edges will be examined. We will be able to determine a relatively more correct structure, but waste more time in computation. For the threshold value  $\varepsilon_2$  of the conditional independent test, the large  $\varepsilon_2$  leads to the omission of a considerable number of edges, and the graph structure will be too simple. However, the small value of  $\varepsilon_2$  will lead to a complex graph, and cannot explain the data well.

In practice, the optimal threshold is unknown beforehand. Moreover, the optimal threshold is problem and data-driven, i.e., it depends, on one hand, on the database and its size and, on the other hand, on the variables and the numbers of their states. Therefore, it is not possible to set a default threshold value that will accurately determine conditional independence while using any database or problem. Therefore, we would score structures learned using different thresholds by a likelihood-based criterion evaluated using the training set and select the threshold leading to the structure achieving the highest score. The most commonly used scoring function is the Bayesian information criteria (BIC) which is derived on the basis of the principles stated in [23]. The BIC is a criterion for model selection among a class of parametric models with different numbers of parameters and has the following formulation:

$$BIC(G, D) = \log P(D|G, \theta^{ML}) - \frac{1}{2} \text{Dim}(G) \cdot \log N, \quad (2.11)$$

where  $D$  is the data-set,  $ML$  are the parameter values obtained by likelihood maximization, and the network dimension  $\text{Dim}(G)$  is used for preventing the graph from becoming very over-fitting. The other methods such as test error are also a good pointer.

## 2.4.2 Orientation Problem

In the last phase of the BNPC algorithm, we orient the edges in the final step. The procedure involves the identification of colliders, as the other edge orientations are virtually based on these identified colliders. Colliders can be found by a conditional independence test. For any two nodes  $X_i$  and  $X_j$  that are not directly adjoined and are dependent according to the smallest given condition-set  $C$ , by d-separation, we can say that nodes  $X_i$  and  $X_j$  are the parents of the node set  $C$ , i.e.,  $X_i \rightarrow C \leftarrow X_j$ .

However, in practice, we cannot always orient all the edges. In a special case, if a part of the structure is a line structure, which means that the nodes adjoin each other one by one, then there are no identifiable collides. Therefore, we cannot know the exact direction of the structure. For example, in Figure 2.2(d), we can only identify the collider  $E$ ;  $C$  and  $D$  are the parents of  $E$ . The other edges cannot be oriented since we do not have any further information.

Therefore, expert knowledge is required in this situation. In the worst case, there is no expert knowledge and we cannot orient all the edges. We will arbitrarily assign the variable order in this case, and the previous order will be the ancestors of the reverse order. Therefore, the directed graph cannot explain a causal relationship between two connected nodes, but just a conditional relationship between them.

# Chapter 3

## Bayesian Network Inference

### 3.1 Introduction

In the previous chapter, we described an algorithm for building a Bayesian network structure. In this chapter, our goal is to draw conclusions when new information, or evidence, is observed. For example, in the field of meteorology, the objective of a meteorological system is to forecast weather with some observed climate data (e.g., temperature, cloud, and humidity data). The mechanism of drawing conclusions in Bayesian networks is called an *inference engine*, or the *propagation of evidence*. The inference engine consists of updating the probability distributions of variables (e.g., weather forecast) according to the newly available evidence (e.g., climate data).

Two types of algorithms for an inference engine are available, namely, exact and approximate. The basic concept of an exact inference is based on the Bayesian theorem. In accordance with the variable dependency relationships residing in a network, we can use more efficient methods to update probabilities. In recent years, several efficient methods have been developed, such as variable elimination [29], junction tree, and belief propagation [20]. However, all the exact algorithms suffer from a combinatorial explosion when dealing with large and complex networks.

Approximate inference methods are used in large-scale systems when exact methods are computationally inefficient. The basic idea of an approximate inference is to generate a sample from the *joint probability distribution* of the variables, and then use the generated sample to compute approximate values for the probabilities of certain events given the evidence. Different approximate methods attempt to improve the quality or efficiency of approximations. The problem with an approximate inference is the requirement of an adequate sample size. In large systems, large amounts of data are required to ensure reasonable results.

Although approximate methods can handle large-scale systems, for obtaining a better inference result, exact methods are used. We have reason to believe that the exact methods can be improved to be efficient while dealing with large, complex systems. Therefore, in the present exercise, we will also focus on the exact inference algorithms in this chapter.

This chapter is organized as follows: Section 3.2 presents an approach for estimating the parameters of Bayesian networks. After both the structure and the parameters are provided, the inference engine can be launched. In Section 3.3, the exact inference algorithms used for Bayesian networks are considered. The exact inference algorithms that we consider include the *junction tree algorithm* and *belief propagation algorithm*. Both algorithms will be used for developing a new algorithm in the next chapter. Section 3.4 discusses the advantages and the shortcomings of both the algorithms and explains why we need to develop a new algorithm for Bayesian networks.

## 3.2 Parameter Learning

Before trying to launch the Bayesian network (BN) inference engine, we have to specify the BN completely. There are two major parts for learning a BN from data, one is the structure learning, and the other is the parameter learning. In Chapter 2, we described a structure learning method. Once the structure of a BN is determined, the problem left is to learn the parameters from the data.

We take a standard statistical modeling approach for parameter learning. The distribution of the data is unknown, but is assumed to belong to some given family of possible distributions. We label the distinct members of the family by the value of a set of parameters  $\boldsymbol{\theta} = (\theta_1, \dots, \theta_M)^T$ . Thus,  $\boldsymbol{\theta}$  determines the unknown probabilities, and our task is to use the data to estimate  $\boldsymbol{\theta}$ . There are many statistical methods for doing so. Here, we adopt the *maximum likelihood estimation* method. In the following paragraphs, we discuss how to parameterize the Bayesian networks by  $\boldsymbol{\theta}$  and use the maximum likelihood approach for estimating the optimal values of  $\boldsymbol{\theta}$ .

### 3.2.1 Bayesian Networks Model Parameterization

Suppose that we have a directed acyclic network (DAG)  $D$  with nodes  $X_i$ , where  $i = 1, 2, \dots, p$ . We use capital letters, such as  $X$ ,  $Y$ , and  $Z$ , as variable names and lowercase letters,  $x$ ,  $y$ , and  $z$ , to denote the specific values taken by these variables. Sets of variables are denoted by boldface capital letters  $\mathbf{X}$ ,  $\mathbf{Y}$ , and  $\mathbf{Z}$ , and the assignments of values to the variables in these sets are denoted by boldface lowercase letters,  $\mathbf{x}$ ,  $\mathbf{y}$ , and  $\mathbf{z}$ . Let  $\Pi_{X_i}$  denote the set of parent nodes of  $X_i$  in  $D$ . If  $X_i$  does not have a parent,  $\Pi_{X_i}$  is the empty set. A particular value in the joint distribution is represented by  $P(X_1 = x_1, X_2 = x_2, \dots, X_p = x_p)$ , or more compactly,  $P(x_1, x_2, \dots, x_p)$ . We can factorize (Section 2.2.2) the joint probabilities as:

$$P(x_1, x_2, \dots, x_p) = \prod_i P(x_i | \Pi_{x_i}). \quad (3.1)$$

Therefore, a Bayesian network can be parameterized using a vector of conditional probability table (CPT) entries, one entry for each value of each node and each instantiation of the parents of the nodes. We now define the overall parameter  $\boldsymbol{\theta}$ , which is composed by parameter  $\theta_{x_i | \Pi_{x_i}}$  to represent the conditional probability table entry  $P(x_i | \Pi_{x_i})$  for each possible value  $x_i$  of  $X_i$ , and  $\Pi_{x_i}$  of  $\Pi_{X_i}$ .

Now, (2.7) becomes

$$P(x_1, x_2, \dots, x_p | \theta) = \prod_i P(x_i | \Pi_{x_i}, \theta_{x_i | \Pi_{x_i}}). \quad (3.2)$$

For example, in graph 3.1, we show a simple Bayesian network with structure and four conditional probability tables (CPTs) for each node.  $\theta_{x_i | \Pi_{x_i}}$  represents each conditional probability in CPT, such as  $\theta_{W=T|S=T,R=F} = P(Wet(W) = T | Sprinkler(S) = T \& Rain(R) = F) = 0.9$ .

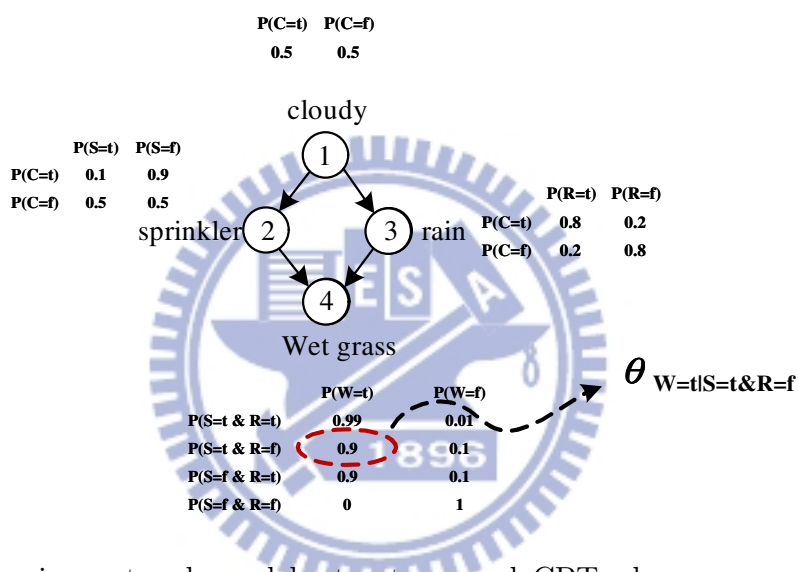


Figure 3.1: Bayesian network model, structure, and CPTs; here, we parameterize all the entries in CPT.

### 3.2.2 Maximum Likelihood with Complete Data

Maximum likelihood estimation (MLE) is a popular statistical method used for fitting a statistical model to data, and providing estimates for the parameters of the model (i.e., the probability of the observed data as a function of the unknown parameters). This procedure involves the calculation of the likelihood on the basis of the data according to the model, and finding the values of the parameters that maximize this likelihood. For complete data with no constraints related to the component parameters, the calculation of the maximum likelihood is divided into a collection of local calculations, as we shall see.



For discrete Bayesian networks, suppose that we have a sample of  $N$  independent and identically distributed cases  $\mathbf{d} = \{\mathbf{x}^{(1)}, \dots, \mathbf{x}^{(N)}\}$ , where  $\mathbf{x}^{(n)} = (x_1^{(n)}, \dots, x_p^{(n)})$ . The *likelihood* is a function of the parameters that is proportional to the probability of the observed data:

$$P(\mathbf{d}|\boldsymbol{\theta}) = \prod_{n=1}^N P(\mathbf{x}^{(n)}|\boldsymbol{\theta}). \quad (3.3)$$

We assume that the parameters are unknown and estimate them from data. We focus on the problem of estimating a single setting of the parameters that maximizes the likelihood (3.3). Equivalently, we can maximize the log likelihood:

$$\begin{aligned} \mathcal{L}(\boldsymbol{\theta}) &= \log P(\mathbf{d}|\boldsymbol{\theta}) = \sum_{n=1}^N \log p(\mathbf{x}^{(n)}|\boldsymbol{\theta}) \\ &= \sum_{n=1}^N \sum_{i=1}^p \log p(x_i^{(n)}|\Pi_{x_i^{(n)}}, \theta_{x_i^{(n)}|\Pi_{x_i^{(n)}}}) \end{aligned} \quad (3.4)$$

where the last equality makes use of the factorization (2.7). Let  $N_{\mathbf{X}}(\mathbf{x})$  be the number of occurrences in the data set where  $\mathbf{X} = \mathbf{x}$ . (3.4) can be rewritten as:

$$\mathcal{L}(\theta_{x_i|\Pi_{x_i}}) = \sum_i \sum_{x_i, \Pi_{x_i}} N(x_i, \Pi_{x_i}) \log(\theta_{x_i|\Pi_{x_i}}) \quad (3.5)$$

The problem of maximizing the data log-likelihood subject to the parameters can be restated as:

$$\max \quad \mathcal{L}(\theta_{x_i|\Pi_{x_i}}) = \sum_i \sum_{x_i, \Pi_{x_i}} N(x_i, \Pi_{x_i}) \log(\theta_{x_i|\Pi_{x_i}})$$

$$\text{subject to} \quad \sum_{x_i} \theta_{x_i|\Pi_{x_i}} = 1.$$

This is a simple optimal problem. We just need to solve the following equation:

$$\frac{\partial \mathcal{L}}{\partial \theta_{x_i|\Pi_{x_i}}} = \frac{N(x_i, \Pi_{x_i})}{\theta_{x_i|\Pi_{x_i}}} + \lambda = 0, \quad (3.6)$$

where  $\lambda$  is the Lagrange multiplier. After computation, we obtain  $\theta_{x_i|\Pi_{x_i}} = -N(x_i, \Pi_{x_i})/\lambda$ . Since  $\sum_{x_i} \theta_{x_i|\Pi_{x_i}} = 1$ , the solution of  $\lambda$  is  $-N(\Pi_{x_i})$ , where  $N(\Pi_{x_i}) = \sum_{x_i} N(x_i, \Pi_{x_i})$ . Therefore ,

$$\theta_{x_i|\Pi_{x_i}} = \frac{N(x_i, \Pi_{x_i})}{N(\Pi_{x_i})} \quad (3.7)$$

In other words, on the basis of a database of cases, for all nodes  $X_i$ , the conditional probabilities are estimated by using the ratio of the corresponding counts.

There are other issues in parameter learning, such as dealing with incomplete data, and updating the parameter on-line. Many approaches to obtain better parameters on different situations have been reported. However, in this present exercise, we just pay attention to the complete data parameter learning.

### 3.3 Inference engine

The inference engine is the core of the Bayesian networks. Only by an inference engine, a BN can work as an expert system to predict, diagnose, or detect fraud. We can think of the inference engine as a human brain, which uses the past experience (i.e., CPT) to answer questions in new situations (i.e., given evidence). Like a human brain, an efficient and general inference engine leads to versatile applications of a BN.

With respect to loops, there are two different types of directed graphs, namely, single-connected networks and multiple-connected networks. The single-connected network, also called a poly-tree, is a directed graph without loops, or for any two nodes in the graph, there is only one path between them. It is straightforward to develop propagation algorithms on poly-trees, but poly-trees are not suitable in many real-world problems due to its oversimplified structures. The multiple-connected network is a directed graph containing loops and is thus adequate for real-world situations. In order to propagate information in a multiple-connected network, the commonly used approach is to transform the multiple-connected structure into a single-connected network.

Both *conditioning* and *clustering* inference methods are widely used in *single-connected* and *multiple-connected* networks. Conditioning methods involve the breaking of the communication pathways along the loops by instantiating a select group of variables. This results in a single-connected network in which poly-tree propagation algorithms can be applied. For either a single-connected or a multiple-connected network, clustering methods build an associated graph in which each node corresponds to a set of variables. This leads to a network with poly-trees.

In the following paragraphs, we discuss two algorithms: conditioning algorithm (conditioning method) and junction tree algorithm (clustering method).

### 3.3.1 Junction Tree Algorithm

The idea of the junction tree algorithm is to aggregate nodes into a set of nodes, called *cliques*, which result in a local structure called a *junction tree* devised for propagating the evidence in the network. The algorithm has five steps, namely, moralization, triangulation, constructing the junction tree, updating the potentials, and propagation.

The junction tree algorithm can be described as follows:

#### **Moralization:**

For a directed acyclic graph, the corresponding *moral graph* is formed by connecting the parents of each node, and then making all edges in the graph undirected. In a moral graph, nodes with a common child are said to be married and thus form a family.

In this step, the BN graph (directed acyclic graph) is transformed into an undirected graph. The moral graph  $G_m$  is obtained by linking the parents of each node and dropping the directions in the original BN graph  $D$ . The goal is to combine the family of nodes, since the family members contain of the conditional probability. An example of the moralization is shown in Figure 3.2(b).

#### **Triangulation:**

A graph is called *chordal* if each of its cycles of four or more nodes has a chord, which is

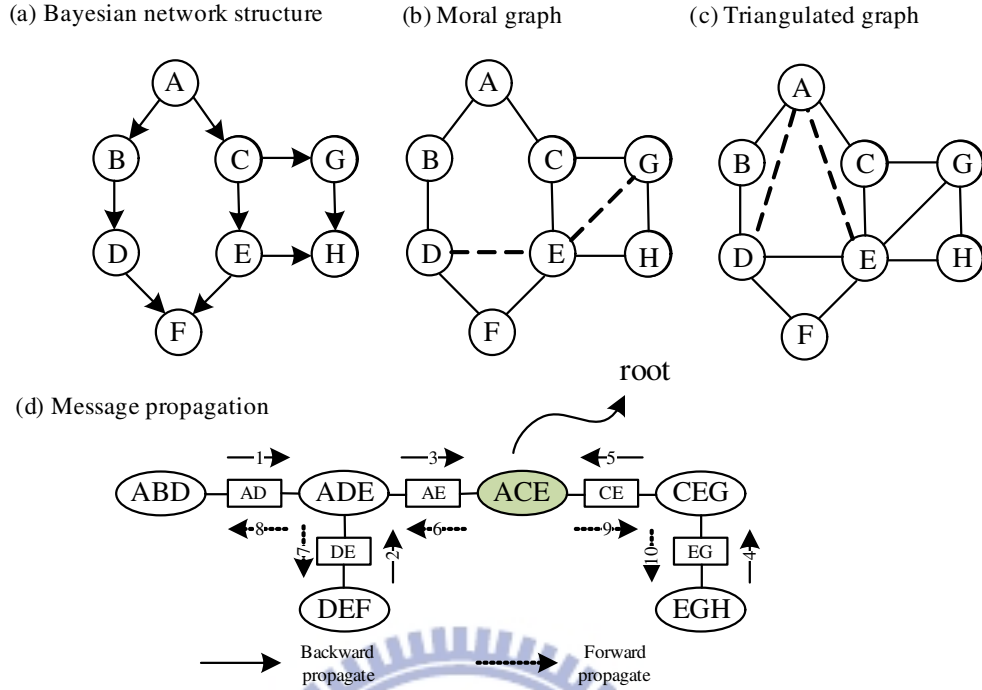


Figure 3.2: (a) Original Bayesian network. (b) Corresponding moral graph. The newly added arcs are shown with dashed lines in the moralized graph. (c) Corresponding triangulated graph with two added chords (dashed). (d) Junction tree structure with corresponding cliques and separators. During the message propagation, messages are passed away from clique  $ACE$ , beginning with  $ACE$ ; these message passing route is indicated by the arrows. The numbers indicate a possible message passing order.

an edge joining two nodes that are not adjacent in the cycle. Because any chord-less cycle has at most three nodes, chordal graphs are also called triangulated graphs. A specific moral graph can have many different triangulated graphs. Several triangulation methods have been proposed, but none of them guarantee the generation of the minimum added chords. Note that it is NP-hard to find the minimum added chords. The triangulated graph is shown in Figure 3.2(c).

### Construction of junction tree:

A *junction tree* is a mapping of a triangulated graph into a tree that can be used for speeding up the message propagation in the graph. An intermediate result of the process is called a clique graph. The basic operating unit or node in a junction tree is called a *clique*, formed by the maximum set of fully-connected nodes in a triangulated graph. In a clique

graph, each clique-node corresponds to a clique, and two clique-nodes are connected by an edge if they have a common node in the graph. A junction tree is then constructed by forming a *maximal spanning tree* from the clique graph. A spanning tree of the undirected graph is a sub-graph that is a tree and connects all the vertexes. We assign the intersection of the adjacent nodes as a weight to each edge, and use this to assign a weight to a spanning tree by computing the sum of the weights of the edges in that spanning tree. The maximal spanning tree is then a spanning tree with a weight that is greater than or equal to the weight of every other spanning tree. Each edge on the junction tree has an attached *separator* consisting of the intersection of the adjacent clique-nodes. The junction tree structure is shown in Figure 3.2(d). We see that the cliques of the triangulated graph are  $EGH$ ,  $CEG$ ,  $DEF$ ,  $ACE$ ,  $ABD$ , and  $ADE$ , which are the ellipse shown in Figure 3.2(d), and the corresponding separator is shown in the square.

**Transfer of potentials:**

For each clique  $C_i$  and separator  $S_i$ , we define a non-negative potential  $\varphi(C_i)$  and  $\phi(S_i)$ , respectively. A potential is actually a table, which eventually represents a joint probability distribution of the corresponding nodes. Both  $\varphi(C_i)$  and  $\phi(S_i)$  are initialized to be unity. For each node  $X$  of the original Bayesian network, we choose one clique  $C_i$  that contains  $X$  and all parents of  $X$ , and then multiply the CPT of  $X$  with  $\varphi(C_i)$ . Figure 3.3 illustrates the initialization procedure on the potential tables of clique  $ACE$  and separator  $CE$ . In this example, node  $C$  and node  $E$  are assigned to clique  $ACE$ , but not node  $A$ . Therefore, after initialization,  $\varphi(ACE) = P(C|A)P(E|A)$ , and  $\phi(CE) = 1$ .

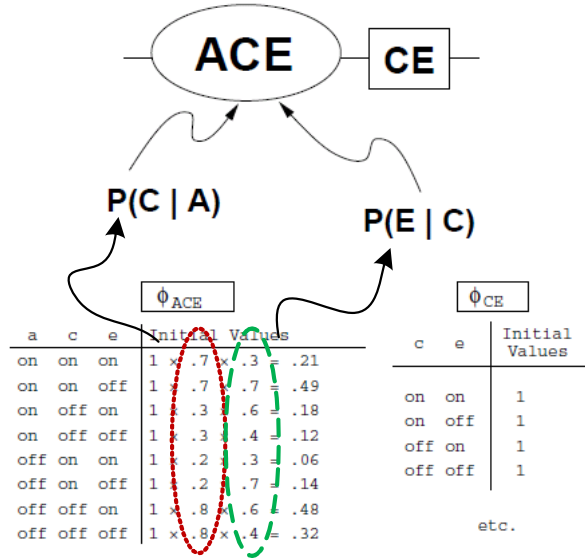


Figure 3.3: Initialization of clique  $ACE$  by multiplying the CPT of node  $C$  and node  $E$ . The separator  $CE$  is initialized to be unity.

**Propagation:**

After initializing the junction tree potentials, we now perform global propagation in order to make these potentials locally consistent. Global propagation consists of a series of local manipulations, called *message passes*, that occur between a clique  $X$  and a neighboring clique  $Y$ . A message pass from  $X$  to  $Y$  forces the belief potential of the intervening separator to be consistent with  $\varphi(X)$ . Before the propagation begins, the graph is orientated by designating one node as the root; any non-root node that is joined to only one other node is called a leaf. In the first step, messages are passed inwards: starting at the leaves, each node passes a message along the (unique) edge towards the root node. The junction tree structure guarantees that it is possible to obtain messages from all other adjoining nodes before passing the message on. This continues until the root has obtained messages from all of its adjoining nodes. The second step involves passing the messages back out: starting at the root, messages are passed in the reverse direction. The execution of the algorithm is complete when all leaves have received their messages, as shown in Figure 3.2(d). The message propagation between two nodes in a junction tree can be shown as follows:

a. First, update the separator potential.

$$\phi^*(S) = \sum_{V/S} \varphi(V). \quad (3.8)$$

b. Next, update the clique potential

$$\varphi^*(W) = \varphi(W) \cdot \phi^*(S) / \phi(S). \quad (3.9)$$

Once the algorithm is terminated, the clique potentials and separator potentials are the joint probabilities of the nodes in the clique.

The second part of propagation is the situation of the given evidence. We only keep the evidence state of nodes in the corresponding cliques, and replace the other state values with zero; we then perform the message propagation again. Upon completion, the clique potentials and separator potentials are proportional to the joint probabilities of the nodes in the clique. We just need to normalize the potential, the correct joint probabilities will then be obtained.

There are two problems related to the junction tree algorithm. First, the optimal result of the triangulation is NP-complete. Second, the size of the cliques can be prohibitive, in terms of memory requirements as well as the computational cost of the propagation. Each message-passing step requires marginalization, which, for the case of tabular CPTs, is exponential with respect to the size of the largest clique. Therefore, in the case of a large-scale or complex system, this algorithm is inefficient.

### 3.3.2 Conditioning Algorithm

Before we describe the conditioning algorithm, we will review Pearl's poly-tree algorithm, which is one of the methods used for message propagation and will be used in the conditional algorithm. Pearl's algorithm for computing the posterior of any variable in a belief network (without loops) exploits the fact that each variable  $X$  separates the belief network into two disjoint parts, one above  $X$ , and the other below  $X$ . This algorithm denotes four elements

in the propagation procedure. The *predictive messages*  $\pi_{U,X}$  which are passed down from each parent  $U$  of  $X$ , and the *likelihood messages*  $\lambda_{Y,X}$  which are passed up from each child  $Y$  of  $X$ . These message are combined to yield the *predictive support*  $\pi_X$ , and the *likelihood support*  $\lambda_X$ . The posterior for  $X$  is obtained by normalizing the product of  $\pi_X$  and  $\lambda_X$ .

Pearl's algorithm is defined by these four elements as follows:

Suppose a given node  $X$  having parents  $U_1, \dots, U_m$  and children  $Y_1, \dots, Y_n$ , as shown in Figure 3.4.  $P(x|u_1, \dots, u_m)$  is a shorthand for CPT of each node  $X$   $P(X = x|U_1 = u_1, \dots, U_m = u_m)$ . Evidence is represented with  $E$ , with evidence above  $X$  (evidence at the ancestors of  $X$ ) written as  $e_X^+$  and evidence below  $X$  (evidence at the descendants of  $X$ ) written as  $e_X^-$ . Further, let  $e_{X,U}^+$  denote the evidence above  $X$  and its parents  $U$ , and let  $e_{X,Y}^-$  denote the evidence below  $X$  and its child  $Y$ . Then, the  $\pi$  and  $\lambda$  functions are calculated in terms of the probabilities involving  $X$ , the parents and the children of  $X$ , and the evidence in the Bayesian network.

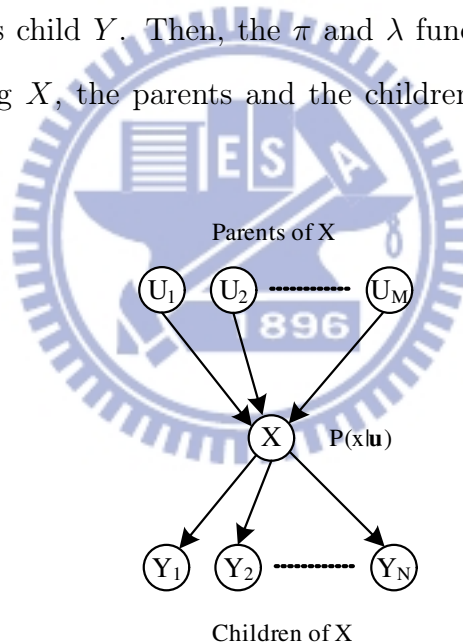


Figure 3.4: Part of poly-tree network for each node  $X$  with parents  $U_1, \dots, U_M$  and children  $Y_1, \dots, Y_N$ .

1. Posterior for variable  $X$ :

$$P_{X|e}(x) \propto \pi_X(x)\lambda_X(x). \quad (3.10)$$

2. Predictive support for  $X$ :



$$\begin{aligned}
\pi_X(x) &= P_{X|e_X^+}(x) \\
&= \sum_{u_1, \dots, u_m} P(x|u_1, \dots, u_m) \cdot \prod_i \pi_{U_i, X}(u_i).
\end{aligned} \tag{3.11}$$

3. Likelihood support for  $X$ :

$$\lambda_X(x) \propto P_{e_X^-|X}(x) = \prod_{j=1}^n \lambda_{Y_j, X}(x). \tag{3.12}$$

4. Predictive message sent to child  $Y_k$ :

$$\pi_{X, Y_k}(x) = P_{X|e_{X, Y_k}^-}(x) \propto \pi_X(x) \prod_{j=1, j \neq k}^n \lambda_{Y_j, X}(x) \tag{3.13}$$

5. Likelihood message sent to parent  $U_k$ :

$$\begin{aligned}
\lambda_{X, U_k}(u_k) &= P_{e_{X, U_k}^-|U_k}(u_k) \\
&= \sum_x \sum_{u_1, \dots, u_m} \lambda(x) P(x|u_1, \dots, u_m) \prod_{j=1, j \neq k} \pi_{U_j, X}(u_j).
\end{aligned} \tag{3.14}$$

The details of these formulas are not as important to our discussion as to the understanding of the conditional algorithm. In conclusion, Pearl's algorithm is similar to the classic *forward-backward* algorithm for information sharing. The posterior for each node depends on the message sent to it by its parents and children, if any. For a more in-depth review of the belief algorithm, please refer to [15].

When the network is not singly-connected, we introduce the notion of *conditioning* in order to apply the poly-tree algorithm. In the method of conditioning, a set of nodes called a *loop cut-set* break the dependency loops in a belief network, so named because its members

are selected such that every loop (a minimal multiply connected subset of the network) is cut by at least one member of the set. After the loop cut-set identified, the conditioning algorithm requires the instantiation of the members of the cut-set. Combinations of the instantiations of the loop-cut-set nodes are the *instances* of the cut-set. In the context of some observed evidence, the instances are solved with an efficient method for solving single-connected networks. We apply Pearl's algorithm for solving the single-connected networks. For single-connected networks, this algorithm is linear with respect to the size of the network. Finally, in the method of conditioning, the answers of the single-connected sub-problems are combined to calculate the final probability of interest.

The following are the main steps of the conditioning algorithm:

Suppose a multiply connected network  $D$  with a loop cut-set  $C = \{C_1, \dots, C_m\}$  and evidence node  $E = e$ . For any given node  $X$ , the method of conditioning calls for the propagation of this evidence in each instance in order to calculate the updated posterior probabilities for the node  $X$  of the network. We associate with each unique instance  $c_1, \dots, c_m$  an integer label  $i$ , and denote  $P(c_1, \dots, c_m)$  as the weight of the instance  $w_i$ .

- In the first step, the weights for all instances are calculated and stored during the initialization of the priors in the network. Therefore,

$$w_i = P(c_1, \dots, c_m) = P(c_1)P(c_2|c_1)\dots P(c_m|c_1, \dots, c_{m-1}). \quad (3.15)$$

If we observe the value  $e$  of node  $E$ , then we calculate the new weight,  $w_i^*$ , of instance  $i$  as follows:

$$w_i^* = P(c_1, \dots, c_m|e) = \alpha P(e|c_1, \dots, c_m)P(c_1, \dots, c_m) = \alpha P(e|\text{instance } i) \times w_i, \quad (3.16)$$

where  $\alpha = \frac{1}{P(e)}$ , obtained by normalizing the new weights.

- In the second step, we calculate the marginal probabilities for each node in the network

for each cut-set instance, given the values assigned to the loop-cut-set nodes in that instance and evidence  $e$ . We apply Pearl's algorithm for propagating the evidence in a single-connected network to solve each instance. For each instance  $i$ , we assign a probability to each value  $x$  of node  $X$ ,

$$P(x|e, \text{instance } i) = p(x|e, c_1, \dots, c_m). \quad (3.17)$$

- In the final step, we simply sum the belief over all instances, weighted by the likelihood of the instances:

$$P(x|e) = \sum_{c_1, \dots, c_m} P(x|e, c_1, \dots, c_m) p(c_1, \dots, c_m|e) = \sum_i P(x|e, \text{instance } i) \times w_i^*. \quad (3.18)$$

For additional evidence, we repeat this procedure each time by multiplying the old weight assigned to an instance with the probability of the observed value given that instance. Therefore, the method of conditioning provides a mechanism for performing a general probabilistic inference in multiply connected belief networks.

## 3.4 Why New Algorithm?

### 3.4.1 Exact Algorithm Problem

In the previous section, we described two exact inference algorithms: junction tree algorithm and conditioning algorithm. In the case of the junction tree algorithm, we might have to spend a considerable amount of time on building the junction tree. In the triangulation step, finding the best elimination order is NP-hard. If we arbitrarily assign the order, we may add many unnecessary fill-in edges, which would lead to large cliques. There are two problems when the clique of a junction tree is large. First, the potential table is huge and grows exponentially with respect to the size of clique. For example, suppose we have 20 nodes in a

clique, and each node has two statuses, then the potential table has  $2^{20}$  entries, which need  $2^{20} \times 32$  memory; 32 is the number of bits required by a float value. If there are 25 nodes in the clique, we may need approximate 1 – GB memory to store the data. Not to mention there are always more than two statuses of the nodes. In general, the limited memory of Matlab is 2 GB~ 3 GB, the same as that of other program platforms. Further, there will be more problems if the algorithm is applied to embedded systems, which always have less memory than a PC. Maybe in the future, we will not be restricted by this problem, but now, this is an obstacle that we have to face. The other problem is that the computational time increases exponentially. In the case of a propagation, we have to calculate the potential of the separators and the cliques. If a clique is large, we need to perform a significantly high number of summations in order to obtain the marginal potential, which will waste a considerable amount of time. Therefore, there are some obstacles when the junction tree is implemented in large-scale networks.

In the case of a conditioning algorithm, unlike in the case of the junction tree algorithm, we preserve the original structure since no new edges are added. Therefore, only CPTs need be stored in the memory for computing the probabilities. However, in a multiple-connected structure, the computation of probabilities is very redundant. We need to calculate three different types of probabilities in order to obtain the marginal probabilities of the nodes given the evidence (See (3.15) to (3.18)). Further, when the loop-cut-set is large, the computation complexity increases exponentially. Finding the minimum loop cut-set is also an NP-hard problem. Therefore, this algorithm is also inappropriate for implementation in the large-scale networks because of the computational inefficiency.

### 3.4.2 Some Approaches for Large-Scale Inference

Since both exact methods collapse when applied to large-scale methods, how do we draw inference in large, complex systems? There are three main methods have been proposed for application to large systems, namely, *multiply sectioned Bayesian networks* [28], *noisy*

*Or-gate models* [26], and *hybrid inference methods* [7]. In the following paragraphs, we will briefly introduce and discuss these methods.

### **Multiply sectioned Bayesian networks (MSBNs)**

A multiply sectioned Bayesian network is an extension of the Bayesian network model for the support of flexible modeling in large and complex problem domains. An MSBN consists of a set of interrelated Bayesian sub-nets, each of which encodes an agent's knowledge concerning a sub-domain. Global consistency among sub-nets in an MSBN is achieved by communication. Once the information for all the agents is updated, we obtain complete knowledge of the system. For example, in the field of medical science, we can separate the physical structure of the body into different parts, such as brain, respiratory system, and gastrointestinal system. All of them have different but related knowledge domains. Therefore, we have several different types of doctors, who check their professional parts, and by communicating other domain knowledge to diagnose the disease. Therefore, we do not require to draw an inference in a large-scale system, but only an inference in some small sectioned networks.

The inference method is called a junction forest algorithm (see Figure 3.5). It is similar to a two-layered junction tree. The first layer is in the sectioned networks. For each agent, we construct a little junction tree to draw the inference. The second layer is on an agent. We view each agent as a supper clique and build a junction tree again to communicate with each other. Since all sectioned networks are small, we can efficiently construct a junction tree and draw an inference. Therefore, we can efficiently handle the large-scale problem.

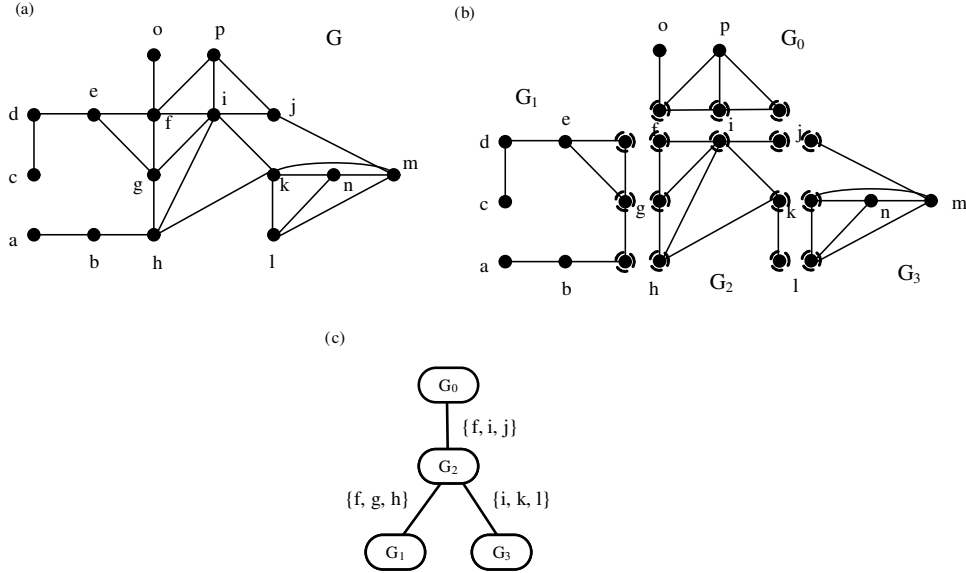


Figure 3.5: (a) $G$  is a moral graph of the Bayesian network and is a union graph in (b). (b)  $G$  is sectioned into four graphs. (c) Junction forest of  $G$ , each square represents a sub-graph in (b) and is called an agent.

### Noisy Or-gate model

Recall that Bayesian Networks require the condition probabilities of each variable given all combination of the values of its parents. Therefore, if each variable has only two states and a variable has  $p$  parents, we must specify  $2^p$  conditional probabilities for that variable. When  $p$  is large, the storage requirements as well as the inference algorithm computations become infeasible.

The idea of noisy Or-gate methods is attempt to avoid specifying every entry in the conditional probability table. In other words, we assume each parent causes child to contribute independently. Thus, the probability that parents have an effect on a child is simply the product of the probabilities of the effect of each parent.

As a simple example, medical causal models commonly assume that all the possible causes of a symptom act independently. The person who either tuberculosis ( $X$ ) or cystic fibrosis ( $Y$ ) will have a normal lung X-ray ( $E$ ). Further, tuberculosis has a failure rate of 70% with respect to showing up on an X-ray, and cystic fibrosis has a corresponding failure rate of 40%. The noisy-OR model states that if someone has both tuberculosis and cystic fibrosis,

the X-ray will have a detection failure rate of  $0.4 \times 0.7 = 0.28 = 28\%$ . In other words, they combine just like coin tosses. The CPT for this example is given in Table 3.1. The probabilities of having an abnormal lung X-ray ( $E = 1$ ) are obtained by  $1 - P(E = 0)$ .

$X$	$Y$	$P(E = 0)$	$P(E = 1)$
0	0	1	0
1	0	0.7	0.3
0	1	0.4	0.6
1	1	0.28	0.72

Table 3.1: X-ray noisy-OR example. The chance that an X-ray (E) will fail to detect two medical conditions  $X$  and  $Y$  is just the product of the individual failure chances.

Since we define the conditional probabilities, on one hand, we do not need to store all the CPT entries and can thus avoid the problem related to limited memory space. On the other hand, we can efficiently draw an inference by using the relationship between the conditional probabilities. Therefore, the noisy Or-gate model is popular in dealing with a large-scale system.

### Hybrid inference

A hybrid inference refers to the simultaneous use of an exact and an approximate inference. For example, in the case of the junction tree algorithm, if some cliques are too large to compute the potential, we can use the approximate inference, which refer to the computation of the approximate potential through simulation. The reason that we do not use an approximate inference in all cliques is the consideration of precision. The greater the number of cliques approximated, the lower is the precision obtained.

Since approximate methods, such as *Gibbs sampling* [11], are being discussed, the computational complexity is not exponential with respect to the number of variables. The other advantage is the hybrid inference method can handle relatively large range of distribution and mixture of distribution. Therefore, this method can be applied to a large-scale system.

In addition to these three methods, other approaches have been proposed, such as divorcing, which attempts to add a hidden variable to the structure to avoid large junction tree cliques. However, these approaches are suitable only for some special cases and worsen in

general. Therefore, we do not discuss these approaches here.

### 3.4.3 Why New Algorithm?

Some problems exist in the above-mentioned three methods. In the case of the MSBN approach, there is a restriction with respect to sectioning the graph in which the parents of a node can not be separated. Therefore, if the original networks is so complicated that it will have large cliques, and even after sectioning into small agents, the size of the cliques will not change. The only benefit of sectioning is that we spend less time on triangulation and the construction of the junction tree. Therefore, the MSBN approach appropriate only for some cases.

The noisy Or-gate mode, may be suitable for medical causal models maybe, but in other fields, most of the time, the conditional distribution will not have a relationship in the noisy-Or way. Therefore, we will obtain a poor inference in other field. Therefore, this method is also not suitable for general cases.

The hybrid inference, which is the most general method, can be utilized in a different domain. However, the precision of the inference will be questioned if we do not sample an adequate amount of data when using approximate methods.

Therefore, we do have an approach that can efficiently handle large-scale networks and obtain an exact or a good approximate result for a general case. In the next chapter, we will propose a new approach to a solve the problem of drawing an inference in a large and complex system.



# Chapter 4

## New Inference Approach

### 4.1 Introduction

In this chapter, we propose a new algorithm, called KLA-algorithm, for drawing an inference from Bayesian networks. While developing a KLA-algorithm, we take the following two facts into consideration. First, cluster methods (Section 3.3.1) are efficient with respect to the propagation of probabilities but require a considerable amount of memory and time to build the junction tree. In order to reduce the memory space, we look for the minimum clique and then get rid of the junction tree structure. Secondly, conditional methods (Section 3.3.2) compute the probabilities directly on the original network instead of building a cluster tree; however, only marginal probabilities instead of joint probabilities are attainable. Conditional methods also suffer from a combinatorial explosion while dealing with a large set of cut-set nodes. Therefore, it is expected that the inference algorithm can directly compute the joint probabilities and adopt a local conditioning approach [9] to refine the size of the cut-set nodes. On the basis of the above observations, we have developed a novel algorithm that combines the clique of cluster methods and the structure of conditional methods to avoid the computing inefficiency and the memory space problem and used the concept of a local conditioning approach to decrease the size of the cut-set nodes. In addition, the proposed

algorithm is capable of efficiently computing the marginal, joint, and conditional probabilities for large and complex systems.

The KLA-algorithm can neither be classified as exact inference methods nor approximate inference methods because it allows one to trade-off the quality of approximations with the computational time. In small and simple networks (e.g., poly-trees), the performance of the KLA-algorithm is as good as the other exact inference. In large and complex networks, the KLA-algorithm approximates the probabilities more accurately than other approximate inference engines. The KLA-algorithm does not require any sample data and produces relatively small cliques. Because of these enhancements, the algorithm remarkably extends the range of the realizable network complexity. Furthermore, the joint and conditional probabilities can be easily computed at the same time. Note that traditional inference algorithms can only compute either joint or conditional probabilities.

This chapter is organized as follows: In Section 4.2, we present our algorithm in detail. We present the algorithm structure in Section 4.2.1. The manner to assign a list of conditioning variables to each node and the propagation are considered in Section 4.2.2. The methods of computing joint probabilities and conditional probabilities are given in Sections 4.2.3 and 4.2.4, respectively. Finally, in Section 4.3, we discuss how to trade-off the quality of approximations with the computation time and analysis and compare the complexity with that of the other algorithms.

## 4.2 KLA-Algorithm

### 4.2.1 Structure Construction

Suppose that we have a DAG structure  $D$ , with nodes  $X_i$ ,  $i = 1, \dots, p$ , and each node has its parents  $\Pi_{X_i}$ . For each  $X_i$ , we use a clique containing the node and its parents, denoted as  $C_i$ . Now, we construct the clique graph on the basis of these cliques, connecting each pair of cliques  $C_i$  and  $C_j$  if  $X_i$  and  $X_j$  are connected, denoted as  $G_c$ . Therefore, the structure of

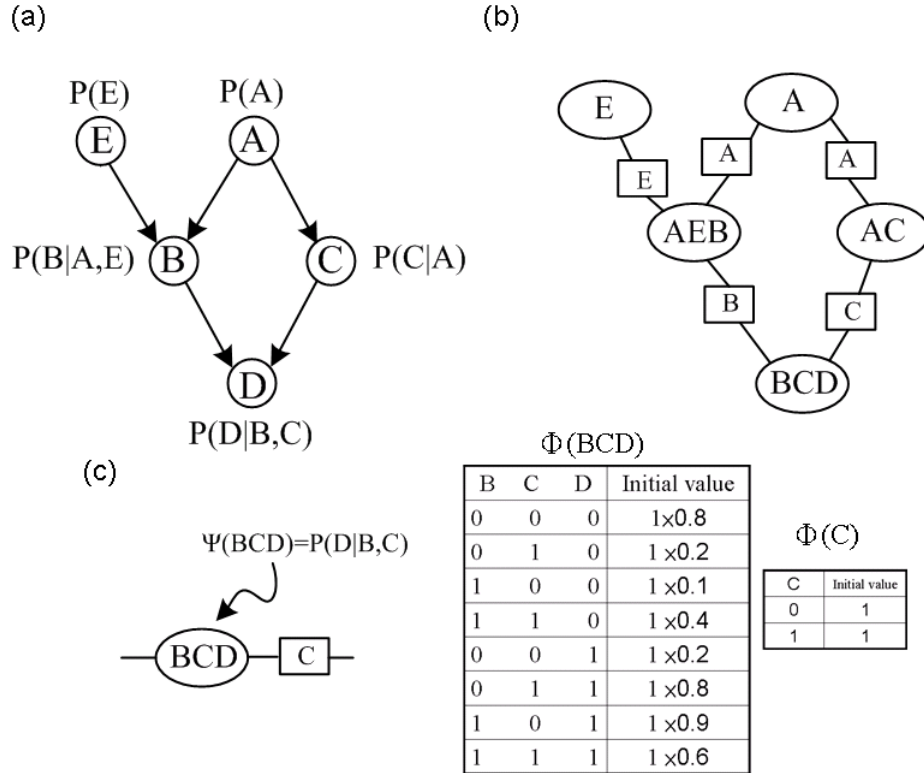


Figure 4.1: (a) Simple Bayesian network with a loop. (b) Clique graph from (a); the parents of node  $X_i$  and  $X_i$  can be contained in corresponding  $C_i$ . (c) Initial clique potential  $\varphi(C_i)$  and separator potential  $\phi(S_j)$ .

$G_c$  would be the same as the original graph structure  $D$ , only replacing the node  $X_i$  with the corresponding cliques  $C_i$ .

The structure of  $G_c$  combines the junction tree algorithm and the conditioning method. The node that we replace with a clique is to ensure the junction tree propagation and preserve the original structure but not the junction tree structure in order to not only apply the conditioning method to it but also avoid the large clique problem in the junction tree algorithm. The additional benefit of this structure is that we can compute the joint probabilities easily. This will be discussed further.

Figure 4.1 is a simple example. Figure 4.1(a) is original, and the corresponding clique graph is shown in (b).

## 4.2.2 Initializing Potentials

As in the case of the junction tree algorithm, in association with each clique we define the potential  $\varphi(C_i)$ , a non-negative function on the realizations of clique  $C_i$  and the potential  $\phi(S_i)$  for each separator. For each clique and separator, set each potential to 1. The conditional distribution  $P(X_i|\Pi_{X_i})$  of each variable  $X_i$  is multiplied into the corresponding clique potential  $\varphi(C_i)$ . Figure 4.1(c) is an example of the clique graph and its initial potential. After message propagation, the potential of the cliques will become the joint probabilities of nodes in a corresponding clique.

## 4.2.3 Propagation

The procedure of message propagation between cliques is the same as in the junction tree algorithm. See (3.15) ~ (3.18). However, the problem is that we have no junction tree properties to ensure the global consistence; hence, we have to apply the concept of the conditional algorithm here.

The conditioning method attempts to transform the multiplied graph into a poly-tree structure by identifying the loop cut-sets. After instantiating of the members of the cut-set, the answers of the single-connected poly-trees are combined to calculate the final probability of interest as described in Section 3.3.2. The only difference from Section 3.3.2 is that we use the clique message propagation to replace Pearl's message propagation algorithm; the message in the former method is based on the potential of cliques and the message in the latter method is based on the four message functions, namely,  $\pi_{U,X}$ ,  $\lambda_{Y,X}$ ,  $\pi_X$  and  $\lambda_X$ . In the conditioning method, the loop cut-set is for the entire graph, but in the KLA-algorithm, the loop cut-set is for each node. We call our loop cut-set *local loop cut-set*. This idea has been proposed by F.J. Diez [9], on the basis of Pearl's message propagation algorithm to develop the algorithm. Here, we use the clique message propagation to implement this idea. Suppose that the local loop cut-set has been identified, after instantiating all the nodes in the local loop cut-sets for the corresponding node  $X_i$  in the structure, we have the local poly-tree for

the node  $X_i$ . After propagating the message from all of the nodes in the local poly-tree of  $X_i$ , we will obtain different values of  $\varphi(C_i)$  for the corresponding different instantiations in the local loop cut-set. The true potential of  $C_i$  will be obtained by using a combination of all the  $\varphi(C_i)$  values. Therefore, once the procedures for each node  $X_i$  are terminated one by one, every potential of  $C_i$  will be correct. The following is a simple example of the propagation procedure with only one loop. Figures 4.2(a) and (b) are the message propagation of all the cliques in the graph. Since cliques  $A$  and  $E$  have no parents so we do not need to propagate a message to them.

The issue here is how to find the local loop cut-sets for the node  $X_i$ . First, we have to find the loops that  $X_i$  is at the *bottom*, which means in the directed loop,  $X_i$  have no edge flowing out to the other node in the loop. Initializing any one of the nodes in the loop except node  $X_i$  can transform the loop into a tree structure. However, in the message propagation, it is convenient to have the same propagation direction, i.e., propagate  $C_i$  to  $C_j$  for  $i < j$ . Therefore, we choose the *top* node in the loop, the node with no flow to the other nodes in the loop. Figure 4.2 is an example of the local loop cut-sets of node  $D$ .

#### 4.2.4 Joint Probabilities

Once the message propagation for all the nodes is terminated, we obtain all potentials of cliques, which is the joint probabilities for the nodes in the corresponding clique. Therefore, we can compute joint probabilities  $P(O)$  for the interest node set  $O$  in clique  $C_i$  as follows:

$$P(X_s) = \sum_{X \setminus O} \varphi(C_i). \quad (4.1)$$

where  $X$  is the node set contained in clique  $C_i$ .

However, how do we obtain the joint probabilities of the set of nodes  $O$  in the different cliques? In the junction tree algorithm, if we want to calculate the joint probabilities of nodes from different clique, we have to find a path that the union of cliques on the path can contain

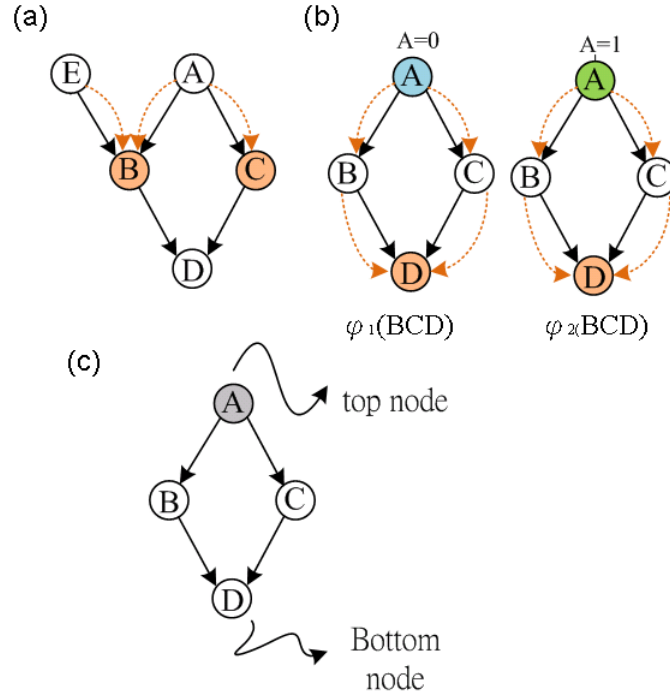


Figure 4.2: Message propagation in the nodes. (a) Update  $\varphi(AEB)$  and  $\varphi(AC)$ . (b) Since there is a loop contain node  $D$  and  $D$  is the bottom node, node  $A$  will be instantiated to 0 and 1. The correct potential of  $\varphi(BCD)$  will be obtained by  $\varphi_1(BCD) \times P(A = 0) + \varphi_2(BCD) \times P(A = 1)$ . (c) Top node and bottom node in a loop. We select top node  $A$  in the local loop cut-set of node  $D$ .

$O$ , and multiply the potential of these cliques and then divide all separators on the path. Then, we obtain the joint probabilities of a large set of nodes that contain  $O$ . Therefore, we just use (4.1) to obtain the joint probabilities of the nodes that we want. This procedure is considerably redundant and complicates. Here, we have a more efficient method to obtain the joint probabilities of any node set.

Suppose we want to compute the joint probabilities of the set of nodes  $O$ . The basic idea is to add a virtual node  $X_v$ , and connect the observed nodes  $O$  to node  $X_v$ . Therefore, the observed nodes are the parents of  $X_v$ . The conditional probabilities can be given arbitrarily. Therefore, we add a clique  $C_v$  to the clique graph. Therefore, by using the propagation method, we can obtain the potential of clique  $C_v$ , which is the joint probability of the nodes in  $C_v$ . Further, we added all the statuses of node  $X_v$ , as shown in (4.1), and obtain the required joint probability. The following Figure 4.3(a) is the example of adding the virtual node. If we

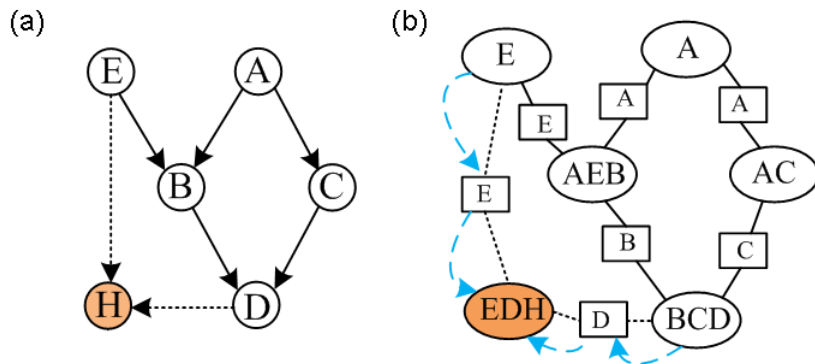


Figure 4.3: (a) Addition of a virtual node  $H$ , and assigning the interesting set of nodes to be its parents. (b) Corresponding clique-node  $EDH$ . The blue line indicates the message propagation path, and  $\varphi(EDH)$  can be obtained.

want to know  $P(E, D)$ , we assign nodes  $E$  and  $D$  to be the parents of virtual node  $H$ . After message propagation, we can obtain the joint probabilities  $P(E, D, H) = \varphi(EDH)$ . Then, the probability  $P(E, D)$  can be calculated by using (4.1), i.e.,  $P(E, D) = \sum_H P(E, D, H)$ .

#### 4.2.5 Conditional Probabilities

The conditional probabilities play the most important role in Bayesian networks. In this clique graph, we can not use message propagation methods like the junction tree or conditioning methods since they are complicated to implement. However, we propose two efficient methods here, namely, *virtual node method* (VN-method) and *forward-Backward method* (FB-method). In the case of VN-method, the basic idea is to use the method of computing joint probabilities. More precisely, if a system is given the evidence of a set of nodes  $E$  and we want to obtain the probabilities of a set of target nodes  $O$ , then, we can compute the joint probability of the set nodes  $O \cup E$ . Next, we find the probability values  $P$  for the target node set  $O$  corresponding to the particular evidence state of node set  $E$ . Once terminated, we normalize  $P$ ;  $P$  is the conditional probability we want. The above procedure can be represented by the following simple formula:

$$P(O|E = e) = \frac{P(O, E = e)}{P(E = e)}. \quad (4.2)$$

In the first and second steps, we obtain  $P(O, E = e)$ , and in the third step, the normalized term is just  $P(E = e)$ .

In our example of Figure 4.3, suppose we have evidence  $E = 1$ , and we want to find the conditional probabilities of  $D$ ,  $P(D|E = 1)$ , we can calculate the joint probabilities of  $P(E, D)$  and find the value of  $P(D, E = 1)$  in the table of  $P(E, D)$ . Then, we normalize the values and obtain the conditional probabilities  $P(D|E = 1)$ .

However, when there are many evidence nodes in the system, the number of parents of the virtual node increases and the size of the local loop cut-set of the virtual node will be large because there might be many loops contained in the virtual node. Hence, the computation will become very efficient. Therefore, the FB-method reconsiders the forward-backward passing as in the junction tree algorithm. First, we propagate a message from the top to the bottom called forward propagation and then propagate the message from the bottom to the top called backward propagation. Since our structure has loops, for forward propagation, we use a conditioning method to guarantee that the result is correct, and for backward propagation, we use another message propagation method called *loopy belief propagation*, which was proposed by PEARL and use it for drawing an approximate inference in a wide variety of BN models [18]. Loopy belief propagation is used for handling the undirected graph with a loop by modifying the message propagation path. We can obtain an approximate inference result if the message propagation can converge. We alter the loopy belief propagation to suit the KLA-algorithm and make it easier to implement; the procedures are shown in Figure 4.4. Any clique  $C_i$  has to receive the message from  $C_j$  if  $C_i$  and  $C_j$  are adjoining and  $j > i$ . When there is a loop containing the clique-nodes  $C_i$  and  $C_j$ , the original propagation may not guarantee that the message in  $C_i$  is consistent with all of  $C_j$ . Therefore, we use iteration to re-propagate the message until the message in  $C_i$  does not change. Further, if message in  $C_i$  can be converged,  $C_i$  will be consistent with all of  $C_j$ . The message will not always converge every time, but if it can converge, the result will be very close to the exact inference. Some researches propose an error assessment for loopy



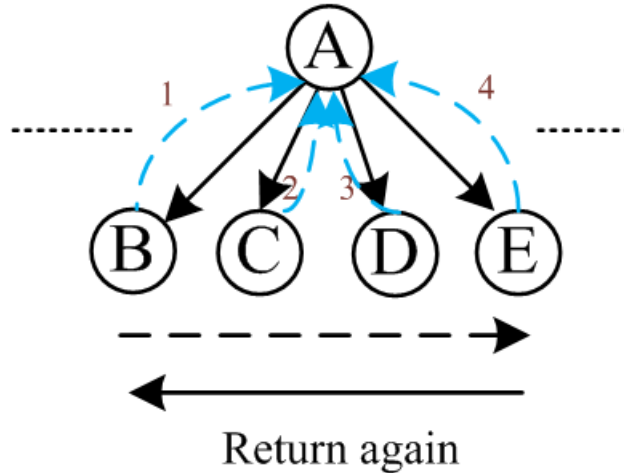


Figure 4.4: Propagation path of loopy belief propagation. Blue line represents the backward propagation path. After clique-nodes  $B$ ,  $C$ ,  $D$  and  $E$  have propagated the message to clique-node  $A$ , return again until all of clique potentials are consistency.

propagation and the situation under which the convergence can be guaranteed.

In conclusion, the procedure for computing conditional probabilities in a forward-backward manner can be shown as follows:

1. Modify the potential of the evidence node, and keep the evidence state and replaced the other state with zero.
2. Move the evidence node to the top, which means change the arrow direction of the evidence node to outward.
3. Forward propagation: the message is passed from  $C_i$  to  $C_j$  if they are adjoining and  $i < j$ .
4. Backward propagation: the message is passed from  $C_j$  to  $C_i$ .

### 4.3 Inference in Large System

In a large, complex system, we might have a large size of the local loop cut-sets for some nodes. Therefore, as in the case of conditioning methods, the computation complexity will

grow exponentially with respect to the size of the loop cut-sets. In this section, we will discuss the relationship between the complexity of the network and the time of computation. Then, we proposed a good approximate method which allows one to trade-off the quality of approximations with the computation time. In the end, we discuss the algorithm computation complexity in different types of network structures and compare with that of other exact inference.

### 4.3.1 Complexity of Graph vs. Computing Time

We know that the computation complexity of the propagation in the KLA-algorithm is  $O(N \cdot e^p)$ , where  $p$  is the largest size of the loop cut-sets and  $N$  is the number of nodes. The value  $p$  can be viewed as an index of graph complexity. When  $p$  is large, there are many loops in the graph, and the graph is complex. However, it is not practical to find the values of  $p$  each time in order to know the approximate computing time. Here, we introduce another simple index of graph complexity, which has a relationship with  $p$  and  $N$ . The index is the max eigenvalue of the graph adjacency matrix.

The adjacency matrix of the graph is the matrix that represents the connection of the graph. Each entry in the matrix represents the pair of nodes. If the pair of nodes has an edge between them, we will obtain the value of one in the corresponding entry, else we set zero. It is a symmetric matrix since we do not care about the arrows. If the graph is fully connected with  $N$  nodes, which is the graph with the maximum complexity. All the entries of the adjacency matrix will be one except diagonal entries. Thus, the maximum eigenvalues is  $N - 1$ . If the graph is a line with  $N$  nodes, which is the simplest graph, the maximum eigenvalue will be  $2\cos(\pi j/(N + 1))$ . If the graph between the line structure and the fully connected graph, the maximum eigenvalue will be in the scale of the corresponding max eigenvalue. Further details can be found in [16].

In Figure 4.5, we see some types of graph with their complexity and the computation time taken by the algorithm. The computation time is defined as the time from message

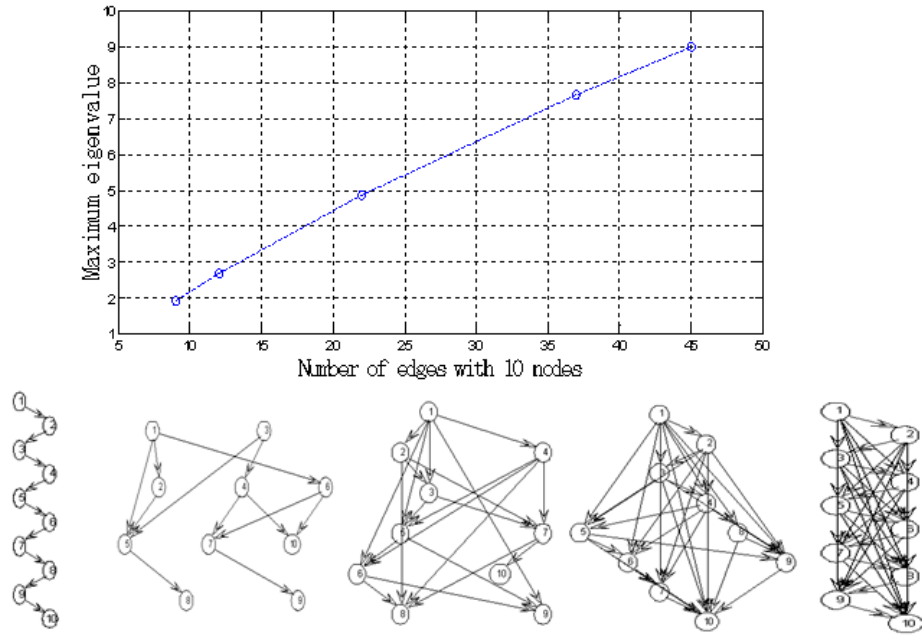
propagation to that required for obtaining the true joint probabilities of each clique (no evidence). In Figure (a), the left graph is the line structure, and the right is the fully connected structure. From the left to the right, we increase the number of edges connected to have more loops. We can find that the maximum eigenvalue increases as well. In Figure (b), in the case of the fixed number of edges, we decrease number of nodes to obtain a graph with more loops, and the maximum eigenvalue increases. Therefore, the maximum eigenvalue can be used as a good pointer of the graph complexity. Figure 4.6 is the computation time vs. different graph complexities. When the complexity increases, the computation time increases exponentially.

### 4.3.2 Good Approximation Method

In the above discussion, when the complexity of the graph is high, the computation time will become intractable. We have two ways to improve the computation time. One is by simplifying the graph structure, and the other is by reducing the size of the local loop cut-sets. The first method can modify structure learning; here, we assume that the structure is given, and therefore, the focus is on the latter method.

The local loop cut-sets for node  $X$  is found by identifying each loop on  $X$  and selecting the top node in each loop. If we initiate the top node, the loop will break and the other nodes in the loop on a different path will be independent, including the parents of  $X$ . In this situation, the local structure can be seen as the tree structure; the message propagation will have a consistent result. However, there are other situations can make the parents of the node nearly independent. That is, the loop is so long that the top node has almost no effect on the parents of the bottom node. The structure is as shown in Figure 4.7(a). The *level* in (a) is the number of nodes between the top node and the bottom node on the left path. In order to understand this concept, we use an analogy. We view a belief network as a system of a family tree. At the top of the family tree, the relationship between the family members is close. However, when every family member has its family, we are not familiar with the other

(a)



(b)

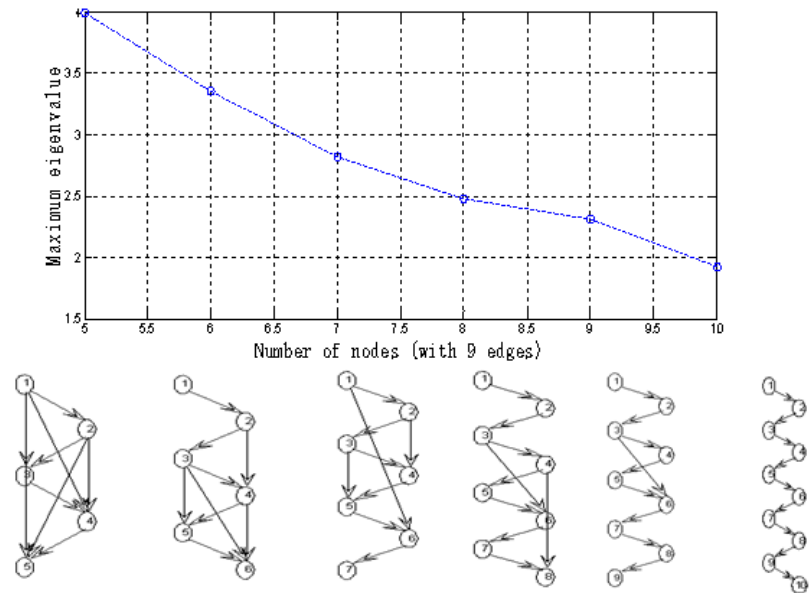


Figure 4.5: (a) Maximum eigenvalue of a structure with ten nodes with different numbers of edges. A greater number of edges will lead to more loops in the structure and a large maximum eigenvalue. The maximum eigenvalue of all the ten nodes is between that of the line structure (the left) and the that of the fully connected structure (the right). (b) Maximum eigenvalue of nine edges with different nodes. A greater number of nodes will lead to simple structures and the maximum eigenvalue will decrease.

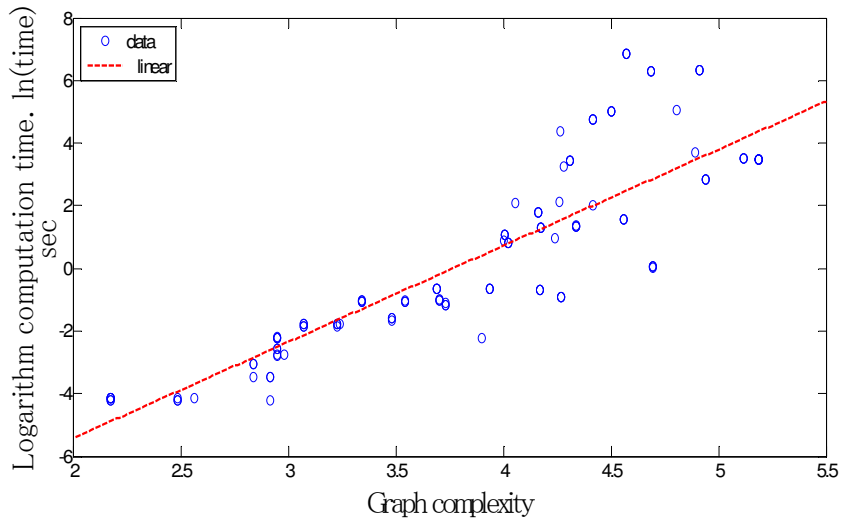


Figure 4.6: Computation time increase exponentially with an increase in the graph complexity. The value on the y-axis is the natural logarithm.

members of the family tree, except the generations close to ours. Therefore, we can mostly say that we have no relationship with the other members except our close family member. It is the same on the Bayesian networks. The loop can be seen as the family tree when the node in the loop is far away from the top node of the loop (ancestor), and the relationship would mostly disappear. Therefore, we can say that the parents of the bottom nodes in the clique would be mostly independent if the loop is sufficiently long.

The Markov property can explain this phenomenon. We know that the relationship between two nodes can be represented in the conditional probability table (CPT). For the sake of simplification, suppose both two nodes  $A$  and  $B$  are two states. Therefore, the CPT will be a  $2 \times 2$  matrix. The CPT matrix is a Markov matrix since the sum of each row is one. The independent relationship is established when the CPT matrix is singular, which implies that the rows of the matrix are the same. Therefore, no matter what the probability of node  $A$  is, the probability of node  $B$  will always be the same value. If the CPT matrix is an identical matrix, then the probability of node  $A$  will be the same as that of node  $B$ . Therefore, nodes  $A$  and  $B$  have a strong relationship. The singular or identity matrix react on the determinant of matrix, and since the CPT is the Markov matrix, the determinant of

matrix will always be smaller than or equal to one. We can judge from the smallest eigenvalue to determine the strength of the relationship. Therefore, suppose any two nodes  $X_i$  and  $X_j$  in only one path  $X_i$  to  $X_j$ , the CPT of  $P(X_j|X_i)$  would be a product of all of the CPT in the path, i.e.,  $P(X_j|X_i) = \prod_{k=i}^j P(X_k|\Pi_{X_k})$ . Therefore, the determinant of matrix will also be a product of all the CPT in the path and will decrease to zero. If the path is sufficiently long, we can say that the node  $X_i$  is approximate irrespective of node  $X_j$ .

We can see the following example. Figure 4.7(a) is a loop structure with a different level from the top node  $X_1$  to the bottom node  $X_4$ . We add a node on the left path and want to justify when path is sufficiently long, the parent node  $X_2$  would be independent of  $X_1$ . Figure 4.7(b) shows the KL-divergence between the real joint probability and the estimated joint probability of node  $X_1$  and  $X_2$ . The estimated joint probability is calculated by supposing that the node  $X_1$  is independent of  $X_2$ . The small value of KL-divergence means that  $X_1$  and  $X_2$  are nearly independent. We can find KL-divergence decreases rapidly when the level is increases. From blue line in (b), we conclude that  $X_1$  and  $X_2$  are nearly independent when the level is more than two; the KL-divergence is smaller than  $10^{-2}$ . Figure 4.7(c) shows another structure in which an outside node is added to the left path. From the green line in (b), we can conclude that  $X_1$  and  $X_2$  are nearly independent at a smaller level than the blue line. This is make sense because the outside node can share dependencies with the original parents of the node. Therefore, the dependence will not be as strong as the original. In practice, the outside node structure is common, and hence the approximate inference obtained by this approach is always good.

In any directed loop, there must be more than two paths from the top node to the bottom node. If we want to ensure that all parents of the bottom node are independent, the shortest path should be sufficiently long. Then, we can ignore this loop since all parents of the bottom node are nearly independent. Therefore, the local loop cut-sets can be reduced; the precision depends on how many loops were be abnegated. In Figure 4.8, we identify the local loop cut-set of node  $X_{12}$ , which are nodes  $X_1$ ,  $X_2$  and  $X_3$ , and have different levels to node  $X_{12}$ .

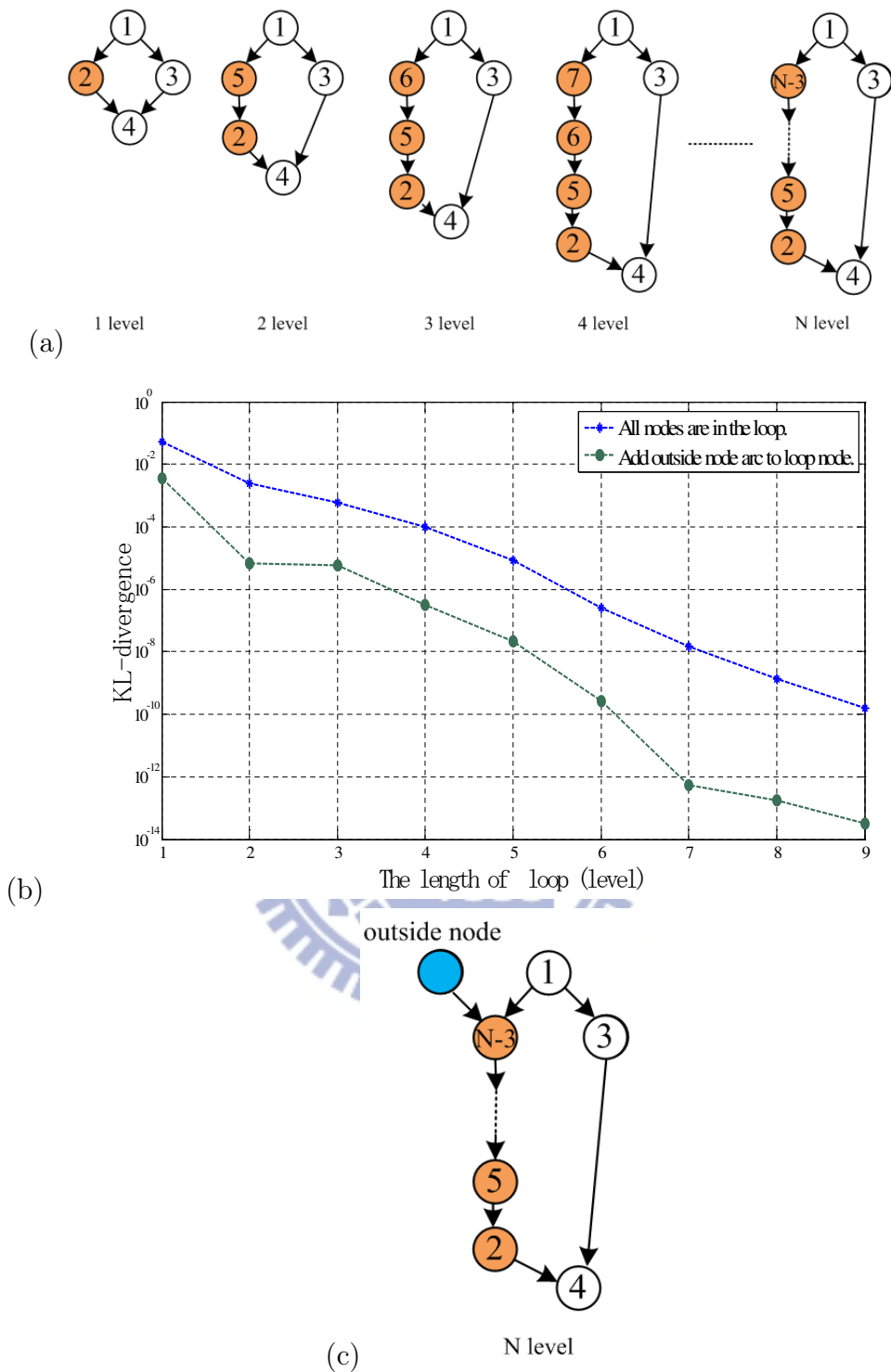


Figure 4.7: (a) Distance level from parents of bottom node  $X_4$  to top node  $X_1$  of the loop. (b) KL-divergence between real joint probability and estimated joint probability of the parents of node  $X_4$  in the loop. The estimated joint probability assumes that the nodes  $X_2$  and  $X_3$  are independent of each other. The low value KL-divergence implies that the parents of node  $X_4$  are closer to the independent node. The blue line indicates a loop structure that does not have an outside node, see (a). The green line indicates an outside node added to parent  $X_3$  (shown in (c)) and has less KL-divergence than the blue line. (c) Addition of an outside node on the left path.

If we want to save the computing time, we can use the approximate method to remove the largest level loop. Thus, node  $X_1$  will be removed from the local loop cut-set since it has three levels. The local loop cut-set can be reduced to only two member sets.

In the above example, if the CPT is close to the identity matrix, then the loop must be longer than 10 levels (the shortest path must pass more than ten nodes) and we can say that the parents of the node are nearly independent. Therefore, a different model would have a different result, and the user has to perform model selection to choose how many levels of the loop to save. If we keep all the loops, we will obtain an exact inference but will have intractable computation time in a complex system.

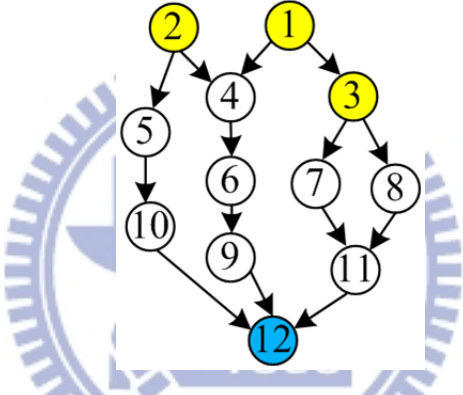


Figure 4.8: Yellow nodes  $\{X_1, X_2, X_3\}$  are the local loop cut-set of node  $X_{12}$ .  $X_1$  is three levels from  $X_{12}$ , and  $X_2$  and  $X_3$  are two levels from  $X_{12}$ . We can just keep two levels for approximation, and the reduced local loop cut-set has only two nodes,  $\{X_2, X_3\}$ .

### 4.4 Complexity of KLA-Algorithm

Since this algorithm adopts the conditioning method, the complexity of the method is  $O(N \cdot e^p)$ , where  $p$  is the largest size of the local loop cut-sets. However, if we can reduce  $p$  by the approximate methods, the complexity will show a substantial reduction. The complexity of computing conditional probabilities by adding the virtual node is  $O(e^p)$ , and can be reduced by the approximate method. The forward-backward method for computing conditional probabilities is  $O(N \cdot e^p)$ . Therefore, the size of the local loop cut-sets plays an



important role in determining the complexity of the KLA-algorithm. Although we can reduce it by using the approximate methods, some types of graphs will still have an intractable computing time. In the following paragraphs, we discuss the time performance of the algorithm in different types of graph structures.

- Poly-tree structure (no loop)

In this structure, the time performance of the KLA-algorithm is the same as that of the junction tree algorithm. Since there is no loop, we just propagate message to all the nodes. The computing complexity of both algorithm is  $O(N)$ . The structure is shown in Figure 4.9(a).

- Multiply networks with few loops

In this case, the complexity of the graph is not large, and the size of clique is small. The junction tree algorithm has a better performance than the KLA-algorithm with respect to the computing time complexity because of trade-of between memory space and computation time [10]. This trade-off is obtained if we aggregate some small cliques to be a large clique; we would need more memory space to store the cliques, but the number of cliques will decrease and result in a saving of the propagation time. The computation time of the propagation of large cliques also grows exponentially, but it is not the dominant computation time until the size of the clique is more than 1 GB.

However, since the graph is simple, the computation time of both methods is tractable. The structure is shown in Figure 4.9(b).

- Multiply networks with many long loops

In this structure, the conditioning method will be broken since the large size of loop cut-sets, As in the case of the junction tree, the large cliques contain not only the parents of the node but also other nodes in the loops. However, in our approximate method, we can reduce the size of the local loop cut-sets and apparently have the same result as that of an exact inference. The structure is shown in Figure 4.9(c).

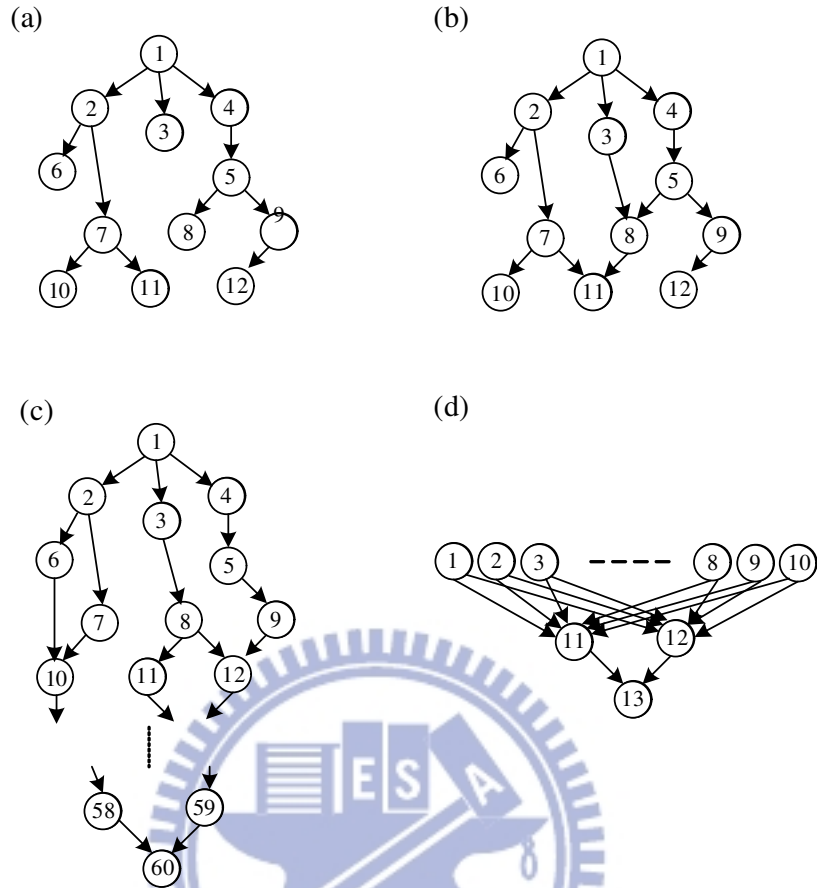


Figure 4.9: (a) Poly-tree structure. (b) Multiply networks with few loops. (c) Multiply networks with many long loops. (d) Multiply networks with many short loops.

- Multiply networks with many short loops

This structure will lead to intractable computing time for all the exact methods. Since all the loops are short, we cannot reduce the local loop cut-sets by the KLA-algorithm if the precision is concerned. Therefore, we have to use other methods to solve this problem. The structure is shown in Figure 4.9(d).

# Chapter 5

## Experiments

In order to verify the KLA-algorithm in a different graph structure, first, we design a series experiment to compare with the junction tree algorithm and discuss the performance of precision and computation time. Second, we apply the KLA-algorithm to real-world data and ozone level detection in order to carry out some simulations.

### 5.1 Verification of KLA-Algorithm

We build seven different structures with a different number of nodes, namely, 15, 30, 45, 60, 75, 90, and 105 nodes with randomly connected arcs and the maximum number of parents is four for the propagation test. By comparing the junction tree algorithm, we discuss the memory, precision, and computation time of both the algorithms. For the given evidence, we compare two computing conditional probability approaches in the KLA-algorithm. The following is the result of the simulation.

As shown in Figure 5.1, the structure with different number of nodes reflects different complexities. In the last experiment, we will use graph complexity for the discussion. Notice that the correlation between the complexity and the number of nodes is not always positive, see Section 4.4.1. Figure 5.1 points out the seven different structures and the corresponding complexity.

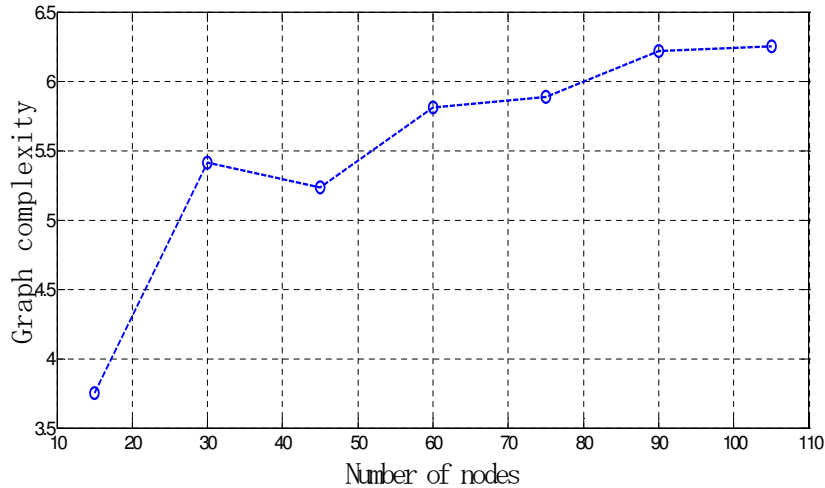


Figure 5.1: Number of nodes vs. graph complexity.

In Figure 5.2, we discuss the memory space in the running algorithm. The considerable space is not only inefficient with respect to computation but also impractical. The clique size is denoted as the number of nodes in the clique. If there are 25 nodes in a clique and each node has two states, the real memory space needs  $2^{25} \times 32 \simeq 1GB$  bits since the float value needs 32 bits to store the data. We can observe that the space of the junction tree algorithm increases exponentially when the graph complexity increases. However, the KLA-algorithm is fixed since the maximum clique of the KLA-algorithm depends on the maximum number of parents. Because the structures consist of four parents, the maximum size of clique is five (four parents and the corresponding node) in the KLA-algorithm.

Since the most important thing in Bayesian networks is the computation of the conditional probabilities for the given the evidence, the system would random choose five evidence nodes. The KLA-algorithm has two different methods for computing the conditional probabilities, one is the VN-method and the other is the FB-method. We will compare both methods and their approximate ways with a junction tree to see the performance. The followings is the simulation result.

### Computation time vs. graph complexity

#### (VN-method compared with junction tree algorithm)

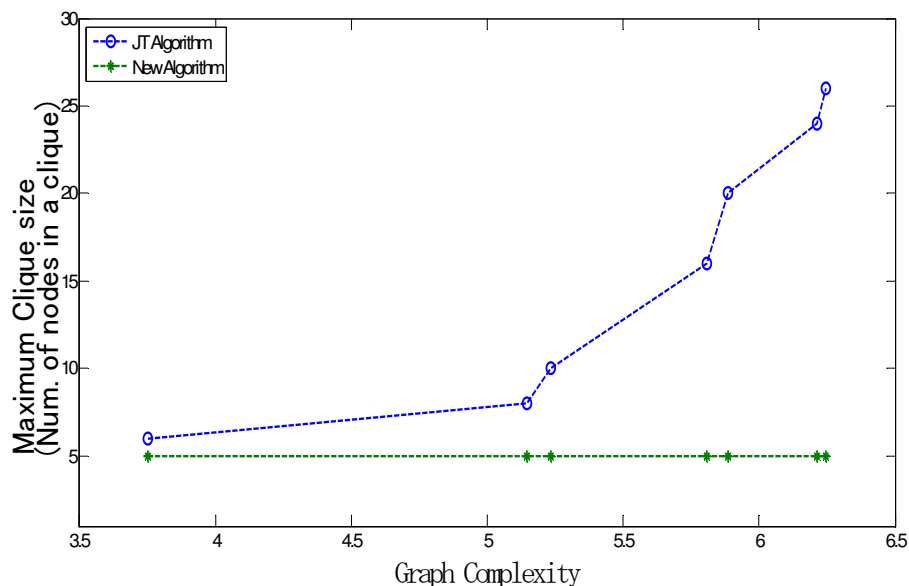


Figure 5.2: Graph complexity vs. maximum clique size. Size of the clique is represented by the number of nodes in the clique. The maximum clique size grows exponentially in junction tree algorithm but fixed in the KLA-algorithm when the graph complexity increases.

The computation time is defined only on the basis of the inference of the conditional probabilities. Thus, we do not care the time of the system construction. The number of levels is used for reducing the local loop cut-sets. For example, five levels implies that we keep the local loops in which the shortest path is less than five levels for the corresponding node. Hence, the fewer the levels, the smaller is the size of the local loop cut-set and the lower is the computation time. In Figure 5.3, we can see the junction tree always has a low computation time in the case of the networks with a complexity of less than six. However, when the complexity is more than six, the computation time increases exponentially and requires a large memory space. The VN-method has intractable computation time in a complex structure (green line). However, by using approximate methods, we can reduce the computation time to increase at a slow rate when the graph becomes more complex. When there are less than three levels, we obtain the result in 10s in these seven different structures.

The computation time of five levels (red line) and four levels (blue line) decreases dramatically in the case of the most complicated structure. This is because the structure of

(Graph complexity)	G1(3.7505)	G2(5.1459)	G3(5.2319)	G4(5.8076)
Junction Tree	0.047	0.0531s	0.0468	0.1906
KLA(VN) (all level)	0.5266	21.531	366.812	161.563
KLA(VN) (5-level)	0.5797	2.8032	23.0363	80.156
KLA(VN) (4-level)	0.5125	0.3718	6.109	9.532
KLA(VN) (3-level)	0.3141	0.5874	0.3626	0.766
KLA(VN) (2-level)	0.1156	0.1406	0.1874	0.297
KLA(VN) (1-level)	0.0938	0.0842	0.1812	0.297

(Graph complexity)	G5(5.8854)	G6(6.2131)	G7(6.2464)
Junction Tree	0.4718	6.1968	Out of memory
KLA(VN) (all level)	Intractable time	Intractable time	Intractable time
KLA(VN) (5-level)	2.21E+03	3.51E+03	860
KLA(VN) (4-level)	143.562	221.906	9.078
KLA(VN) (3-level)	0.844	1.625	2.031
KLA(VN) (2-level)	0.297	0.563	0.719
KLA(VN) (1-level)	0.281	0.594	0.438

Table 5.1: Computation time (seconds) of VN-method with different numbers of level approximations and junction tree algorithm. The corresponding graph is shown in Figure 5.3.

the most complex graph (105-node structure) has more long loops than in the case of the previous complexity (90-node structure). Therefore, when we consider only five levels for calculation, the 105-node structure will have fewer local loop cut-sets than 90-node structure and will spend less time on computation.

### Precision vs. graph complexity

#### (VN-method compared with junction tree algorithm)

Because the VN-method is an oriented-method, which calculates only the probabilities of the interesting nodes. In this simulation, we focus on the last-node conditional probability and compare it with the exact value calculated by the junction tree algorithm. Further, since the junction tree algorithm cannot work in a 105-node structure, the exact value of the structure is estimated by the five-level approximation. We use the K-L divergence to present the difference between the exact value and the approximate value. The K-L divergence is defined to be

$$D_{KL}(P|Q) = \sum_i P(i) \log \frac{P(i)}{Q(i)}, \quad (5.1)$$

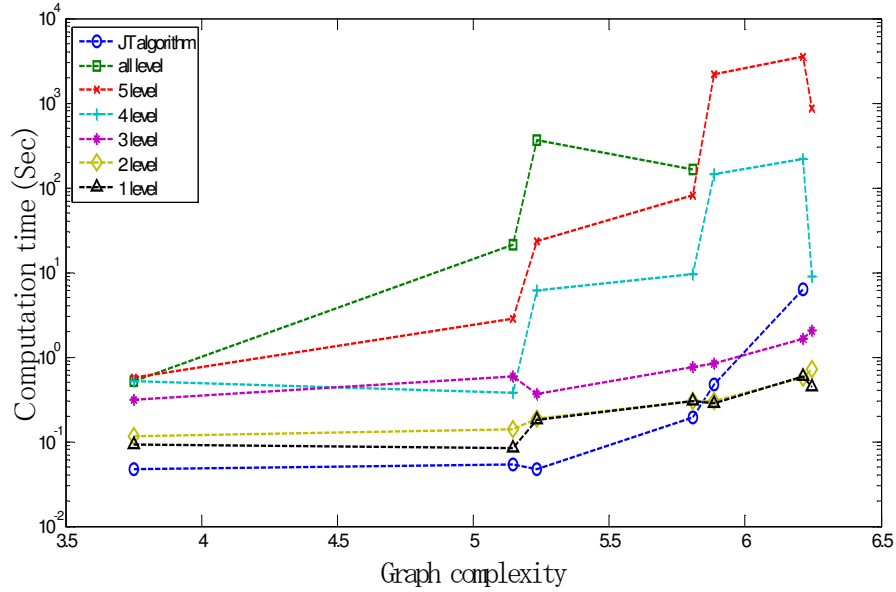


Figure 5.3: Computation time (seconds) of VN-method with different numbers of level approximations and junction tree algorithm. The computation time is represented in a logarithmic form. Junction tree algorithm lacks one point because the algorithm can not work in the most complex structure (The memory is not sufficient). The VN-method with all levels only has 4 points because the computation time is intractable in the last three structures.

and is only defined when  $P > 0$  and  $Q > 0$  for all values of  $i$ , and when the sum of  $P$  and  $Q$  both is 1. Typically,  $P$  represents the exact distribution of the data. The measure  $Q$  typically represents an approximation of  $P$ . When the value of K-L divergence is smaller than  $10^{-2}$ , we define the approximate value to be close to exact value and a good approximation.

In Figure 5.4, keep all levels would obtain the exact value; however, in the case of a large complexity the computation time is intractable. Therefore, the line of all levels (blue line) only has four values. The value of the K-L divergence will increase when the number of levels decreases. For example, the structure with 60 nodes with a complexity value of 5.8076 has a small divergence in the case of five-level approximation. However, in the case of 4-level approximation, the K-L divergence increases dramatically, which means that there is an influential top nodes in some 5-level loops, but we abnegate them. Overall, we can always obtain a good approximate value by keeping only some levels.

### Computation time vs. graph complexity

(Graph complexity)	G1(3.7505)	G2(5.1459)	G3(5.2319)	G4(5.8076)
KLA(VN) (all level)	7.32E-09	1.98E-09	9.43E-10	3.87E-09
KLA(VN) (5-level)	7.32E-09	9.08E-05	0.0813	3.87E-09
KLA(VN) (4-level)	7.32E-09	9.08E-05	0.0034	0.0133
KLA(VN) (3-level)	2.66E-04	7.62E-05	0.003	0.02
KLA(VN) (2-level)	0.0015	3.06E-04	3.04E-04	0.1506
KLA(VN) (1-level)	0.0015	1.74E-04	0.0022	0.1788

(Graph complexity)	G5(5.8854)	G6(6.2131)	G7(6.2464)
KLA(VN) (all level)	NaN	NaN	NaN
KLA(VN) (5-level)	5.08E-05	5.62E-05	0(Basis)
KLA(VN) (4-level)	1.99E-08	6.08E-05	2.01E-06
KLA(VN) (3-level)	0.0013	1.51E-05	0.00E+00
KLA(VN) (2-level)	2.55E-04	1.29E-05	7.25E-07
KLA(VN) (1-level)	2.55E-04	0.0016	9.87E-07

Table 5.2: K-L divergence between approximate value and exact value. NaN represents that we do not have approximate value. The corresponding graph is shown in Figure 5.4.

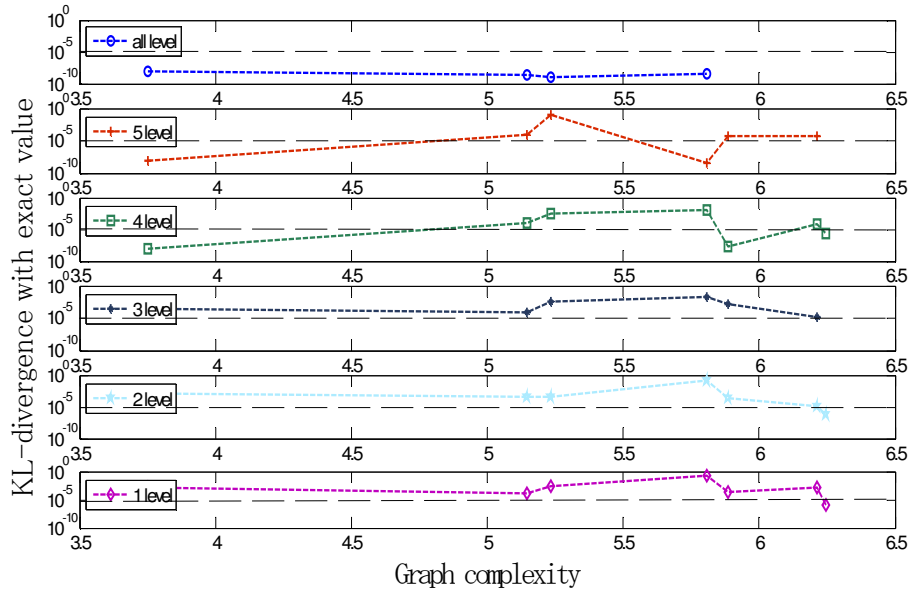


Figure 5.4: K-L divergence between approximate value and exact value. There is no clear relation between KL-divergence value and graph complexity. Most K-L divergence values are less than  $10^{-2}$ , which means that we can obtain a good approximation by the VN-method of the KLA-algorithm.



### **(FB-method compared with junction tree algorithm)**

The FB-method is the combination of conditional method (forward propagation) and loopy belief propagation (backward propagation). Thus, the computation time is the sum of the computation time in both propagation methods. The computation time of forward propagation depends on the size of the local loop cut-sets. Hence, we can save time by keeping only small level loops. The computation time of backward propagation depends on how long the potential will converge. That is, we can modify the system, and the potential value will have an effect on the convergence time. Therefore, the entire computation time will not always decrease in the case of small levels.

In Figure 5.5, except the junction tree that grows exponentially, the others grow slowly. However, the computation time of the structure with 105 nodes (most complexity) increasing drastically, especially the 1-level approximation. This is because the system is difficult to converge. The most stable approximate level is 2 and can obtain the result in a few seconds in this figure. Notice that a high-complexity graph is not always difficult to converge. It depends on the structure and the parameter value. All in all, the computation time of the FB-method always grows slowly in the small levels approximation and thus can handle the most complex system well.

### **Precision vs. graph complexity**

### **(FB-method compared with junction tree algorithm)**

Unlike the VN-method, the FB-method can update all the conditional probabilities in the networks at once, and not just of the interesting node. However, we still just look at the final node to compare with the VN-method. The exact value is still obtained from the junction tree algorithm, and the exact value of the most complexity structure is calculated by the VN-method when five levels are kept. In Figure 5.6, we can see that all the K-L divergence values are less than  $10^{-2}$ , and the value of the K-L divergence in different level approximations is very similar. That means that the difference in the levels has little effect on the precision of the FB-method. This is because in backward propagation, the loopy

(Graph complexity)	G1(3.7505)	G2(5.1459)	G3(5.2319)	G4(5.8076)
Junction Tree	0.047	0.0531s	0.0468	0.1906
KLA(FB) (7-level)	3.4167	220.073	215.5937	1.00E+03
KLA(FB) (5-level)	3.2397	44.9533	65.266	173
KLA(FB) (4-level)	2.6407	15.0363	25.625	32.891
KLA(FB) (3-level)	2.1877	6.9633	8.578	14.313
KLA(FB) (2-level)	1.8333	4.0937	5.547	8.578
KLA(FB) (1-level)	1.4113	3.1613	5.328	7.968

(Graph complexity)	G5(5.8854)	G6(6.2131)	G7(6.2464)
Junction Tree	0.4718	6.1968	Out of memory
KLA(FB) (7-level)	1.44E+03	4.04E+03	2.05E+04
KLA(FB) (5-level)	689.406	528.672	5.45E+03
KLA(FB) (4-level)	105.203	135.157	427.875
KLA(FB) (3-level)	18.406	28.672	340.75
KLA(FB) (2-level)	8.391	14.734	21.063
KLA(FB) (1-level)	8.281	27.297	923.906

Table 5.3: Computation time (seconds) of FB-method with different numbers of level approximations and junction tree algorithm. The corresponding graph is shown in Figure 5.5.

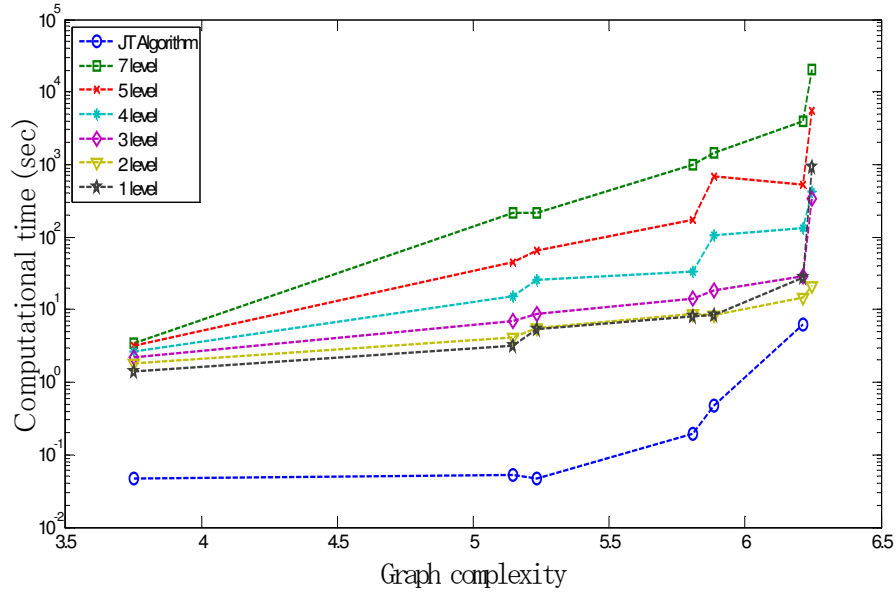


Figure 5.5: Computation time (seconds) of FB-method with different numbers of level approximations and junction tree algorithm. The computation time is represented in a logarithmic form. The FB-method when all levels are kept is replaced by that when seven levels are kept here, since the computation time of keeping all values is always intractable.

(Graph complexity)	G1(3.7505)	G2(5.1459)	G3(5.2319)	G4(5.8076)
KLA(FB) (7 level)	9.47E-04	4.22E-04	6.95E-04	0.0048
KLA(FB) (5 level)	9.47E-04	3.32E-04	0.001	0.0048
KLA(FB) (4 level)	1.50E-03	3.44E-04	3.79E-04	0.0048
KLA(FB) (3 level)	4.49E-04	1.87E-04	3.15E-05	0.0045
KLA(FB) (2 level)	4.41E-04	1.50E-04	0.0017	0.0054
KLA(FB) (1 level)	0.0038	1.03E-06	0.0098	0.0089

(Graph complexity)	G5(5.8854)	G6(6.2131)	G7(6.2464)
KLA(FB) (7 level)	1.26E-05	6.56E-05	7.00E-05
KLA(FB) (5 level)	1.38E-05	7.05E-05	3.09E-04
KLA(FB) (4 level)	3.17E-04	3.13E-04	9.03E-05
KLA(FB) (3 level)	3.35E-04	1.90E-04	1.89E-04
KLA(FB) (2 level)	9.18E-05	0.0012	5.82E-06
KLA(FB) (1 level)	7.39E-05	2.46E-04	5.16E-06

Table 5.4: K-L divergence between approximate value and exact value in FB-method. NaN represents that we do not have approximate value. The corresponding graph is shown in Figure 5.6.

belief propagation attempts to find the convergence value of all clique potentials. Thus, different approximate values in forward propagation are just the different starting points in backward propagation and cause different ways to converge. In other words, the different level approximations would change the convergence way and the convergence time, but the convergence value is caused by the system or some other factor.

Compared to Figure 5.4, the K-L divergence in the VN-method is smaller than the value in the FB-method at some points. However, both of them have good approximation in these seven structures. By Comparing the computation time of both methods, we find that the VN-method needs a smaller computation time than the FB-method. Thus, irrespective of the precision or computation time, the VN-method has a better performance than the others. Why do we need the FB-method? First, it can update all of conditional probabilities at once. When we want to know the conditional probability of other nodes, we do not need to calculate them again. Second, the VN-method would have intractable computation time when ga considerable amount of evidence is given, since the hidden node would have many parents and might cause many short loops. The following is the simulation when we change

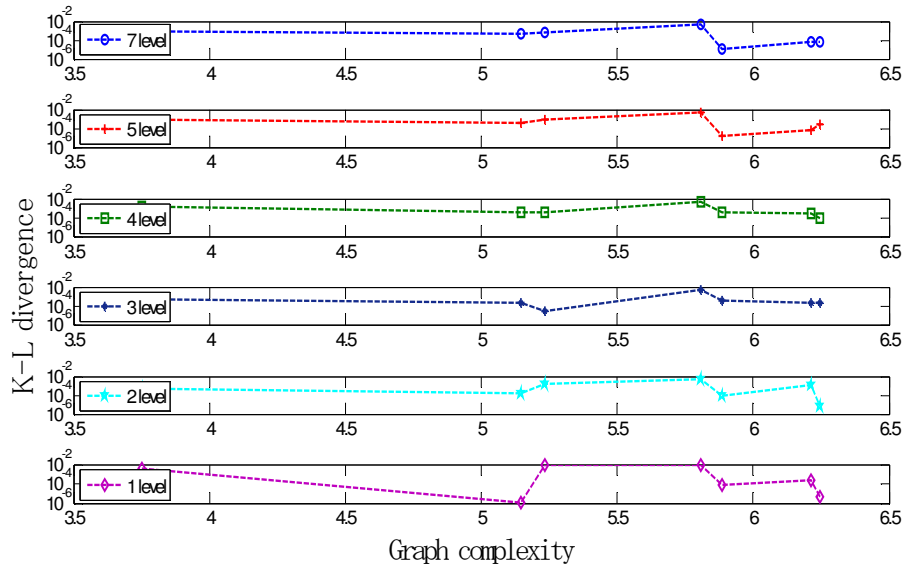


Figure 5.6: K-L divergence between approximate value and exact value in FB-method. There is no clear relation between KL-divergence value and graph complexity, and the K-L divergence in different level approximations are similar. All of the K-L divergence value are less than  $10^{-2}$ , which means that we can obtain a good approximation by FB-method of KLA-algorithm.

the number of evidence nodes in the structure.

### Different number of evidence nodes vs. computation time (FB-method compared with VN-method)

In Figure 5.7, we give different numbers of evidence nodes in the structure with 60 nodes, and use the approximate method on 2, 3 and 4-levels. The circle with a thin line is the result of the VN-method. The line moves vertically when the number of evidence nodes increases. If the number of evidence nodes keep increasing, the 2 or 3-level approximation might also be intractable. The diamond with a thick line is the result of the FB-method. The lines remains nearly unchanged when the number of evidence nodes are increasing. We can find when the number of evidence nodes is more than 17, the FB-method performs better than the VN-method except in the case of 2-level approximation. Thus, if the system just receives little evidence, we can adopt the VN-method for the calculation. If the number of evidence nodes is large, or we want the observed conditional probabilities of all nodes, then the FB-method

Number of evidence nodes	evid = 2	evid = 5	evid=8	evid = 11	evid = 14	evid = 17
KLA(VN) (2 level)	0.312	0.438	0.36	0.515	0.437	0.75
KLA(VN) (3 level)	1.344	1.671	1.735	6.688	9.406	42.672
KLA(VN) (4 level)	10.922	13.36	111.375	117.875	76.859	169.375
KLA(FB) (2 level)	7.453	7.25	7.672	6.125	6.609	7.547
KLA(FB) (3 level)	11.969	11.766	15.281	12.765	13.828	16.047
KLA(FB) (4 level)	23.594	24.734	27.578	27.61	28.578	39.891

Table 5.5: Computational time (sec) of different number of evidence nodes in two different methods of KLA-Algorithm. The corresponding graph is shown in Figure 5.7.

is appropriate.

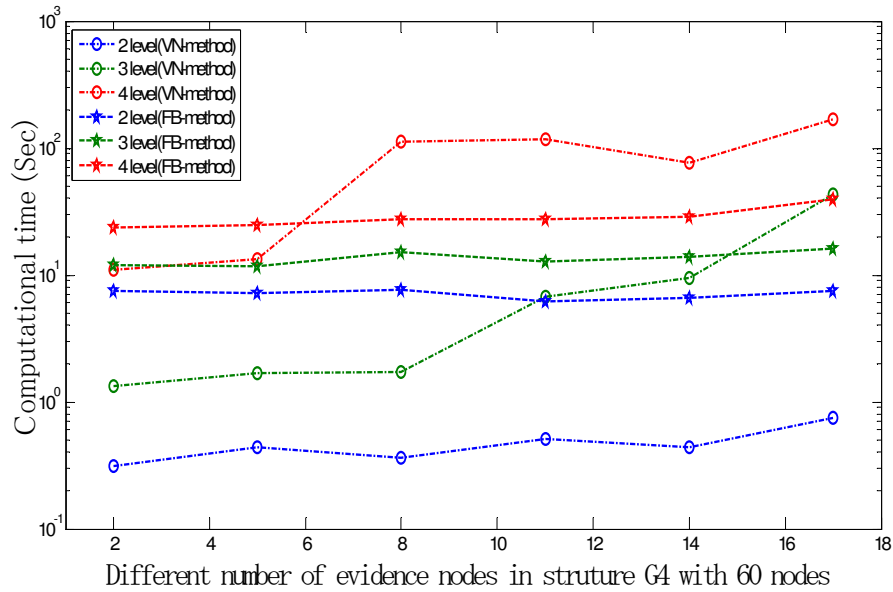


Figure 5.7: Computational time of different number of evidence nodes in two different methods. The FB-method is presented as diamond with thick line, and the VN-method is circle with thin line. The FB-method has more stable computing time than VN-method. The y-axis represents the logarithm.

In conclusion, the KLA-algorithm just needs little memory space and can obtain the conditional probability efficiently by the approximate approach. Thus, we would adopt the KLA-algorithm for the analysis of the real-world data.

## 5.2 Application to Ozone Level Detection Data Set

### 5.2.1 Problem and Data Introduction

Ozone level forecasting has been popular in environmental science and meteorology field. High concentrations of ozone near ground level can be harmful to people, animals and crops. As ozone builds up, it becomes toxic, causing shortness of breath, coughing, etc. Therefore an accurate ozone alert forecasting system is necessary to issue warnings to the public before the ozone reaches a dangerous level. There are mainly two families of methods, air dynamic and statistical models. The dynamic forecasting uses 3-D air quality models to simulate the atmospheric processes that influence the formation, transport and dispersion of ozone. The statistical methods, on the other hand, find the empirical statistical correlation between ozone and atmospheric parameters such as wind, temperature, etc. However, due to limited knowledge about the true physical and chemical mechanism, existing approaches can only use a rather small number of parameters ( $\leq 10$ ), and are still rather inaccurate. In the present exercise, we use the Bayesian networks model for ozone prediction problem, and obtain some insight into the dependence structure of the phenomena.

The data set, contains 2500+ examples with 72 continuous features, and were collected from 1998 to 2004 at the Houston, Galveston and Brazoria area. Two ground ozone level data sets are included in this collection. One is the eight hour peak set, the other is the one hour peak set. Those data Depending on the criterion for ozone days, either 2% or 5% of them. These 72 data attributes are extracted from several databases within two major federal data warehouse and one local database for ozone level detection. Table 5.6 lists all of attributes in the data file. In the present exercise, we only analysis the eight hour peak data set.

Relevant information of attributes is described in [30]. The air quality data such as hourly ozone data for this data set were extracted from the EPA Aerometric Information Retrieval System (AIRS) data set. AIRS is the national repository for information about

airborne pollution in the U.S. It provides the sensory information that is used in the regulatory feedback. Ozone exceedances of the National Air Quality Standard are based on the AIRS data. As stated in the EPA guideline, the meteorology at both surface level and upper atmospheric level affects the formation, transport and dispersion of pollutants. Thus, meteorological data from both surface and upper-air level were obtained. Specifically, hourly surface level wind speed and direction, relative humidity, pressure (SLP), sky cover were taken from NCDC (National Climate Data Center) Surface Airway data set; daily maximum temperature, precipitation were extracted from NCDC Summary of the Day data set. Fifteen upper-air variables were extracted from the NCDC Radiosonde Data of North America: temperature (T), geopotential height (HT), dew point, wind speed and direction at the 850, 700, and 500 hPa levels. To ensure data quality, all variables were pre-screened to remove erroneous entries. Then all the data sets for the same site were pre-processed, matched and combined into one set.

### 5.2.2 Bayesian network construction

Each attribute in the ozone level detection data set would correspond to a variable (node) in the Bayesian network. Thus, we have total 73 nodes in the network. Except the attribute for Ozone Day is discrete data, other attributes are continuous and we have to do discretization. To simplify, we use two or three bins for discretization. Then, we use the BNPC (Section 2.3) to build the networks. The order of nodes is according to the number of each node since some of variables are time sequence data. We prefer the events occurs first would be in the earlier order. The target node is defined as output node and at the end of order. The model would be selected after the following four procedures.

**Find the best structure in different value of threshold  $\varepsilon_2$**   
**and two different number of bins for discretization**

In BN power constructor, we can modify the two thresholds to change the complexity of structure (Section 2.4.1). First, we choose a small value on threshold  $\varepsilon_1 = 0.0008$  to pass

Name of Attribute	Description	Type	Node(BN)
WSR 0 ~ 23	WSR <sub>xx</sub> : 1-hour wind speed resultant. Ex: WSR1 is the WSR at 1am.	Cont.	No.1 ~ 24
WSR_PK	The peak value of WSR.	Cont.	No. 25
WSR_AV	The average value of WSR.	Cont.	No. 26
T 0 ~ 23	T <sub>xx</sub> : Temperature at various hours. Ex: T1 is the temperature at 1am.	Cont.	No. 27 ~ 50
T_PK	The peak value of temperature.	Cont.	No. 51
T_AV	The average value of temperature.	Cont.	No. 52
T85, T70, T50	Temperature at at 850, 700, 500 hpa level.	Cont.	No. 53, 58, 63
RH85, RH70, RH50	Relative humidity at at 850, 700, 500 hpa level.	Cont.	No. 54, 59, 64
U85, U70, U50	U-velocity <sup>1</sup> at 850, 700, 500 hpa level.	Cont.	No. 55, 60, 65
V85, V70, V50	V-velocity <sup>2</sup> at 850, 700, 500 hpa level.	Cont.	No. 56, 61, 66
HT85, HT70, HT50	Geo-potential height at 850, 700, 500 hpa level.	Cont.	No. 57, 62, 67
KI	K-Index. <sup>3</sup>	Cont.	No. 68
TT	T-Totals.	Cont.	No. 69
SLP	Sea level pressure.	Cont.	No. 70
SLP_	Sea level pressure change in the past 24 hour.	Cont.	No. 71
Precp	Precipitation.	Cont.	No. 72
Ozone Day	Two classes. 1: ozone day, 0: normal day.	Disc.	No. 73

U-velocity: East-West component of wind speed and direction.

V-velocity: South-North component of wind speed and direction.

K-Index: A measure of the heavy rain and thunderstorm potential.

Table 5.6: Seventy-two continuous attributes and one target variable in the data file.



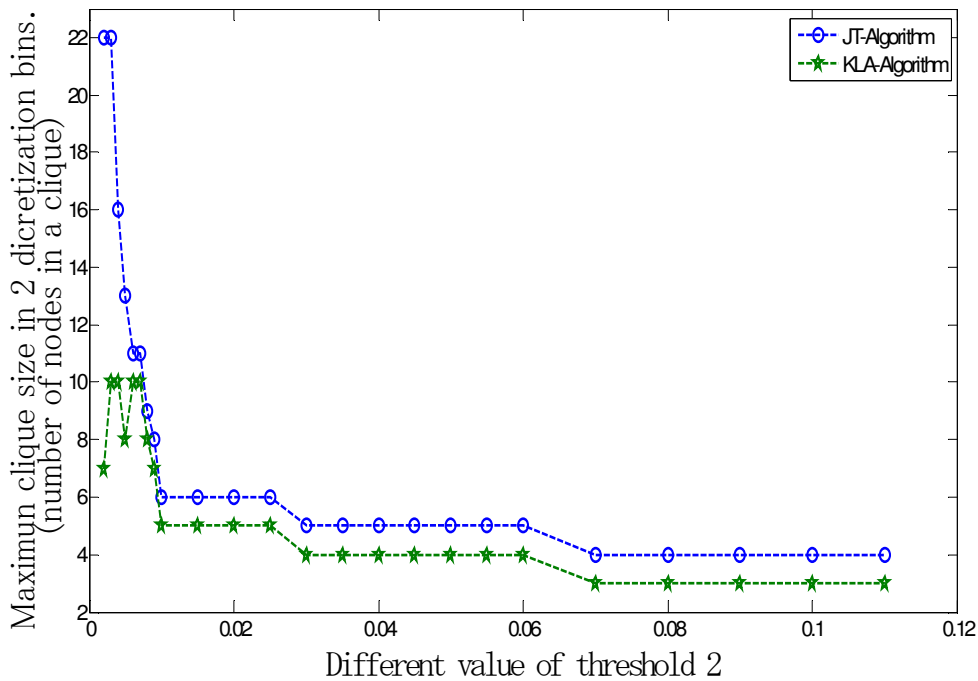
all the edges in Phase I and only change the threshold  $\varepsilon_2$  to construct different complexity networks. The large  $\varepsilon_2$  usually correspond to simple structure. Then we would do model selection by BIC test. The Figure 5.8 shows the clique size of KLA-algorithm and Junction tree algorithm in two and three bins discretization. When  $\varepsilon_2$  is smaller than 0.01, the junction tree algorithm needs a huge clique and runs out of memory in inference. For example, in two bins discretization case, the maximum size of clique at  $\varepsilon_2 = 0.001$  is thirty-three in junction tree, which means there are thirty-three nodes in a clique. Each node has three states, and we use 32 bits (float type) to store the potential value. Thence, we need  $3^{33} \times 32$  bits to store the potential value. Obviously, the memory space much larger than the system equipment. However, the KLA-algorithm just need  $3^7 \times 32$  bits in memory. Therefore, the KLA-algorithm works well in any complicated networks in this case. In this ozone prediction problem, we just use KLA-algorithm to do each test and inference.

The Figure 5.9 shows the result of BIC value with different  $\varepsilon_2$  value and two different number of bins for discretization . The larger BIC value, better the network is. The blue line is two bins discretization, and green line is three bins discretization. When threshold  $\varepsilon_2$  is smaller than 0.02, both BIC values of two line drop down. This means the networks with small  $\varepsilon_2$  is too complicate for this data and might be over-fitting. The blue line always has better BIC value than green line. It implies simple networks is more suitable for ozone prediction. The best model is at  $\varepsilon_2 = 0.04$  with two bins discretization.

#### **Find the best structure in different value of threshold $\varepsilon_1$**

After selecting the best value of  $\varepsilon_2$  and the numbers of bins, we try to find the value of  $\varepsilon_1$  to generate largest BIC value. The large  $\varepsilon_1$  usually correspond to simple structure too. We fix  $\varepsilon_2 = 0.04$  with two bins discretization. Figure 5.10 shows the result of BIC test with different threshold  $\varepsilon_1$  values. We can find that when the value of  $\varepsilon_1$  between 0.0001 to 0.0016, we would have same structure and the largest BIC value. When  $\varepsilon_1$  value is larger than 0.016, the structure is too simple to fit the data set with smaller BIC value. Thus, we do not need to change the original  $\varepsilon_1$  value.

(a)



(b)

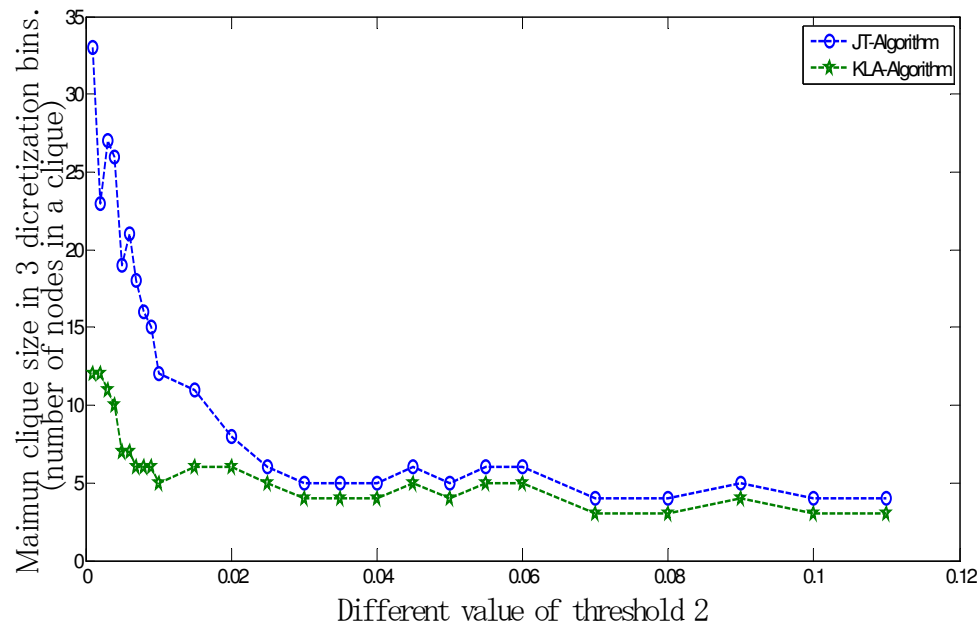


Figure 5.8: Maximum clique of JT and KLA algorithm in different value of threshold  $\varepsilon_2$ . (a) In 2-bins discretization situation. (b) In 3-bins discretization situation. Both two situation indicate the JT-algorithm need more large space than KLA-algorithm.

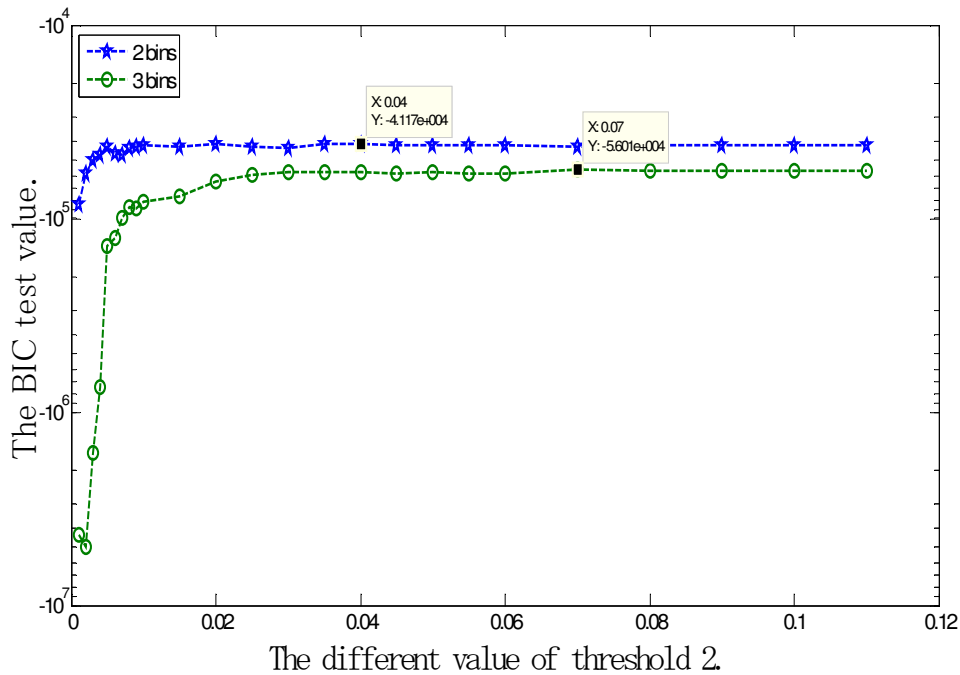


Figure 5.9: BIC value of the model with different threshold and bins. The blue line is two bins discretization and the green line is three. The two bins discretization is always better than three bins. The best network is occur at  $\varepsilon_2 = 0.04$ , with two bins discretization.

The final network is shown in Figure 5.11. We can make the following conclusions according to the networks and they fits our common sense.

1. Hourly wind resultants are only relevant with previous hour resultants. Node1  $\sim$  24 are a series connected in the networks.
2. Hourly temperature are only relevant with previous hour temperature. Node27  $\sim$  50 are a series connected in the networks, too.
3. Node40 (temperature at 1 pm.) arcs to Node51 (peak value of temperature), which means the peak temperature usually occur at 1 pm of the day.
4. Fifteen upper-air variables (Node53  $\sim$  67) have complicate relationship to each other.
5. Node59 (RH70) arcs to Node68 (K-Index) , which means the relative humidity at 700 have effect on the K-index, a measure of the heavy rain and thunderstorm potential.

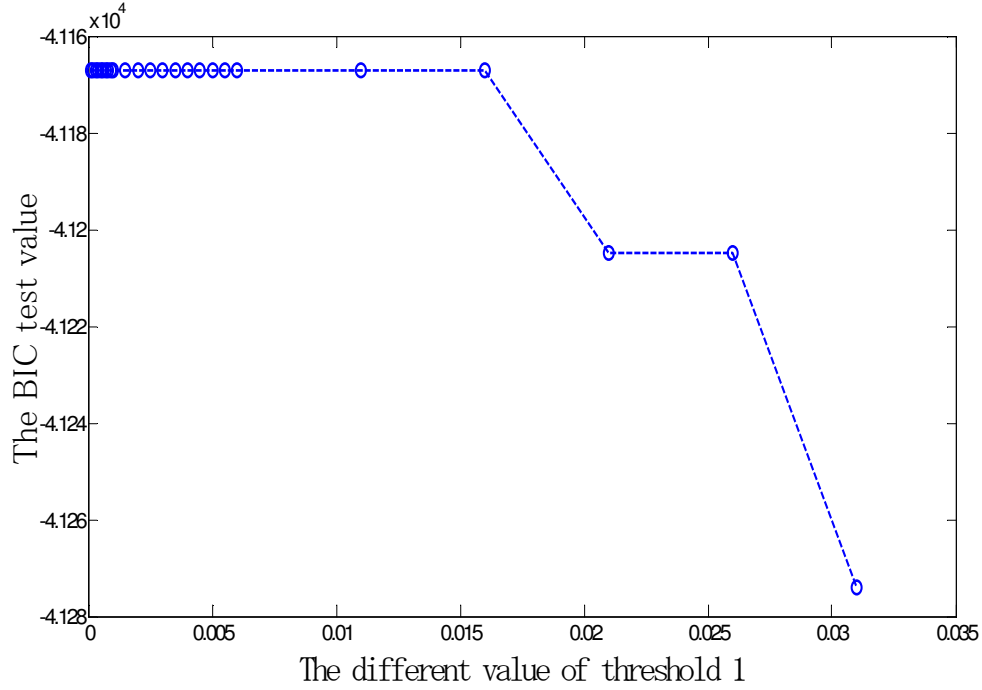


Figure 5.10: BIC value of the model with different threshold  $\varepsilon_1$ . The  $\varepsilon_1$  value smaller than 0.0016 would have the best structure.

However, there is only one attribute connected the Node73 (ozone day). It is Node12 which is wind speed resultant at 11.am. But there is no reports can prove it. We would do some tests to justify this network.

**Find the value of threshold  $v_E$  corresponding to the best precision**

To predict whether ozone day or not, the number of classification error is the most important index. However, if we predict ozone day only based on the probabilities we calculate, the result would always be a normal day. Because the result of ozone day only takes 2% or 5% of the data set. Thus, we choose a subjective threshold  $v_E$ , and whenever  $P(y = \text{“ozone day”}|x, \theta) \geq v_E$ , we issue an alert. Obviously, with different values of  $v_E$ , different precision will result. However, we would redefine the precision in this problem since the the ozone day case is too small but important to us. The precision is defined as the following formula:

$$Percission = 50\% \times \frac{\#Pred. Normal Day}{\#Normal Day} + 50\% \times \frac{\#Pred. Ozone Day}{\#Ozone Day}$$

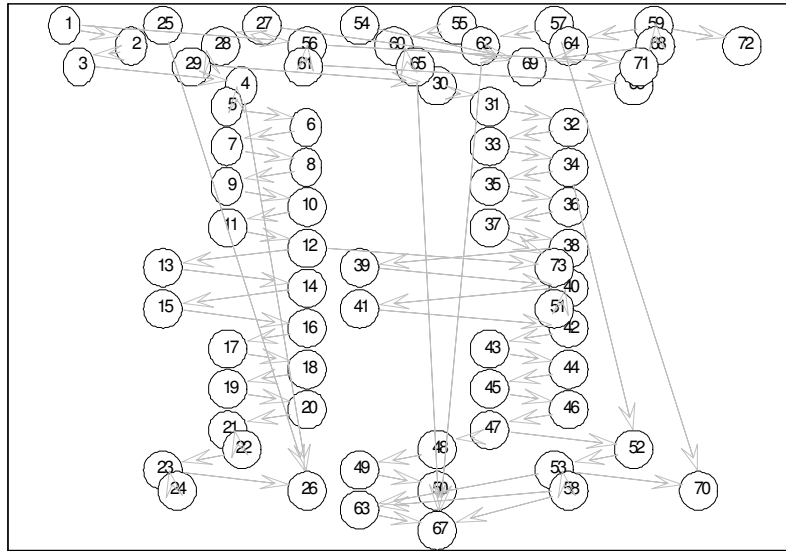


Figure 5.11: Final networks of ozone detection problem. Each circle corresponds a attributes according to the number. The target node is No.73 and only be connected by Node12 which is wind speed resultant at 11 am.

Figure 5.12 shows the result of classification precision of the validation data. The appropriate value of  $v_E$  is between 0.04 and 0.16 since we have the highest precision value 0.76. We select the middle value 0.1 as the value of  $v_E$ .

### 5.2.3 Inference simulation

In this section, we discuss a series inference performed on the test-data. Then we simulate a prediction procedure from a begin of a day. Some of the data is time-sequenced, and we only can collect some data at first. With difference evidence nodes set, the performance of model would be examined.

#### Test data result:

Test data has some missing value in difference attributes. The attributes with missing value are usually adjacent, like a section of hours temperature or wind speed resultants. In this situation, the performance of the model is shown in the following tables. We test the

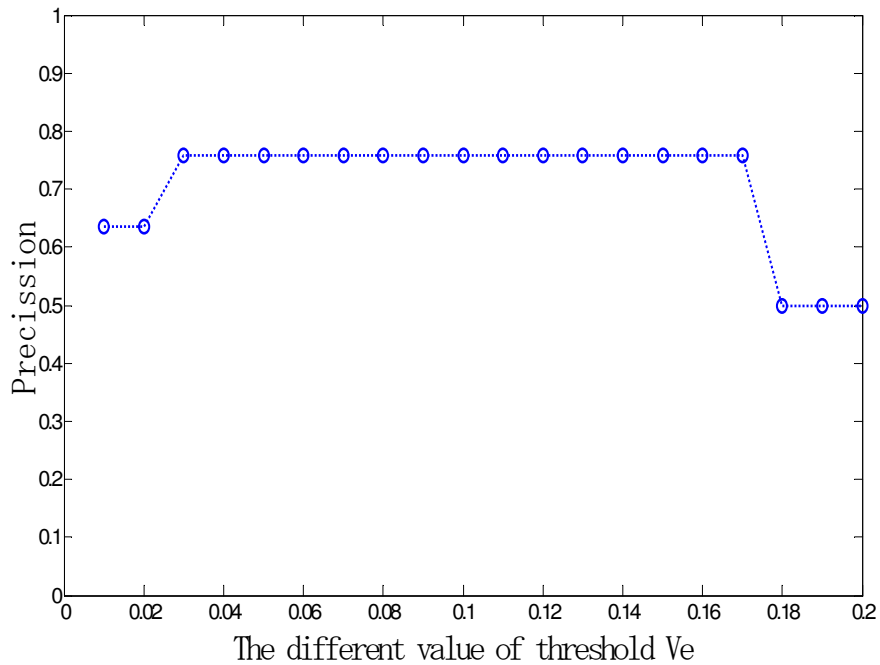


Figure 5.12: The corresponding precision with different value of  $v_E$ . The highest precision is 0.76 and  $v_E$  is between 0.04 and 0.16.

all levels and 4, 3, 2-level to approximate conditional probabilities by using the FB-method. Note that we denote “o\_” as the observed data, and “p\_” as the predicted data. “CE” is defined as cross entropy error.

Table 5.7 shows the result of different level approximations. The all levels is the exact result from junction tree algorithm. We can observe that irrespective of which number of levels approximation, we would obtain the same result, and the performance is very close to exact result. This is because the loopy belief propagation in FB-method has same converge value on different level approximations in this case. Thus, the number of level can be arbitrary chosen.

### **Prediction on a new day:**

From the beginning of a day, we start collecting the data. Every hour we can obtain the new data. However, it is meaningless to know today is ozone day after the day is over. Thus,

All level	p_normal	p_ozone	Total	2,3,4 level	p_normal	p_ozone	Total
o_normal	492	163	655	o_normal	490	165	655
o_ozone	10	22	32	o_ozone	10	22	32
Total	502	185	687	Total	500	187	687
Precision: 0.72, CE: 0.3383				Precision: 0.717, CE: 0.3387			

Table 5.7: Precision and cross entropy error of different level approximation. All level is the exact result.

we will predict hourly and keep modify our prediction after collecting data. In this situation, the inference of Bayesian networks plays an important role. Suppose except hourly data such as  $T_0 \sim 23$  and  $WSR_0 \sim 23$ , the other data have been collected from past 24 hours. The following figures shows the probability of ozone day of hourly predictions of random six days from test-data.

In Figure 5.13, the left three graphs is the normal day case, and right three graphs is the ozone day case. In the case of normal day, if the probability of ozone day less than  $v_E$  at beginning, then we can obtain a good prediction, since the ozone day probability will keep decreasing. However, if the ozone probabilities near the  $v_E$ , it is difficult to predict whether the ozone day or not until at 11 am. This is because the wind speed resultant at 11.am is the only one attributes connected with the target node (ozone or normal day). If we know the information from 11 am, the other information will useless. This is why the probability of the ozone day are same after 11 am. In the case of ozone day, we can find that the probability of ozone day is close to  $v_E$  at beginning. The first graph of ozone day case is the miss-classification case. The probability is decreasing and leads to error prediction. However, in other two graphs, the probability is near threshold  $v_E$  all of time, and thus we can have a good prediction. Overall, we will have the most exact prediction after 11 am. However, in most of time, we can predict whether the ozone day or not before the 11 am correctly. In this simulation, we can conclude that the attributes of wind speed resultant at 11.am is a key factor of ozone level, since the precision is high in ozone detection problem.

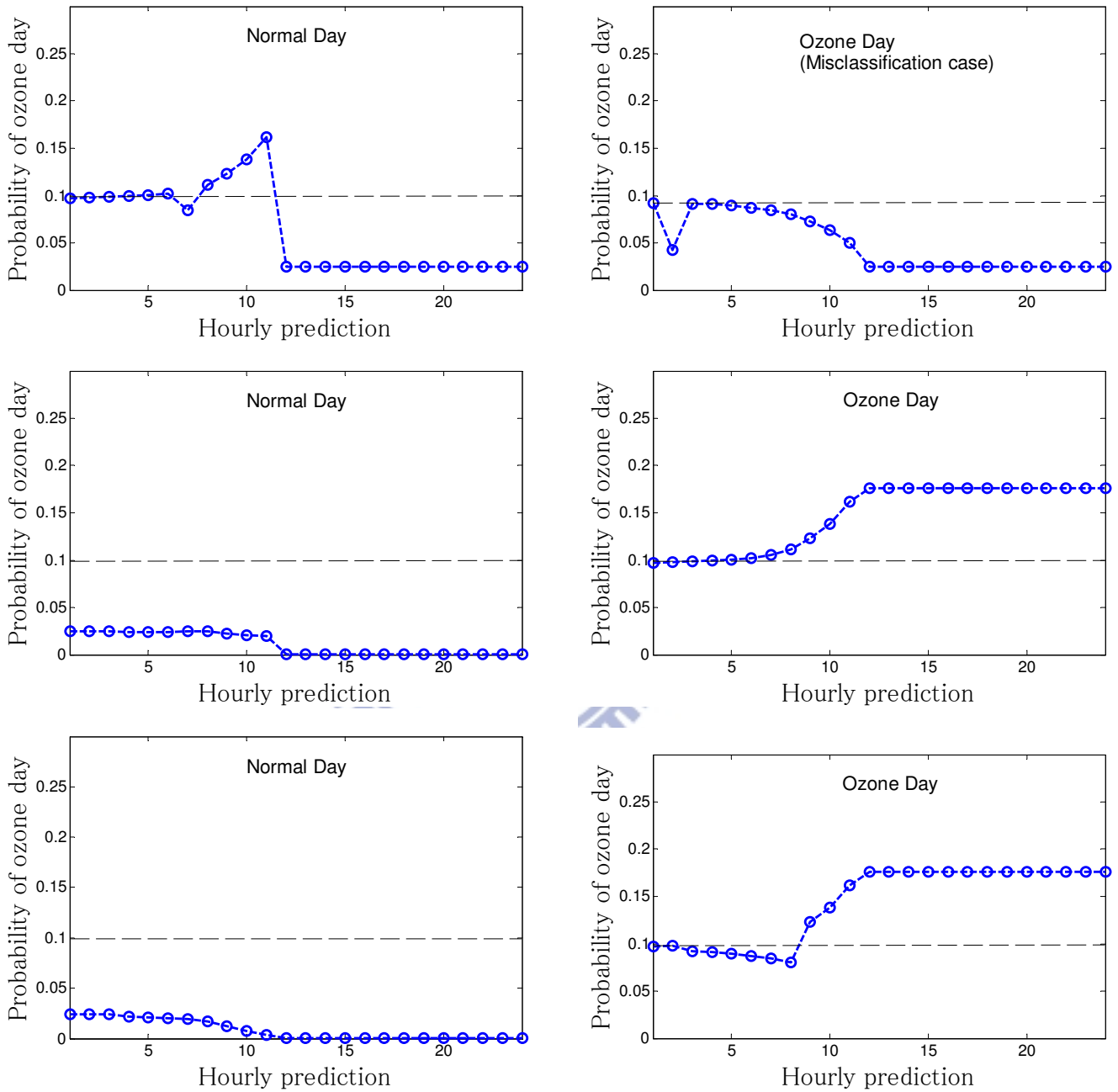


Figure 5.13: Probability of ozone day on random six days from test-data. The left column is the normal day case and right column is the ozone day case. The dashed line is the threshold  $v_E$ .



# Chapter 6

## Conclusions

In this thesis, we have considered the learning of Bayesian networks, namely structure learning, parameter learning and inference algorithms, and developed the KLA-algorithm for large-scale Bayesian networks. We also presented an application of large-scale networks. In the following paragraphs, we summarize the present exercises and derive several important conclusions.

1. We have presented a BN power constructor algorithm for the construction of large-scale Bayesian networks. The BN power constructor can build the network structure efficiently. We can tune the parameter to obtain different complexity structures and perform model selection for the best one.
2. We have described the parameter learning methods, which can easily estimate the conditional probability table in the Bayesian networks.
3. We developed the KLA-algorithm, which always has tractable computational time and a trade-off with precision. The value of the levels that we keep can be selected on the basis of the structure and the performance. The required memory in KLA-algorithm is the minimum as compared to the other inference algorithms. This advantage extends large-scale Bayesian networks to some limited resource applications.

4. The simulation results in Section 5.1 show that both the VN-method and the FB-methods of the KLA-algorithm can obtain a good approximate, which have a low K-L divergence than the exact result from a junction tree algorithm. The VN-method can calculate the conditional probability of an interesting node and just spend a short time when the size of the evidence nodes is small. The FB-method calculates the conditional probabilities of all nodes. Thus, it may spend more time, but the computation time will stabilize with different sizes of evidence nodes.
5. The application in Section 5.2 shows that large-scale Bayesian networks by the KLA-algorithm with different levels approximations can have high precision with respect to classification of the ozone day. We also simulate the missing data case, and a new day prediction by drawing an inference from Bayesian network. We still have good result in this case.

Bayesian networks using the inference algorithm proposed here is computationally efficient and require a small memory space than traditional algorithms. Future researches includes how to draw inferences in continuous-type systems and mixed-type systems and how to handle the networks with the minimum-sized clique is still out of memory space.

# Bibliography

- [1] Booker, L. B., and Hota, N. "Probabilistic Reasoning about Ship Images," *Uncertainty in Artificial Intelligence*, vol. 2, pp. 371-379, 1988.
- [2] Charniak, E., and Goldman, R. P. "Plan Recognition in Stories and in Life," *In Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, pp. 54-60, 1989.
- [3] Chickering, D. M. "Optimal Structure Identification with Greedy Search," *Journal of Machine Learning Research*, vol. 3, pp. 507-554, 2002.
- [4] Chickering, D. M., Geiger, D., and Heckerman, D. "Learning Bayesian Networks is Np-Hard," Microsoft Research, Technical Report MSR-TR-94-17, 1994.
- [5] Chow, C. K., and Liu, C.N. "Approximating Discrete Probability Distributions with Dependence Trees," *IEEE Transactions on Information Theory*, vol. 14, no. 3, pp. 462-467, 1968.
- [6] Cooper, G., and Hersovits, E. "A Bayesian Method for the Introduction of Probabilistic Networks from Data," *Machine Learning*, vol. 9, pp. 309-347, 1992.
- [7] Dawid, A. P., Kjaerulff, U., Lauritzen, S.L. "Hybrid Propagation in Junction Trees," *Advances in Intelligent Computing (IPMU)*, pp. 85-97, 1994.
- [8] Dempster, A., Laird, N., and Rubin, D. "Maximum Likelihood from Incomplete Data via the EM Algorithm," *Journal of the Royal Statistical Society, Series B*, vol. 39, pp. 1-38, 1977.

- [9] Diez, F. J. “Local Conditioning in Bayesian Networks,” *Artificial Intelligence*, vol. 87, pp. 1-20, 1996.
- [10] Fishelson, M. and Geiger, D. “Optimizing Exact Genetic Linkage Computations,” *Proceedings of 7th Conference on Computational Molecular Biology (RECOMB)*, pp. 114-121, 2003.
- [11] Geman, S. and Geman, D. “Stochastic Relaxation, Gibbs Distributions and the Bayesian Restoration of Images,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 12, pp. 609-628, 1984.
- [12] Hansson, O., and Mayer, A. “Heuristic Search as Evidential Reasoning,” *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*, 1989.
- [13] Heckerman, D. “A Tutorial on Learning with Bayesian Networks,” Microsoft Research, Technical Report MSR-TR-95-06, 1996.
- [14] Huang, C., Darwiche, A. “Inference in Belief Networks: A Procedural Guide,” *International Journal of Approximate Reasoning*, vol. 15, no. 3, pp. 225-263, 1996.
- [15] Kim, J. H. and Pearl, J. “A Computation Model for Causal and Diagnostic Reasoning in Inference Systems,” *Proceedings of the Eighth International Joint Conference on Artificial Intelligence*, Los Angeles, pp. 190-193, 1983.
- [16] Kim, J. and Wilhelm, T. “What is a complex graph?,” *Physica A: Statistical Mechanics and its Applications*, vol. 387, no. 11, pp. 2637-2652, 2008.
- [17] Leray, P., and Francois, O. “BNT Structure Learning Package: Documentation and Experiments,” Laboratoire PSI, Technical Report, 2004.
- [18] Murphy, K. P., Weiss, Y., and Jordan, M. “Loopy Belief Propagation for Approximate Inference: An Empirical Study,” *Proceedings of Uncertainty in Artificial Intelligence*, pp. 467-475, 1999.

- [19] Pearl, J. *Causality: Models, Reasoning, and Inference*. London: Cambridge University Press, 2000.
- [20] Pearl, J. *Probabilistic Reasoning in Intelligent Systems*. CA: Morgan Kaufmann, 1988.
- [21] Pearl, J. "Fusion, Propagation and Structuring in Belief Networks," *Artificial Intelligence*, vol. 29, pp. 241-288, 1986.
- [22] Pearl, J. and Verma, T.S. "A Theory of Inferred Causation," *Principles of Knowledge Representation and Reasoning: Proceedings of the 2nd International Conference*, pp. 441-452, 1991.
- [23] Schwarz, G. E. "Estimating the Dimension of a model," *Annals of Statistics*, vol.6, no. 2, pp. 461-464, 1978.
- [24] Spiegelhalter, D. J., Franklin, R. and Bull, K. "Assessment, Criticism, and Improvement of Imprecise Probabilities for a Medical Expert System," *Proceedings of the Fifth Conference on Uncertainty in Artificial Intelligence*, pp. 285-294, 1989.
- [25] Spirtes, P., Glymour, C. and Scheines, R. *Causation, Prediction and Search*. New York: Springer, 2000.
- [26] Srinivas, S. "A Generalization of the Noisy-Or Model," *Proceedings of Ninth Conference on Uncertainty in Artificial Intelligence*, pp. 208-215, 1993.
- [27] Suemondt, H. J. and Cooper, G.F. "Probabilistic Inference in Multiply Connected Belief Networks Using Loop Cutsets," *International Journal of Approximate Reasoning*, vol. 4, pp. 283-306, 1990.
- [28] Xiang, Y., Poole, D., and Beddoes, M.P. "Multiply Sectioned Bayesian Networks and Junction Forests for Large Knowledge Based Systems," *Computational Intelligence*, vol. 9, no. 2, pp. 171-220, 1993.

- [29] Zhang, N. L. and Poole, D. “A Simple Approach to Bayesian Network Computations,” *Proceedings of the Tenth Canadian Conference on Artificial Intelligence*, pp. 171-178, 1994.
- [30] Zhang, K. and Fan, W. “Forecasting Skewed Biased Stochastic Ozone Days: Analyses, Solutions and Beyond,” *Knowledge and Information Systems*, vol. 14, no. 3, pp. 299-326, 2008.

