APPROVAL SHEET

Title of Dissertation: On the Integration of Inconsistent Knowledge with Bayesian Networks

Name of Candidate: Yi Sun
Doctor of Philosophy, 2018

Dissertation and Abstract Approved: _____

Doctor Yun Peng
Professor (Chair)
Department of Computer Science
and Electrical Engineering

Date Approved: _____

Abstract

Title of Dissertation: On the Integration of Inconsistent Knowledge with Bayesian Networks

Yi Sun, Doctor of Philosophy, 2018

Directed By: Yun Peng
Professor
Department of Computer Science and Electrical Engineering
University of Maryland, Baltimore County

Incorporating or integrating new knowledge into existing knowledge bases (KBs) is critical for developing and maintaining the reliability and accuracy thereof. This thesis focuses on integrating pieces of discrete probabilistic knowledge, represented as low dimensional distributions (also called constraints), into an existing Bayesian network (BN) where the probabilistic dependency relations among the variables in these constraints are inconsistent with those captured by the network structure of the existing BN. In this situation, we say the constraints have structural inconsistencies with the BN. None of the existing methods for probabilistic knowledge integration deal with structural inconsistencies specifically. When such inconsistencies occur, these methods either do not converge or converge by modifying the constraints to remove these inconsistencies.

In this thesis we develop a theoretical framework and related methods to fill this gap. The contributions of this thesis are in three areas. First, we define structural inconsistency so it can be distinguished from other types of inconsistencies. Second, we develop a method, referred to as InconsId, to identify structural inconsistencies between a BN and a set of constraints. Third, we propose two classes of methods to overcome the structural

inconsistencies by modifying the structure of the existing BN. The class of AddNode methods adapts virtual evidence method of BN reasoning to solve the problem of integrating structurally inconsistent constraints with a BN. The class of AddLink methods addresses the same issue by compensating for the missing dependencies in the BN with added links. Variations of methods are developed in each class to balance the computational cost and solution quality. Additionally, both theoretical analysis and experiments are conducted to validate the effectiveness of these methods and compare their performance. At the end of this thesis, the developed framework and related methods are extended to solve a real-world problem of constructing a large BN from a set of small BNs.

By modifying the structure of the existing BN in a principled way, our work pioneers the research in the area of integrating structurally inconsistent constraints without sacrificing their integrity. Our research can be applied to a wide range of problems for knowledge integration with BNs without imposing structural restrictions on the inputs. Therefore, it may yield more accurate and more reliable knowledge models. It can also be extended to other related KB integration tasks such as KB merging.

On the Integration of Inconsistent Knowledge with Bayesian Networks

by
Yi Sun

Dissertation submitted to the Faculty of the Graduate School
of the University of Maryland, Baltimore County in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2018

To my advisor Dr. Peng, for guiding me through my dissertation journey.

Table of Contents

List of Tables

List of Figures

1 Introduction

1.1 The Motivations

Knowledge bases (KBs), such as Wikipedia [92], DBpedia [1, 47, 87], and Google's Knowledge Vault [25, 52, 90], are usually developed incrementally with segments of new knowledge added separately to them. Properly incorporating or integrating such new knowledge into existing knowledge bases is critical for developing and maintaining the reliability and accuracy of the KBs [7, 8, 11, 22, 50]. Since pieces of new knowledge may come from different sources, it would not be a surprise to find them inconsistent with each other or with the existing knowledge base [24, 74, 75]. In this situation we say *inconsistency* occurs during knowledge integration.

The issue of inconsistency is more notable when integrating uncertain knowledge. Uncertainty usually arises, among other things, when the knowledge is incomplete or contains noise [26, 41]. The degree of uncertainty can be conveyed through probabilities. In this thesis we focus on a specific kind of knowledge integration for uncertain knowledge, referred to as probabilistic knowledge integration, in which the knowledge base is represented as a joint probability distribution (JPD) over a set of variables of interest, and the pieces of new knowledge are represented as lower dimensional distributions (also called probabilistic constraints, or *constraints* for short). The process of probabilistic knowledge integration involves updating the JPD with the constraints until the modified JPD can satisfy all the constraints. Additional objectives can be imposed on the integration process, the most common one being the requirement of

minimizing the change to the original JPD according to certain metrics for the distance between distributions [18]. Existing works in this field mainly utilize *Iterative Proportional Fitting Procedure* (IPFP) [76] to update the JPD in various ways. Research has shown that inconsistencies can occur among the constraints if the marginal or conditional distributions do not agree for variables shared by the constraints. Methods such as GEMA and CC-IPFP [78] as well as SMOOTH [66, 86] have been developed to solve this kind of inconsistency. Their main idea is to find a JPD whose marginal distributions are as close to each of the inconsistent constraints as possible. In other words, the solution is a compromise among inconsistent constraints.

Representing a knowledge base directly using a JPD has several problems. Most notably among them are the large space requirement and high computational cost when the number of variables is large. In contrast, as a graphical model, the Bayesian network (BN) [37, 38, 59] can compactly represent the JPD and support efficient computation by adding *structures* to the flat distribution table that captures the interdependencies among the variables. These features make the BN very appealing in representing the probabilistic knowledge base [14, 48, 62], and its popularity has been increasing steadily since it was first brought to the AI community by Pearl and others in the 1980s. However, knowledge integration methods such as those based on IPFP cannot be applied directly to the BN since operations of IPFP are defined over the full JPD tables, not the BN. In order to solve this problem, IPFP was extended to E-IPFP [23, 65] by adding a structural constraint during the iteration, which forces the result to comply with the BN structure. To reduce the computational cost for large BNs, D-IPFP [23, 65] was proposed to decompose a global E-IPFP problem into a set of smaller local E-IPFP problems.

A new type of inconsistency may occur during the integration when the knowledge base is represented as a BN. Specifically, an inconsistency can occur when the probabilistic dependencies among some of the variables in a constraint disagree with the dependencies represented by the structure of the BN. We call this kind of inconsistency *structural inconsistency*. When the structural inconsistency occurs, one can choose to 1) keep the structure of the BN unchanged and integrate as much consistent knowledge in the constraint as possible, and ignore or reject the inconsistent part of the knowledge in the constraint; 2) modify the structure of the BN so that the constraint can be completely integrated into the BN; or 3) find a trade-off between the above two options.

Existing methods in this area typically choose the form of the first option. The integration of pieces of new knowledge is carried out by only updating the numerical parameters, i.e., the conditional distribution tables, of the existing BN while keeping the network structure intact. The rationale behind this is that compared to the dependency relations in the constraint, the structure of the BN often represents more stable and reliable aspects of the domain knowledge. Therefore, it is desirable to not change the BN structure frequently. E-IPFP-SMOOTH [65] is an example of this option. Similar to E-IPFP, E-IPFP-SMOOTH also maintains the structural invariance and updates only the numerical parameters of the BN while keeping its structure untouched. In addition, it also iteratively modifies the constraint during the integration process to gradually reduce or smoothen the structural inconsistencies. At convergence, the inconsistencies are removed from the constraint and the modified constraint is integrated into the BN.

This thesis aims to explore the second option to completely integrate the constraint that has structural inconsistency with the BN. The motivation for doing this is that when

the constraint is more current or comes from a more reliable source, the new dependency relations it brings should be respected. In such situations it is necessary and beneficial to modify the structure of the existing BN so that the constraint, including the new dependency relations it brings, can be integrated into the BN in its entirety. This can be illustrated by a tiny example BN in Figure 1.1. The 4 node BN is built for a hypothetical disease diagnosis domain. Its variables include two kinds of diseases, D_1 and D_2, and two symptoms, S_1 and S_2. This BN has three arcs that correlate D_1 with S_1 and D_2 with S_1 and S_2. This graphical structure implies that D_1 and S_2 are independent of each other given D_2 and S_1, i.e., $P(D_1, S_2 \mid D_2, S_1) = P(D_1 \mid D_2, S_1) \cdot P(S_2 \mid D_2, S_1)$. Hypothetically, a more recent survey on the four variables generates more accurate interdependencies among them as it targets a specific population and uses an improved method. The result of the survey is represented by a JPD $Q(D_1, D_2, S_1, S_2)$ with the conditional distribution $Q(D_1, S_2 \mid D_2, S_1) \neq Q(D_1 \mid D_2, S_1) \cdot Q(S_2 \mid D_2, S_1)$, which means D_1 and S_2 are dependent given D_2 and S_1. But this dependency correlation does not exist in the existing BN. To integrate this piece of new knowledge, the structure of the existing BN needs to be changed. One obvious suggestion for structural modification is to add a link from D_1 to S_2. The updated BN with the newly integrated dependency relation can provide more accurate results for future disease diagnoses for that specific population.

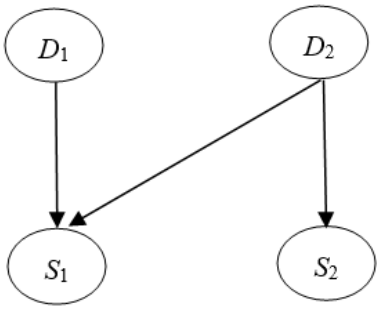


Figure 1.1 A 4 node BN for a hypothetical disease diagnosis domain

The results of our research may have practical applications for other knowledge integration tasks. One of these tasks we have investigated is to construct a large probabilistic knowledge base from several smaller knowledge bases represented as BNs. The existing solutions to this problem include but not limited to Multiply Sectioned Bayesian Network (MSBN) and MSBN-based multi-agent system proposed by Xiang [81, 82, 83, 84], as well as Agent Encapsulated Bayesian Network (AEBN) proposed by Bloemeke [5]. These solutions have strict structural restrictions on the Bayesian networks. For example, MSBN constructs a global BN from a set of subnets with shared variables, where each of the subnets is controlled by an agent. The knowledge represented by the subnets are accessed as a single large BN during inference, which is made possible by enforcing the shared variables to be identical and all parents of a shared variable to appear in one subnet in MSBN. In AEBN, each agent uses a BN to represent its domain knowledge, and exchanges beliefs with other agents via connections between identical variables. When there are multiple connections between two BNs, the probabilistic influences cannot be properly propagated and the conflicts cannot be resolved in AEBN. Such issues have limited the applications of these solutions. In contrast, methods from our research shall be able to lift these restrictions on the subnets and properly handle the structural inconsistencies among the subnets when solving this problem.

1.2 Thesis Statement

In this thesis, we develop a theoretical framework and related methods for integrating inconsistent probabilistic knowledge with Bayesian networks. Going beyond existing works in this area, our framework is able to identify the structural inconsistencies and overcome them by modifying the structure of the existing BN in a principled way. The contributions of this thesis are as follows:

First, we define structural inconsistency so that it can be distinguished from other types of inconsistencies and be properly targeted during the knowledge integration process.

Second, we establish the theorem that any dependency that is implied by the constraint but does not hold in the BN structure will cause structural inconsistencies. Based on this theorem we develop a method, named InconsId, to identify structural inconsistencies between a BN and a set of constraints. This is accomplished by first extracting dependency information from each constraint, then checking in the BN to see if any of those dependencies are not captured by the BN structure.

Third, we propose two classes of methods to address the issue of modifying the structure of the existing BN to overcome the structural inconsistencies. The class of AddNode methods adapts Pearl's virtual evidence method [59] while the class of AddLink methods addresses the issue of missing dependencies in the BN with added links. Variations of methods are developed in each class to balance the computational cost and solution quality. Both theoretical analysis and experiments are conducted to validate the effectiveness of these methods and compare their performance.

The developed framework and related methods are extended to solve the problem of constructing a large BN from a set of small BNs. Experiments show that, with the help of the integration constraints that provide the missing dependencies among the small BNs, these BNs can be successfully merged into a large BN.

This research is primarily based on the pioneering work done by Dr. Peng's research group [63, 64, 65, 66]. Their existing work enables the integration of consistent and inconsistent constraints into the BN without making changes to the BN structure. This thesis is carried out to complement their work for situations where BN structure update is necessary and beneficial when structural inconsistencies happen during the integration. Some of the preliminary work in this thesis has been presented in [71, 72].

By modifying the structure of the existing BN in a principled way, our work will pioneer the area of integrating structurally inconsistent constraints without sacrificing their integrity. Our research can be applied to a wide range of problems for knowledge integration with BNs without imposing structural restrictions on the inputs. It can also be extended to other related KB integration tasks.

1.3 Dissertation Outline

The rest of this thesis is organized as follows. In Chapter 2 we introduce some background and review the existing works related to our problem. It includes IPFP, Bayesian networks, knowledge integration methods, virtual evidence method, BN structure learning methods, MSBN, and other relevant material.

In Chapter 3 we first establish the theorem that any dependency that is implied by the constraint but does not hold in the BN structure will cause structural inconsistency. This

leads to our definition of the structurally inconsistent constraint. Based on this definition, we develop a method, named InconsId, to identify structural inconsistencies between a BN and a set of constraints. Examples are provided to show how this method works. Time complexity analysis is performed for this method. Experiments are also conducted to evaluate its performance.

In Chapter 4 we propose a class of methods, referred to as the AddNode methods, to overcome the structural inconsistencies by adding nodes to the existing BN. The basis of the AddNode methods, called AddNode-Basic, adapts the virtual evidence method to solve the structural inconsistencies between a BN and a set of constraints. Several variations of the AddNode-Basic method are also developed to balance the computational cost and solution quality, as well as to address other concerns. These include AddNode+Merge and AddNode+Factorization which can be used when it is computationally beneficial to merge small constraints or to split large constraints. This chapter also investigates the issue of efficient integration in situations where both structurally consistent and inconsistent constraints are presented. Examples are provided for those methods to show how they work. Experiments are conducted to compare their performance in different situations.

In Chapter 5 we propose another class of methods, referred to as the AddLink methods, to address the issue of structural inconsistencies by adding links to the existing BN. The basis of the AddLink methods, called AddLink-Basic, solves this problem by adding one link for each dependency that exists in the constraints but is missing in the BN. A variation of the AddLink-Basic method, named AddLink-Prune, is developed to minimize the number of links to be added to the existing BN. Reducing the number of

added links is consistent with our integration objective of minimizing the changes to the original KB, which may reduce the computational complexity for the probabilistic reasoning in the updated BN. Examples are provided to show how the methods work. Experiments are conducted to validate the effectiveness of the methods and to compare their performance.

In Chapter 6 we extend the developed framework and related methods to solve an instance of the problem of constructing a large BN from a set of small BNs with very encouraging experiment results.

In Chapter 7 we conclude our work with directions for future research.

2 Background and Related Work

This thesis focuses on the discrete probabilistic knowledge integration where the knowledge base is represented as a joint probability distribution (JPD), and the pieces of new knowledge are represented as lower dimensional distributions (also called probabilistic constraints, or *constraints* for short). In Section 2.1 we review several methods for probabilistic knowledge integration. We first introduce the Iterative Proportional Fitting Procedure (IPFP), which can be used to find a distribution that satisfies a set of constraints and guarantees that this distribution is as close to the original distribution as possible. Then we review GEMA, CC-IPFP, and SMOOTH, which extend IPFP to integrate constraints when they are inconsistent with each other.

In Section 2.2 we introduce the Bayesian network (BN) which can represent the interdependencies among the variables graphically and decompose the JPD into a set of smaller distributions associated with each variable. After presenting several important properties of the BN, we review some existing approaches for knowledge integration with a BN, including E-IPFP, D-IPFP and E-IPFP-SMOOTH.

In Section 2.3 we introduce Pearl's virtual evidence method because it inspires us to develop the class of AddNode methods for overcoming structural inconsistencies. Virtual evidence method plays an important role in some BN reasoning problems. After a brief explanation of the differences and relations among hard evidence, soft evidence, and virtual evidence in BN reasoning, we present the principle of the virtual evidence method, and show how a soft evidence represented as a distribution such as our probabilistic constraint can be converted into a virtual evidence.

In Section 2.4 we review BN-IPFP algorithms which combine Pearl's virtual evidence method with IPFP for BN reasoning with multiple soft evidences. After introducing and comparing the three versions of BN-IPFP algorithms, we provide an example to show how they work.

In Section 2.5 we compare our problem with the problem of BN structure learning. BN structure learning focuses on determining which links are the most appropriate to be added to the BN given the data, which is similar to the concern we have in the class of AddLink methods. After briefly reviewing the existing methods for BN structure learning, we explain why these methods cannot be applied directly to the problem we set out to solve.

In Section 2.6 we introduce the MSBN framework because it is related to a knowledge integration problem we will investigate, namely how to automatically construct a large BN from a set of small BNs. After reviewing its definitions for hypertree and d-sepset interface, which are the two key constraints MSBN enforces upon the small BNs, we point out its limitation in situations where small BNs do not obey structural restrictions.

The following convention is used in this thesis. To name a set of variables, we use capital italic letters such as X , Y , Z . To name their instantiations, we use lower case italic letters such as x , y , z correspondingly. To name an individual variable in a set, we use subscript to indicate its position in a set, such as X_i for the i^{th} variable in set X , and x_i for its instantiation. *P*, *Q*, *R* are specifically used to indicate probability distributions, and bold $\mathbf{P}$, $\mathbf{Q}$, $\mathbf{R}$ are used for sets of distributions. $\mathbf{Y}$ is used to represent all the possible instantiations of a set of variables Y .

2.1 Knowledge Integration with JPD

In this section we will describe the problem of probabilistic knowledge integration for knowledge bases represented as joint probability distributions and review the related methods for solving this problem.

2.1.1 The Problem of Probabilistic Knowledge Integration

In this thesis, knowledge integration refers to the process of incorporating new knowledge into an existing knowledge base. The probabilistic knowledge base can be represented using a JPD $P(X)$, where X represents the set of variables in the domain. The new knowledge that needs to be integrated into $P(X)$ usually comes from more up-to-date or more specific observations for a certain perspective of the domain. It can be represented as a lower dimensional probability distribution over a subset of X, which is called probabilistic constraint, or *constraint* for short.

We use probability distribution $R_j(Y^j)$ to denote the j^{th} piece of new knowledge or the j^{th} constraint over the set of variables Y^j, where $Y^j \subseteq X$. A set of constraints can be represented as $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$, where m is the number of constraints.

Definition 2.1 (Constraint Satisfaction) Let $P(X)$ be a JPD, and $R(Y)$ be a constraint, where $Y \subseteq X$. $P(X)$ is said to *satisfy* $R(Y)$ if $P(Y) = R(Y)$, i.e., the marginal distribution of $P(X)$ on variable set Y equals $R(Y)$.

The set of all JPDs that satisfy constraint R is denoted as $\mathbf{P}_R$. The set of all JPDs that satisfy all the constraints in $\mathbf{R}$ is denoted as $\mathbf{P}_\mathbf{R}$.

The problem of probabilistic knowledge integration is to modify the existing knowledge base $P(X)$ so that the resulting JPD $Q(X)$ satisfies all constraints in $\mathbf{R}$, i.e., $Q(X) \in \mathbf{P}_{\mathbf{R}}$. There are other desirable integration objectives, a commonly adopted one is to require that $Q(X)$, while satisfying $\mathbf{R}$, is as close to $P(X)$ as possible [18]. To measure the degree of changes to $P(X)$, we can use some distance metrics for distributions such as I-divergence, which is also known as K-L (Kullback–Leibler) distance as defined below [18, 43, 78].

Definition 2.2 (I-divergence) Given two probability distributions $P(X)$ and $Q(X)$, *I-divergence* from $P(X)$ to $Q(X)$ is defined as:

$$I(P(X) \,||\, Q(X)) = \begin{cases} \sum_{x \in X} P(x) \log \dfrac{P(x)}{Q(x)} & \text{if } P(X) \ll Q(X) \\ +\infty & \text{otherwise} \end{cases} \qquad \textbf{(2.1)}$$

where $P(X) \ll Q(X)$ means $Q(X)$ dominates $P(X)$, i.e. $\{X \mid P(X) > 0\} \subseteq \{X' \mid Q(X') > 0\}$.

Note that I-divergence is zero if and only if $P(X)$ and $Q(X)$ are identical. Also note that I-divergence is non-symmetric.

2.1.2 The Iterative Proportional Fitting Procedure

The problem of probabilistic knowledge integration can be solved with the Iterative Proportional Fitting Procedure (IPFP). IPFP is a mathematical procedure that iteratively modifies a JPD in order to find a distribution that satisfies a set of probability constraints while maintaining minimum I-divergence to the original distribution. It was first proposed by Kruithof in 1937 [42]. Later it was used as a procedure to estimate cell frequencies in contingency tables under marginal constraints [21]. Its convergence was

proved in [18, 27, 68, 78]. It was also extended as Conditional Iterative Proportional Fitting Procedure (CIPFP) to deal with conditional constraints [6, 16].

The core of IPFP is to calculate I-projection [76], which is defined as follows:

Definition 2.3 (I-projection) JPD $Q^*(X)$ is said to be an *I-projection* of JPD $P(X)$ on the set of JPDs $\mathbf{Q}(X)$ if for any $Q(X) \in \mathbf{Q}(X)$, $Q^*(X)$ has the minimum I-divergence with $P(X)$, i.e.,

$$I(Q^*(X) \,||\, P(X)) = \min_{Q \in \mathbf{Q}} I(Q(X) \,||\, P(X)) \tag{2.2}$$

It has been shown in [17] that the I-projection is unique. If $\mathbf{Q}(X)$ is the set of all JPDs that satisfies a set of constraints $\mathbf{R}$, i.e., $\mathbf{Q}(X) = \mathbf{P}_{\mathbf{R}}$, then $Q^*(X)$ is a JPD that satisfies all the constraints $\mathbf{R}$, and at the same time $Q^*(X)$ has the minimal I-divergence with $P(X)$ [78].

IPFP iteratively modifies the current JPD $Q_{k-1}(X)$, starting with $Q_0(X) = P(X)$, each time using one constraint $R_j(Y^j)$ in $\mathbf{R}$, according to the following formula:

$$Q_k(X) = \begin{cases} 0 & \text{if } Q_{k-1}(Y^j) = 0 \\ Q_{k-1}(X) \cdot \dfrac{R_j(Y^j)}{Q_{k-1}(Y^j)} & \text{otherwise} \end{cases} \tag{2.3}$$

where $j = k \bmod m$, which determines the constraint used at iteration k, and m is the number of constraints in $\mathbf{R}$. It has been shown that the resulting distribution $Q_k(X)$ is the I-projection of $Q_{k-1}(X)$ on $\mathbf{P}_{R_j}$ for $R_j(Y^j)$ [76].

Definition 2.4 (Consistency among Constraints) Let $\mathbf{R}$ be a set of constraints. If $\mathbf{P}_{\mathbf{R}} \neq \varnothing$, i.e., there is at least one JPD that satisfies $\mathbf{R}$, then we say constraints in $\mathbf{R}$ are

consistent with each other. Otherwise, we say constraints in $\mathbf{R}$ are *inconsistent* with each other.

IPFP will only converge when constraints are consistent with each other. In such situation, IPFP will converge to a JPD $Q^*(X)$, which is the I-projection of the initial JPD $Q_0(X) = P(X)$ on $\mathbf{P}_\mathbf{R}$, for the given constraints $\mathbf{R}$. That is, $Q^*(X)$ satisfies $\mathbf{R}$, and among all the JPDs that satisfy $\mathbf{R}$, $Q^*(X)$ has the minimal I-divergence with the initial JPD $Q_0(X)$. Csiszar [18] has provided a convergence proof for IPFP based on I-divergence geometry. When constraints are inconsistent with each other, we have $\mathbf{P}_\mathbf{R} = \varnothing$, and IPFP will not converge but oscillate among multiple JPDs [18, 63, 79].

2.1.3 Integrating Inconsistent Constraints

In real-world situations, since knowledge usually comes from different sources and is obtained by different means, inconsistency often exists among constraints [78, 86]. Vomlel proposed CC-IPFP and GEMA [78, 79] which extend IPFP to integrate constraints that are inconsistent with each other.

At each iteration of CC-IPFP, it first computes the I-projection of $Q_{k-1}(X)$ to $\mathbf{P}_{R_j}$, and then mixes the result with $Q_{k-1}(X)$ using the following formula to get $Q_k(X)$:

$$Q_k(X) = (1-\alpha_k)Q_{k-1}(X) + \alpha_k Q_{k-1}(X) \cdot \frac{R_j(Y^j)}{Q_{k-1}(Y^j)} \qquad \textbf{(2.4)}$$

where $j = k \bmod m$, which determines the constraint used at iteration k, and m is the number of constraints in $\mathbf{R}$. $\alpha_k = 1/(k+1)$, which monotonically decreases towards 0 when k increases. It can be seen from (2.4) that the process starts like the standard IPFP.

But the influence of constraints and the inconsistency among constraints are gradually diminished along with the iteration process.

The resulting JPD of CC-IPFP does not satisfy all the constraints. But it is as close to each of the constraints as possible, which can be measured by the total variation.

Definition 2.5 (Total Variation) Given two probability distributions $P(X)$ and $Q(X)$, the *total variation* between them is defined as:

$$\delta(P,Q)=\sum_{x\in X}|\,P(x)-Q(x)\,| \tag{2.5}$$

It has been proven that CC-IPFP would converge to a distribution $Q^*(X)$ such that, among all the distributions in $\mathbf{Q}(X)$, $Q^*(X)$ has the minimum sum of total variations for all the constraints, i.e.,

$$\sum_{j=1}^{m}\delta(Q^*(Y^j),R_j(Y^j))=\min_{Q\in\mathbf{Q}}\sum_{j=1}^{m}\delta(Q(Y^j),R_j(Y^j)) \tag{2.6}$$

In contrast to CC-IPFP, GEMA allows one to give different weights to the constraints. The weight can reflect the degree of trust for each of the constraints. At each iteration of GEMA, it first computes the I-projection of $Q_{k-1}(X)$ for $R_j(Y^j)$, then takes a weighted sum of the I-projections for all the constraints to get $Q_k^{'}(X)$:

$$Q_k^{'}(X)=\sum_{j=1}^{m}\omega_j Q_{k-1}(X)\cdot\frac{R_j(Y^j)}{Q_{k-1}(Y^j)} \tag{2.7}$$

where ω_j is the weight for $R_j(Y^j)$, $0<\omega_j<1$, and $\sum_{j=1}^{m}\omega_j=1$.

Then it computes a new set of constraints $\mathbf{R}^{'}=\{R_1^{'}(Y^1),\cdots,R_m^{'}(Y^m)\}$ from $Q_k^{'}(X)$ with $R_j^{'}(Y^j)=Q_k^{'}(Y^j)$, and performs a standard IPFP on $Q_{k-1}(X)$ with $\mathbf{R}^{'}$ to get $Q_k(X)$.

Definition 2.6 (I-aggregate) Let $Q(X)$ be a JPD, and $\mathbf{P} = \{P_1(Y^1), \cdots, P_m(Y^m)\}$ be a set of distributions, where $Y^j \subseteq X$. *I-aggregate* is the weighted sum of the I-divergence for each distribution $P_j(Y^j)$ in $\mathbf{P}$, i.e.,

$$\psi(\mathbf{P}, Q) = \sum_{j=1}^{m} \omega_j \cdot I(Q(Y^j) \,||\, P_j(Y^j)) \tag{2.8}$$

where ω_j is the weight for $P_j(Y^j)$, $0 < \omega_j < 1$, and $\sum_{j=1}^{m} \omega_j = 1$.

It has been proven that GEMA would converge to a JPD $Q^*(X)$, such that for any distribution $Q(X)$ in the set of JPDs $\mathbf{Q}(X)$, $Q^*(X)$ has the minimum I-aggregate for all the constraints in $\mathbf{R}$, i.e.,

$$\sum_{j=1}^{m} \omega_j \cdot I(Q^*(Y^j) \,||\, R_j(Y^j)) = \min_{Q \in \mathbf{Q}} \sum_{j=1}^{m} \omega_j \cdot I(Q(Y^j) \,||\, R_j(Y^j)) \tag{2.9}$$

Experiments in [86] have shown that CC-IPFP converges very slowly, and GEMA is very sensitive to the initial JPD and the constraints. To address the limitations of CC-IPFP and GEMA, Peng and Zhang proposed the SMOOTH algorithm [65, 86] for knowledge integration when constraints are inconsistent with each other. Compared to CC-IPFP and GEMA, SMOOTH is more efficient and stable, and is not sensitive to the initialization of the data. SMOOTH makes bi-directional modification at each iteration. It not only pulls the JPD closer to the constraints, but also pulls the constraints towards the JPD. In this way, the inconsistency among the constraints is gradually reduced or smoothened, which may lead to a faster convergence.

At each iteration of SMOOTH, the first step is to modify constraint $R_j(Y^j)$ with smoothing factor α using the following formula, where $0 < \alpha < 1$:

$$R_j^{'}(Y^j) = \alpha R_j(Y^j) + (1-\alpha) Q_{k-1}(Y^j) \quad \textbf{(2.10)}$$

In this step the updated constraint $R_j^{'}(Y^j)$ has incorporated a small portion of $Q_{k-1}(Y^j)$, which is the marginal distribution of the JPD that has been modified by all the constraints in **R** in proceeding sequence of revisions. This helps $R_j(Y^j)$ to pull itself closer to all the other constraints. Thus the inconsistency is reduced or smoothened among the constraints.

The second step is to compute the I-projection of $Q_{k-1}(X)$ to $R_j^{'}(Y^j)$:

$$Q_k(X) = Q_{k-1}(X) \cdot \frac{R_j^{'}(Y^j)}{Q_{k-1}(Y^j)} \quad \textbf{(2.11)}$$

These two steps are repeated for each of the constraints in **R** until the process converges. The resulting JPD satisfies all the updated constraints after their inconsistency is gradually smoothened. The convergence of SMOOTH for two constraints that are inconsistent with each other has been proved in [65].

2.2 Knowledge Integration with BN

In this section we will introduce Bayesian network and review the related methods for knowledge integration with a BN.

2.2.1 Bayesian Network

The JPD that is used to represent the knowledge base can become intractably large as the number of variables increases. For *n* discrete variables, their JPD requires $O(2^n)$ space to store all the probabilities for the knowledge base. Moreover, manipulating such huge full JPD is very challenging from the computational perspective. In contrast to JPDs, graphical models such as BNs can capture the conditional independence among the

variables and represent their interdependencies through the graph structure among the variables. Using such graphical models one can greatly reduce the space and time complexity of representing and reasoning with probabilistic knowledge in general and improve the efficiency of knowledge integration process in particular [13, 32, 60, 61]. These advantages make BNs very popular in representing probabilistic knowledge bases [3, 9, 28, 35].

Specifically, a BN G is a directed acyclic graph (DAG) which can represent a JPD $P(X)$ over a set of variables $X = (X_1, \cdots, X_n)$, where each node in the BN represents a variable in X, and an arc (also called *link* in this thesis) between two nodes represents the direct dependency between the variables the two nodes represent. A conditional probability table (CPT) is attached to each node to represent the strength of the dependencies. We use G_S to refer to the DAG, i.e., the network structure of G, and G_P to refer to the set of all the CPTs in G. A BN can be represented as $G = (G_S, G_P)$ where $G_S = \{(X_i, \pi_i)\}$, and $G_P = \{P(X_i \mid \pi_i)\}$, with π_i denoting the set of parents of X_i and an arc is drawn from each variable in π_i to X_i. In this thesis we assume $X_1, \cdots, X_n$ are in topological order according to G_S.

Definition 2.7 (Topological Order) Let $X_1, \cdots, X_n$ be all the nodes in the DAG G_S, where the subscripts $1, \cdots, n$ are the indexes of the nodes. $X_1, \cdots, X_n$ are said to be in *topological order* according to G_S if for every arc $X_i \to X_j$ in G_S, X_i has lower index than X_j, i.e., $i < j$.

With this ordering, we can express JPD $P(X)$ in the following way:

$$P(X) = P(X_1, \cdots, X_n) = P(X_1)P(X_2 \mid X_1) \cdots P(X_n \mid X_1, \cdots, X_{n-1}) \qquad \textbf{(2.12)}$$

Definition 2.8 (Local Markov Property) In a Bayesian network, a variable X_i is independent of all of its non-descendants N_i given its parents π_i, i.e.,

$$P(X_i \mid N_i, \pi_i) = P(X_i \mid \pi_i) \quad \textbf{(2.13)}$$

Based on the local Markov property, the JPD $P(X)$ of $G = (G_S, G_P)$ can be *factored* into a product of CPTs according to the BN structure G_S, i.e.,

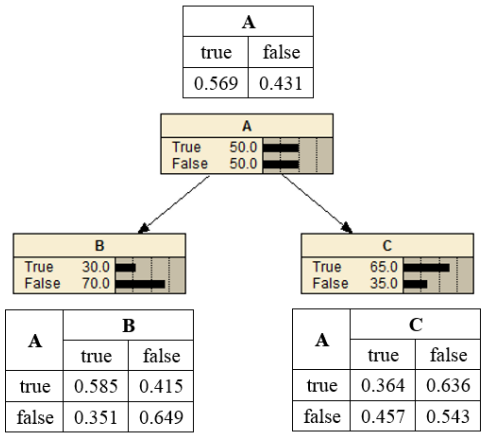
$$P(X_1, \cdots, X_n) = \prod_i P(X_i \mid X_1, \cdots, X_{i-1}) = \prod_i P(X_i \mid \pi_i) \quad \textbf{(2.14)}$$

This is called *the chain rule* of the BN and $P(X_i \mid \pi_i)$ is called a *factor* of the chain rule decomposition. We can also say $P(X_i \mid \pi_i)$ is *extracted* from $P(X)$ according to G_S if π_i is determined by G_S. We use $\mathbf{P}_{G_S}$ to denote the set of all JPDs that can be factored according to G_S, i.e., $\mathbf{P}_{G_S}$ contains all JPDs that have the same structure as G_S.

2.2.2 E-IPFP and D-IPFP

Is IPFP approach applicable to knowledge integration when the knowledge base is represented as a BN? One may suggest the following simple steps: 1) generate the JPD from the given BN, 2) integrate the given constraint into this JPD using IPFP, and 3) generate a modified BN from the resulting JPD from step 2) according to the original BN structure. However, as illustrated by the following example, this approach would not work. Consider a BN given in Figure 2.1 and constraint $R(B,C)$ in Figure 2.2. We run IPFP and get $Q_1(A,B,C)$ which is shown in Figure 2.3(a) as the resulting JPD. It can be seen that Q_1 satisfies $R(B,C)$. However, if we extract CPTs which are shown in Figure 2.3(b) from $Q_1(A,B,C)$ according to the BN structure, and generate the JPD from these

CPTs using the chain rule, we would get $Q_1'(A,B,C)$ shown in Figure 2.3(c). $Q_1'(A,B,C)$ is different from $Q_1(A,B,C)$ and it no longer satisfies $R(B,C)$.



A	B	C	$Q_0(A,B,C)$
true	true	true	0.120
true	true	false	0.080
true	false	true	0.180
true	false	false	0.120
false	true	true	0.070
false	true	false	0.030
false	false	true	0.280
false	false	false	0.120

A	B	
	true	false
true	0.585	0.415
false	0.351	0.649

A	C	
	true	false
true	0.364	0.636
false	0.457	0.543

Figure 2.1 A 3 node BN with its CPTs and JPD

B	C	$R(B,C)$
true	true	0.204
true	false	0.280
false	true	0.200
false	false	0.316

Figure 2.2 Constraint $R(B,C)$

A	B	C	$Q_1(A,B,C)$
true	true	true	0.129
true	true	false	0.204
true	false	true	0.078
true	false	false	0.158
false	true	true	0.075
false	true	false	0.076
false	false	true	0.122
false	false	false	0.158

(a) Resulting JPD that satisfies $R(B,C)$.

A	
true	false
0.569	0.431

A	B	
	true	false
true	0.585	0.415
false	0.351	0.649

A	C	
	true	false
true	0.364	0.636
false	0.457	0.543

(b) CPTs extracted from $Q_1(A,B,C)$.

A	B	C	$Q_1'(A,B,C)$
true	true	true	0.121
true	true	false	0.212
true	false	true	0.086
true	false	false	0.150
false	true	true	0.069
false	true	false	0.082
false	false	true	0.128
false	false	false	0.152

(c) JPD generated from the CPTs in (b) using the chain rule.

Figure 2.3 Result after running IPFP with the 3 node BN and $R(B,C)$

The problem stems from the fact that IPFP does not preserve the dependencies depicted by the BN structure when applied to modify the JPD of the BN. To solve this problem, Peng et al. proposed the E-IPFP algorithm [63] based on IPFP with an extension of the simple steps mentioned at the beginning of this subsection: when the JPD from step 3) is different from that of step 2), E-IPFP will continue to iterate to step 1). In a sense this is equivalent to take the BN and its JPD from step 3) as an additional constraint. Since this JPD obeys BN structure, it is called a *structural constraint*. Specifically, the structural constraint, denoted as R_{m+1}, is generated by first extracting CPTs from the current $Q_{k-1}(X)$ according to G_S, then getting the JPD from these CPTs using the chain rule, i.e.,

$$R_{m+1}(X) = \prod_{X_i \in X} Q_{k-1}(X_i \mid \pi_i) \tag{2.15}$$

In each round after iterating through the constraints $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$, $R_{m+1}(X)$ is applied to $Q_{k-1}(X)$ to force the JPD to satisfy G_S.

Convergence of E-IPFP can be determined by testing if the difference between $Q_k(X)$ and $Q_{k-1}(X)$ is below a given threshold. The following is the E-IPFP algorithm.

E-IPFP ($G = (G_S, G_P)$, $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$)

1. $Q_0(X) = \prod_i P(X_i \mid \pi_i)$, where $P(X_i \mid \pi_i) \in G_P$;

2. Starting with $k = 1$, repeat the following procedure until convergence:

   2.1. $j = ((k-1) \bmod (m+1)) + 1$;

   2.2. If $j < m+1$, $Q_k(X) = Q_{k-1}(X) \cdot \frac{R_j(Y^j)}{Q_{k-1}(Y^j)}$;

   Else, extract $Q_{k-1}(X_i \mid \pi_i)$ from $Q_{k-1}(X)$ according to G_S,

   $$Q_k(X) = \prod_{X_i \in X} Q_{k-1}(X_i \mid \pi_i);$$

   2.3. $k = k+1$;

3. Return $\tilde{G} = (G_S, \tilde{G}_P)$, with $\tilde{G}_P = \{Q_k(X_i \mid \pi_i)\}$.

Given the BN in Figure 2.1 and the constraint in Figure 2.2, E-IPFP converges after 134 iterations. The resulting CPTs and JPD are shown in Figure 2.4.

A	
true	false
0.568	0.432

A	B	
	true	false
true	0.593	0.407
false	0.341	0.659

A	C	
	true	false
true	0.463	0.537
false	0.326	0.674

(a) CPTs of the resulting BN.

A	B	C	$Q^*(A,B,C)$
true	true	true	0.156
true	true	false	0.181
true	false	true	0.107
true	false	false	0.124
false	true	true	0.048
false	true	false	0.099
false	false	true	0.093
false	false	false	0.192

(b)JPD of the resulting BN.

Figure 2.4 Resulting CPTs and JPD after running E-IPFP

As can be seen from Figure 2.4, the resulting JPD $Q^*(A,B,C)$ which is shown in Figure 2.4(b) satisfies $R(B,C)$. Also, because $Q^*(A,B,C)$ is the product of CPTs of the resulting BN which are shown in Figure 2.4(a), $Q^*(A,B,C)$ satisfies the BN structure.

One issue of E-IPFP is that it is computationally intractable for large BNs. In step 2.2 of E-IPFP, each entry in $Q_{k-1}(X)$ is checked against all the entries of $R_j(Y^j)$. This computational cost is in the order of $O(2^{|X|} \cdot 2^{|Y^i|})$, which grows exponentially with the number of variables in the BN. To address this issue, Peng et al. developed D-IPFP which decomposes a global E-IPFP problem into a set of local E-IPFP problems. Each of the local E-IPFP problems is performed on a subnet of the BN that only contains Y^j and their parents for each constraint $R_j(Y^j)$. This greatly improves the performance for knowledge integration with large BNs. The following is the D-IPFP algorithm.

D-IPFP ($G = (G_S, G_P)$, $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$)

1. $Q_0(X_i \mid \pi_i) = P(X_i \mid \pi_i)$ for all $P(X_i \mid \pi_i) \in G_P$;
2. Starting with $k = 1$, repeat the following procedure until convergence:
   - 2.1. $j = ((k-1) \bmod (m+1)) + 1$;
   - 2.2. $Q_k^{'}(Y^j \mid S^j) = Q_{k-1}(Y^j \mid S^j) \cdot \frac{R_j(Y^j)}{Q_{k-1}(Y^j)} \cdot \alpha_k$,

     where $S^j = (\bigcup_{X_i \in Y^j} \pi_i) \setminus Y^j$, and α_k is a normalization factor;
   - 2.3. $Q_k(X_i \mid \pi_i) = Q_k^{'}(X_i \mid \pi_i), \forall X_i \in Y^j$;
   - 2.4. $k = k + 1$;
3. Return $\tilde{G} = (G_S, \tilde{G}_P)$, with $\tilde{G}_P = \{Q_k(X_i \mid \pi_i)\}$.

2.2.3 E-IPFP-SMOOTH

E-IPFP only converges when constraints in **R** are consistent with each other and with the network structure G_S. Otherwise, it oscillates among several JPDs. To solve this issue, Peng et al. proposed the E-IPFP-SMOOTH algorithm which extends the SMOOTH algorithm to the situation where the knowledge base is represented as a BN [64].

The following example shows how E-IPFP-SMOOTH overcomes the inconsistency among the constraints, as well as the inconsistency between the constraint and the BN structure. This example uses the BN in Figure 2.1 and the constraints in Figure 2.5.

A	B	$R_1(A,B)$
true	true	0.204
true	false	0.280
false	true	0.200
false	false	0.316

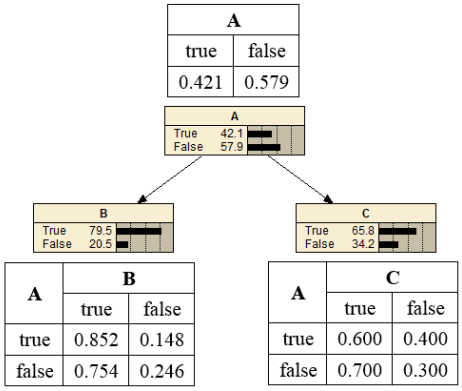
A	C	$R_2(A,C)$
true	true	0.360
true	false	0.060
false	true	0.440
false	false	0.140

A	B	C	$R_3(A,B,C)$
true	true	true	0.228
true	true	false	0.372
true	false	true	0.040
true	false	false	0.040
false	true	true	0.076
false	true	false	0.124
false	false	true	0.060
false	false	false	0.060

Figure 2.5 Constraints $R_1(A,B)$, $R_2(A,C)$ and $R_3(A,B,C)$

$R_1(A,B)$ and $R_2(A,C)$ are inconsistent with each other because $R_1(A) \neq R_2(A)$. $R_3(A,B,C)$ is inconsistent with the structure of the BN because $R_3(B,C \mid A) \neq R_3(B \mid A) \cdot R_3(C \mid A)$, which conflicts with the BN structure's assertion that *B* and *C* are conditionally independent given *A*. Figure 2.6(a) shows the resulting BN after running E-IPFP-SMOOTH with $R_1(A,B)$ and $R_2(A,C)$. Figure 2.6(b) shows the resulting BN after

running E-IPFP-SMOOTH with $R_3(A,B,C)$. The constraints in Figure 2.5 have been modified to $R_1^{'}(A,B)$, $R_2^{'}(A,C)$ and $R_3^{'}(A,B,C)$ as shown in Figure 2.7.

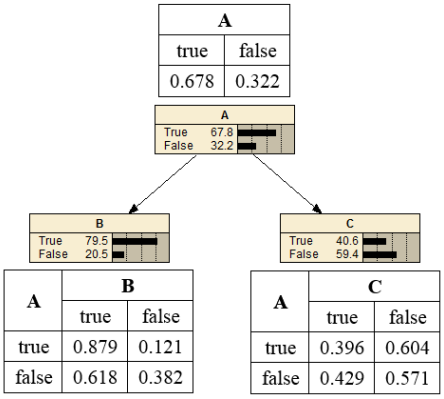


A	true	false
0.421	0.579	

A	B	
	true	false
true	0.852	0.148
false	0.754	0.246

A	C	
	true	false
true	0.600	0.400
false	0.700	0.300

(a) Resulting BN after integrating $R_1(A,B)$ and $R_2(A,C)$.



A	true	false
0.678	0.322	

A	B	
	true	false
true	0.879	0.121
false	0.618	0.382

A	C	
	true	false
true	0.396	0.604
false	0.429	0.571

(b) Resulting BN after integrating $R_3(A,B,C)$**.**

Figure 2.6 Resulting BN after running E-IPFP-SMOOTH

A	B	$R_1'(A,B)$
true	true	0.358
true	false	0.062
false	true	0.437
false	false	0.143

A	C	$R_2'(A,C)$
true	true	0.252
true	false	0.168
false	true	0.406
false	false	0.174

A	B	C	$R_3'(A,B,C)$
true	true	true	0.236
true	true	false	0.360
true	false	true	0.032
true	false	false	0.050
false	true	true	0.085
false	true	false	0.114
false	false	true	0.053
false	false	false	0.070

Figure 2.7 Constraints $R_1'(A,B)$, $R_2'(A,C)$ and $R_3'(A,B,C)$

We can see from the above examples that E-IPFP-SMOOTH overcomes both kinds of inconsistencies by modifying the constraints. For the inconsistency among the constraints, a compromise is reached at the end for all the constraints. The algorithm can be easily modified to accommodate weights that reflect levels of trust or confidence for each constraint. For the inconsistency between the constraint and the BN structure, E-IPFP-SMOOTH modifies the dependencies in the constraint to align with the BN structure while keeping the BN structure unchanged. In this example, after running E-IPFP-SMOOTH with $R_3(A,B,C)$, $R_3'(A,B,C)$ is consistent with the BN structure because $R_3'(B,C \mid A) = R_3'(B \mid A) \cdot R_3'(C \mid A)$. That is, B and C are no longer conditionally dependent given A as in the original constraint $R_3(A,B,C)$. In other words, the dependency relations brought by the constraint is not integrated into the BN. Therefore, this method can only be used in applications where it is desirable to keep the dependency relations given in the BN intact during the integration, but not applicable when it is desirable for the BN to adapt to the new dependencies given in the constraints.

2.3 Virtual Evidence Method

Pearl proposed virtual evidence method [57] for BN reasoning with virtual evidence. BN reasoning is an inference process for calculating posterior probabilities given evidence. Posterior probabilities are also called *beliefs* in Bayesian networks. Thus, BN reasoning is also referred to as BN belief update [2, 12, 51].

2.3.1 Soft Evidence and Virtual Evidence

Evidence is a collection of findings. Researchers have identified three kinds of evidences in BN belief update: *hard evidence*, *virtual evidence* and *soft evidence*.

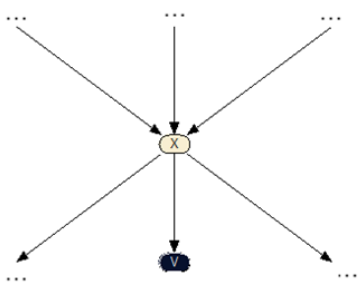
To explain each kind of evidence, we use variable X to represent a specific event. If the event occurs, X is in the true state. If the event does not occur, X is in the false state. Each kind of evidence specifies the state X is in from a different perspective.

Hard evidence specifies the particular state variable X is in. For example, "the event represented by variable X has occurred" is a hard evidence.

Soft evidence [76] specifies the probability distribution for the states variable X is in. It is the *evidence of uncertainty*. This type of evidence is used when one is uncertain about the state X is in, but is certain about the distribution $P(X)$ for the states variable X is in. $P(X)$ is based on the true observations and can be treated as a hard evidence. For example, "the event represented by variable X has been observed to occur and not occur with distribution $P(X) = (0.45, 0.55)$ " is a soft evidence. $P(X) = (0.45, 0.55)$ is a true observation and itself can be considered a hard evidence. That is, when a soft evidence is applied, its accompanied $P(X)$ should be accepted by the belief system.

Virtual evidence [57] specifies the *likelihood ratio* for the state variable X is claimed to be in. It is the *evidence with uncertainty*. Suppose one believes with probability p about the claim that the event represented by X occurs, and does not occur with probability $1-p$. Then the likelihood ratio can be represented as $L(X) = p:(1-p)$, which does not necessarily need to be a specific probability. For example, "the event represented by variable X has been observed to occur, but there is only 45% chance that this observation is true" is a virtual evidence with likelihood ratio $L(X) = 0.45:0.55$.

Virtual evidence and soft evidence are two types of uncertain evidences [4]. Each of them has its own characteristics and obeys a different belief update rule. Pearl proposed virtual evidence method for BN belief update with virtual evidence by extending the given BN with a binary *virtual* node [57]. Figure 2.8 shows how the method works for virtual evidence $L(X) = 0.45:0.55$. It first creates a virtual node V with X as its only parent in the BN, with state v standing for the event that $X = x$ has occurred. Then it sets up the CPT of V which satisfies $P(V = v \mid X = x) : P(V = v \mid X \neq x) = L(X)$. The CPT can contain any specific probability entries that satisfy this likelihood ratio requirement. In Figure 2.8 the CPT of V satisfies this requirement because $0.63:0.77 = 0.45:0.55$. At the end the method treats v as a hard evidence by instantiating V to v. This will update the belief in the given BN, and the updated BN will satisfy the given virtual evidence.



X	V	
	$V = v$	$V \neq v$
$X = x$	0.63	0.37
$X \neq x$	0.77	0.23

Figure 2.8 Virtual node V created for virtual evidence $L(X) = 0.45:0.55$

2.3.2 Converting Soft Evidence into Virtual Evidence

For belief update with soft evidence, Jeffrey proposed Jeffrey's rule [36] to update a joint distribution $P(X)$ with a soft evidence $Q(Y)$, where $Y \subseteq X$. Jeffrey's rule assumes that the conditional probability of all the other variables given Y should not change, i.e.,

$$Q(X \setminus Y \mid Y) = P(X \setminus Y \mid Y) \tag{2.16}$$

where $X \setminus Y$ refers to all the variables in X but not in Y.

Based on Jeffrey's rule, we have:

$$Q(X) = P(X \setminus Y \mid Y)Q(Y) = \frac{P(X,Y)}{P(Y)}Q(Y) = \frac{P(X)}{P(Y)}Q(Y) \tag{2.17}$$

Thus the distribution $P(X)$ under the observation $Q(Y)$ should be updated to:

$$Q(X) = P(X)\frac{Q(Y)}{P(Y)} \tag{2.18}$$

It has been proven that after updating $P(X)$ by $Q(Y)$ using Jeffery's rule in (2.18), the resulting distribution $Q(X)$ is the same as the I-projection of $P(X)$ on $\mathbf{P}_{Q(Y)}$ by (2.3) [66]. Jeffrey's rule guarantees that the updated distribution can satisfy the evidence while making minimum changes to the original distribution. However, it cannot be applied directly to a joint distribution represented as a BN. Chan and Darwiche solved this problem by first converting a soft evidence to a virtual evidence, then applying Pearl's virtual evidence method to realize the belief update in the BN with a soft evidence [12].

Let $P(X)$ be a distribution and $Q(Y)$ be a soft evidence, where $Y \subseteq X$, and $\mathbf{Y}$ be all the possible instantiations of Y such that $y_{(1)}, y_{(2)}, \cdots, y_{(m)} \in \mathbf{Y}$ form a mutually exclusive and exhaustive set of events. Then $Q(Y)$ can be converted to a virtual evidence with the following likelihood ratio:

$$L(Y) = \frac{Q(y_{(1)})}{P(y_{(1)})} : \frac{Q(y_{(2)})}{P(y_{(2)})} : \cdots : \frac{Q(y_{(m)})}{P(y_{(m)})} \tag{2.19}$$

To set up the CPT for the virtual node *V* using this likelihood ratio, we use the following formula in this thesis to calculate its entries when Y takes the instantiation of $y_{(i)}$:

$$\begin{aligned} P(V = v \mid y_{(i)}) &= \frac{Q(y_{(i)})}{P(y_{(i)})} \Big/ \sum_{i=1}^{m} \frac{Q(y_{(i)})}{P(y_{(i)})} \\ P(V \neq v \mid y_{(i)}) &= 1 - \left(\frac{Q(y_{(i)})}{P(y_{(i)})} \Big/ \sum_{i=1}^{m} \frac{Q(y_{(i)})}{P(y_{(i)})} \right) \end{aligned} \tag{2.20}$$

When this virtual evidence is applied to $P(X)$ by instantiating *V* to *v*, the updated distribution is the same as what is obtained using Jeffrey's rule after applying $Q(Y)$ [12].

2.4 BN-IPFP

For multiple virtual evidences, the BN belief update by one virtual evidence will not affect the belief update by another virtual evidence. This is because each of the virtual evidences is treated as a hard evidence on the virtual node, which is instantiated to true. The likelihood ratio on *V*, which is determined by the virtual evidence itself and reflected in the CPT, will not be affected by the belief update caused by the instantiation of other virtual nodes.

The situation is different for BN belief update with multiple soft evidences. After satisfying one soft evidence, the updated distribution may not satisfy the other soft evidences. For example, given two soft evidences $se1 = Q(Y^1)$ and $se2 = Q(Y^2)$, suppose we first convert $se1$ and $se2$ to two virtual evidences and then apply the virtual evidence

method with these two virtual evidences. After applying $Q(Y^1)$, the revised JPD would satisfy $se1$ (i.e., its marginal on Y^1 equals $Q(Y^1)$). After applying $Q(Y^2)$, the revised JPD would satisfy $se2$, but there is no way to guarantee it still satisfies $se1$. This is also true when the soft evidences are applied in different orders or applied at the same time [54, 55, 65, 80].

To solve this problem, Peng et al. proposed BN-IPFP algorithms which combine Pearl's virtual evidence method with IPFP for BN belief update with multiple soft evidences. There are three versions of BN-IPFP algorithms: BN-IPFP-1, BN-IPFP-2 and BN-IPFP-SMOOTH. BN-IPFP-1 uses one soft evidence at each iteration. If the marginal of the current distribution $P(Y)$ does not satisfy this soft evidence $Q(Y)$, BN-IPFP-1 first converts this soft evidence to a virtual evidence according to (2.19), then updates the distribution with this virtual evidence using virtual evidence method. The iteration continues until the process converges. The following is the BN-IPFP-1 algorithm.

BN-IPFP-1 ($G = (G_S, G_P)$, $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$)

1. Let $Q_0(X)$ denote $P(X)$, the joint distribution of the given BN;
2. Starting with $k = 1$, repeat the following steps until convergence:

   2.1. $j = 1 + (k-1) \bmod m$; $l = 1 + |(k-1)/m|$;

   2.2. Construct virtual evidence with likelihood ratio

   $$L_{j,l}(Y^j) = \frac{R(y^j_{(1)})}{Q_{k-1}(y^j_{(1)})} : \frac{R(y^j_{(2)})}{Q_{k-1}(y^j_{(2)})} : \cdots : \frac{R(y^j_{(j_s)})}{Q_{k-1}(y^j_{(j_s)})}$$

   where $y^j_{(1)}, y^j_{(2)}, \cdots, y^j_{(j_s)} \in \mathbf{Y}^j$ are the state configurations of Y^j ;

   2.3. Update the BN with $L_{j,l}(Y^j)$ using virtual evidence method, and let $Q_k(X)$ denote the distribution of the updated BN;

   2.4. $k = k + 1$;

BN-IPFP-2 first merges all the soft evidences into a single soft evidence using IPFP, then converts this merged soft evidence into a virtual evidence, and updates the entire BN with it using the virtual evidence method. The following is the BN-IPFP-2 algorithm.

BN-IPFP-2 ($G = (G_S, G_P)$, $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$)

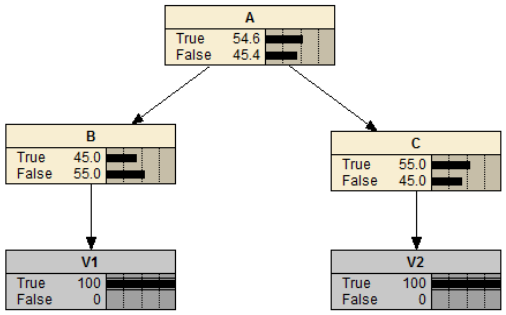
1. Use any BN inference method to obtain $P(Y)$ from $P(X)$, where $Y = Y^1 \cup \cdots \cup Y^m$, and $P(X)$ is the joint distribution of the given BN;
2. Update $P(Y)$ by IPFP using $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$ as constraints until converging to $Q^*(Y)$;
3. Construct a virtual evidence with likelihood ratio $L(Y)$ computed from $Q^*(Y)$ and $P(Y)$ by (2.19);
4. Apply $L(Y)$ as a single virtual evidence to update $P(X)$.

Given a set of soft evidences that are consistent with each other, these two algorithms converge to the same distribution, which can satisfy all the soft evidences and also has minimum I-divergence with the original distribution. In comparison, BN-IPFP-1 is more suitable for small BNs with a large number of variables in the soft evidences. This is because at each iteration it computes the marginal distribution for variables in the soft evidence and updates the belief for the entire BN. BN-IPFP-2 works better for large BNs with a small number of variables in the soft evidences because it only updates the joint distribution of all those variables. In this case it is more efficient than BN-IPFP-1 because by merging the soft evidences first, it avoids repeated computations with the large BN.

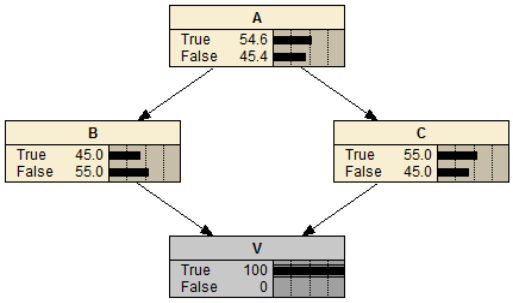
BN-IPFP-SMOOTH was proposed for multiple soft evidences that are inconsistent with each other. It incorporates SMOOTH into BN-IPFP and smoothens the inconsistencies by modifying the soft evidences.

To get a better understanding of BN-IPFP-1 and BN-IPFP-2, we show the result of applying them to the BN in Figure 2.1 with two soft evidences: $R(B) = (0.45, 0.55)$

and $R(C) = (0.55, 0.45)$. Figure 2.9(a) shows the resulting BN after applying BN-IPFP-1, where two virtual nodes are added to the given BN, one for each soft evidence. In Figure 2.9(b), only one virtual node is added to the given BN after applying BN-IPFP-2. We can see that in both cases the two soft evidences can be satisfied simultaneously. Also, as expected, the two algorithms converge to the same distribution at the end.



(a) Resulting BN of BN-IPFP-1.



(b) Resulting BN of BN-IPFP-2.

Figure 2.9 Resulting BNs of BN-IPFP-1 and BN-IPFP-2

2.5 BN Structure Learning

In this section we briefly review the BN structure learning methods that target problems similar to those our AddLink methods are set to solve, i.e., finding the set of links between the variables to form a DAG that can best represent the given data. After

introducing the existing methods for BN structure learning, we list the reasons why these methods are not fit for the specific problem we set to solve.

There are mainly two approaches to learn BN structures from raw data, one is score-based and the other is constraint-based [40]. The score-based approach first defines a scoring function to evaluate how well a DAG matches the data, then searches for a DAG that maximizes the score. An example for this approach is the K2 algorithm [15]. It defines the following Bayesian scoring function:

$$\prod_{i=1}^{n}\prod_{j=1}^{q_i}\frac{(r_i-1)!}{(N_{ij}+r_i-1)!}\prod_{k=1}^{r_i}\alpha_{ijk}! \qquad \textbf{(2.21)}$$

where n is the number of variables, q_i is the number of all possible configurations of the parents of variable X_i, r_i is the number of states of variable X_i, α_{ijk} is the number of instances in the data where variable X_i takes its k^{th} value and the parents of X_i take the j^{th} configuration, and N_{ij} is the number of instances in the data where the parents of variable X_i take the j^{th} configuration. This score is decomposable [15] and the value at variable X_i can be calculated using the following function:

$$f(i,\pi_i)=\prod_{j=1}^{q_i}\frac{(r_i-1)!}{(N_{ij}+r_i-1)!}\prod_{k=1}^{r_i}\alpha_{ijk}! \qquad \textbf{(2.22)}$$

where π_i are the parents of X_i. The informal intuition is that $f(i,\pi_i)$ is the probability of the data given that the parents of X_i are π_i. To find a DAG that maximizes the score, the K2 algorithm performs a hill-climbing search over the possible DAGs by adding or removing a link at each step according to a localized scoring function decomposed from (2.22) and a pre-ordering of the variables that sets their topological relations. The process

stops when no link change can increase the score. Other score-based methods may use a different scoring function such as BDe (Bayesian Dirichlet equivalence) score [20, 31] or MDL (Minimum Description Length) score [44, 73], or adopt a different search strategy such as heuristic search [34, 69, 85] or simulated annealing [30, 33, 53]. All these methods have the common goal of finding a DAG with the highest score so it can best represent the given data.

In contrast, the constraint-based approach first applies statistical tests to the data to discover the interdependencies among the variables, then builds the skeleton of the DAG with undirected edges, and finally determines their directions. An example of this approach is IC (Inductive Causation) algorithm [77]. For each pair of variables X_i and X_j, it searches for a set of variables S_{ij} such that X_i and X_j are independent given S_{ij} based on the result of the statistical test. If S_{ij} is not found, an undirected edge is added between X_i and X_j. In the next step, for each pair of non-adjacent nodes X_i and X_j with a common neighbor X_k such that $X_k \notin S_{ij}$, it creates a V-structure of the form $X_i \rightarrow X_k \leftarrow X_j$. In the last step, it tries to orient as many of the undirected edges as possible as long as no new V-structure or a directed cycle is created. Other constraint-based methods such as the PC (Peter and Clark) algorithm [70] or GS (Grow-Shrink) algorithm [49] go through the similar phases of first identifying the skeleton of the DAG based on the statistical results and then orienting as many undirected edges as they can.

When the DAG already exists, new data can be used to improve the existing DAG so that the refined DAG can more accurately reflect the structure of the new data. Buntine proposed a Bayesian score based approach to update the existing BN with new data [10].

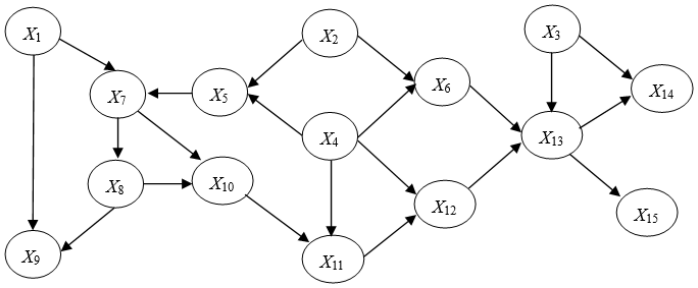
It aims to maximize the posterior probabilities of the updated BN with new data. Wai Lam proposed a more flexible approach to update the BN structure with new data based on MDL. This method is also able to preserve the existing DAG to various degrees [45]. To reduce the computational cost, its update scope for adding, removing, or reversing links is limited to be among the nodes covered by new data and the parents of these nodes.

As can be seen from the above examples, the focus of the BN structure learning methods is about determining which links are the most appropriate to be added to the BN given the data. It is similar to the concern we have for overcoming structural inconsistencies by adding links. However, the BN structure learning methods cannot be used directly to solve our problem for the following reasons. First, the data in our problem is a list of probabilistic constraints, which need to be completely satisfied instead of being satisfied to the maximum degree. The score-based BN structure learning methods operate on raw data, which is usually not possible to be completely satisfied. Besides, the instances of learning data for these structure learning methods are complete instantiations of all variables in the domain while in our problem each constraint is a low dimensional distribution which only involves a small subset of these variables. Therefore, the existing methods for BN structure learning are not fit for the problem of overcoming structural inconsistencies by adding links.

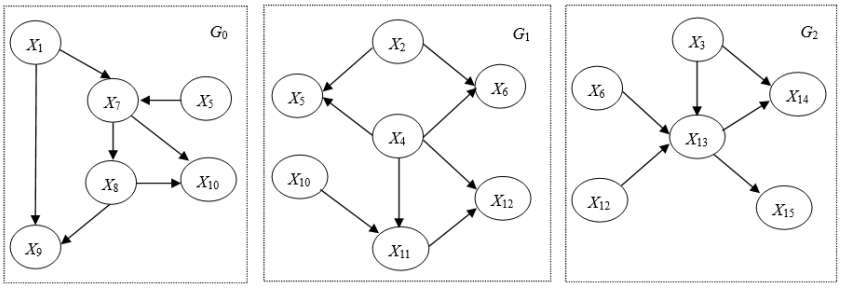
2.6 MSBN

In this section we briefly review MSBN (Multiply Sectioned BN) proposed by Xiang [81, 82, 83, 84] because it is related to the problem of constructing a large BN from a set of small BNs to which we hope our methods developed in this thesis can be applied. MSBN

sections a large BN representing an uncertain domain into multiple small BNs, and deploys each of them to an agent which is in charge of a subdomain. Each agent reasons about the state of its subdomain based on its partial knowledge of the entire domain together with its local observations, and communicates with other agents to estimate the state of the entire domain. Figure 2.10 shows an example of a BN G being sectioned into three smaller BNs, G_0, G_1 and G_2, each of which can be deployed to an agent.



(a) A 15 node BN G .



(b) Three BNs, G_0, G_1, and G_2 sectioned from G .

Figure 2.10 A 15 node BN G is sectioned into three BNs G_0, G_1, and G_2

To accomplish the task of estimating the state of the entire uncertain domain with limited amount of communication, MSBN enforces two key technical constraints: hypertree and d-sepset interface [81].

Definition 2.9 (Hypertree) Let $G = (V, E)$ be a connected graph with V being the set of all of its nodes and E being the set of all of its links. Let $\{G_i = (V_i, E_i)\}$ be a set of

subgraphs sectioned from G . Let the subgraphs be organized into an undirected tree Ψ where each node is uniquely labeled by G_i and each link between G_k and G_m is labeled by the non-empty *interface* $V_k \cap V_m$ such that for each i and j , $V_i \cap V_j$ is contained in each subgraph on the path between G_i and G_j in Ψ . Then Ψ is a *hypertree* over G. Each G_i is a *hypernode* and each interface is a *hyperlink*.

The hypertree represents how agents communicate with each other, where variables in each hyperlink are shared by agents. Figure 2.11 illustrates the hypertree for G in Figure 2.10(a). X_5 and X_{10} are shared by G_0 and G_1. X_6 and X_{12} are shared by G_1 and G_2.

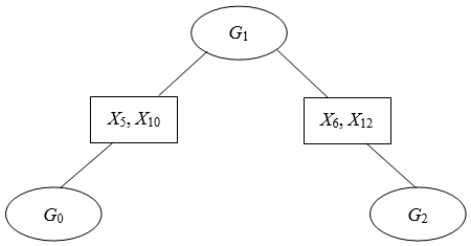


Figure 2.11 The hypertree for graph G in Figure 2.10(a)

Definition 2.10 (d-sepset) Let G be a directed graph such that a hypertree over G exists. A node X_i contained in more than one subgraph with its parents π_i in G is a *d-sepnode* if there exists at least one subgraph that contains π_i. An interface I is a *d-sepset* if every $X_i \in I$ is a d-sepnode.

In MSBNs agents communicate by exchanging their beliefs over shared variables. Each shared variable needs to be a d-sepset. The interface between G_0 and G_1 in Figure 2.11 is a d-sepset because X_5 and X_{10} are only shared by G_0 and G_1, and X_5 has all its parents contained in G_1 while X_{10} has all its parents contained in G_0. For the same reason, the interface between G_1 and G_2 in Figure 2.11 is also a d-sepset.

Definition 2.11 (MSDAG) A hypertree *MSDAG* $G = \bigcup_i G_i$, where each G_i is a DAG, is a connected DAG such that (1) there exists a hypertree over G, and (2) each hyperlink in Ψ is a d-sepset.

The structure of an MSBN needs to be an MSDAG with a hypertree organization. Graph G in Figure 2.10(a) is an MSDAG because it satisfies the two requirements.

The hypertree organization guarantees the effective communication among the small BNs, so they can estimate the state of the entire domain through cooperation. But its strict restrictions on the shared variables limit the application of MSBN when the subnets are not obtained by sectioning a large BN but directly from individual agents' modeling of their respective subdomains. In particular, the requirement for the shared variables to be identical can be reasonable only if all the agents model their part of the world based on the same conceptualization of the domain and have accurate information. The requirement for all parents of a shared variable to appear in one subnet is hard to meet when the agents have different access levels or different scopes of information. Therefore, when structural restrictions need to be relaxed, MSBN is not suitable for modeling the overall uncertain knowledge of a large distributed domain represented as a set of BNs.

2.7 Summary

In this chapter we introduced background knowledge that our research is based on, including the problem of probabilistic knowledge integration, IPFP, and Bayesian networks. We also reviewed the existing methods including CC-IPFP, GEMA and SMOOTH for knowledge integration with JPD when constraints are inconsistent with

each other. These IPFP-based methods cannot be applied directly to the BNs since operations of IPFP are defined over the full JPD, not the BNs.

For knowledge integration with BNs, we reviewed E-IPFP which extends IPFP by adding a structural constraint during the iteration to force the result to comply with the BN structure. We also reviewed D-IPFP which reduces the computational cost for large BNs by decomposing a global E-IPFP problem into a set of smaller local E-IPFP problems. Lastly, we reviewed E-IPFP-SMOOTH which incorporates SMOOTH into E-IPFP to resolve the situation where the constraints are inconsistent with each other or with the BN structure. In the same manner as E-IPFP and D-IPFP, E-IPFP-SMOOTH keeps BN's structure intact and smoothens the inconsistencies by modifying the constraints during the integration process.

We also introduced virtual evidence method because it is closely related to our AddNode methods for overcoming structural inconsistencies. Virtual evidence and soft evidence are two types of uncertain evidences, each of which has its own characteristics and obeys a different belief update rule. BN belief update with virtual evidence can be achieved by Pearl's virtual evidence method which adds a virtual node to the BN. BN belief update with soft evidence can be accomplished by first converting the soft evidence into a virtual evidence and then applying virtual evidence method.

Belief update with multiple virtual evidences can be carried out directly because the update for one virtual evidence does not affect the update for another virtual evidence. However, this is not the case for belief update with multiple soft evidences because the update for one soft evidence affects the update for the other soft evidences. We reviewed

BN-IPFP algorithms which combine Pearl's virtual evidence method with IPFP to overcome the challenge.

We provided a brief review for BN structure learning methods because our AddLink methods for overcoming structural inconsistencies have similar concern as to which links to be added to the BN based on the data. We listed the differences between our problem and the problem of BN structure learning, and explained why the methods for BN structure learning cannot be applied directly to our problem.

At the end of this chapter we reviewed MSBN because it is related to the problem we will investigate, namely the problem of constructing a large BN from a set of small BNs. MSBN sections a large BN into multiple small BNs, and estimates the state of the entire domain with limited amount of communication among the small BNs. MSBN imposes strict restrictions on the structure of the small BNs, making it hard to be applied to problems such as merging small BNs into a large BN when the structural restrictions need to be relaxed.

3 Identify Structural Inconsistencies

When the knowledge base is represented as a BN, structural inconsistency can occur during knowledge integration when the probabilistic dependencies among some of the variables in a constraint disagree with the dependencies represented by the structure of the BN. In this chapter we take a closer look at this issue and investigate how to identify the structural inconsistencies between a BN and a constraint. This chapter is organized as follows:

In Section 3.1 we establish the theorem that any dependency that is implied by the constraint but does not hold in the BN structure will cause structural inconsistencies. This leads to our definition for structurally inconsistent constraint. Based on this definition, structural inconsistencies can be easily distinguished from other types of inconsistencies and be properly targeted during the knowledge integration process.

In Section 3.2 we develop a method called InconsId to identify structural inconsistencies between a BN and a set of constraints. This is accomplished by first extracting dependency information from each of the constraints, then checking the BN to see if any of those dependencies are not captured by its structure. An example is provided to show how this method works.

In Section 3.3 we analyze the time complexity of the InconsId method. The factors that could affect the execution time of the algorithm include the size of the BN, the number of constraints, and the number of variables in each constraint. Experiments are conducted to evaluate the execution time of the algorithm when these factors take different values.

3.1 Structural Inconsistencies

As we know the conditional independencies among variables in a probability distribution $P(X)$ can be encoded by the DAG G_S when $P(X)$ is represented as a BN $G = (G_S, G_P)$. In this section we will analyze how the independencies encoded in a DAG G_S and the independencies exist in a probability distribution $P(X)$ affect the relation between G_S and $P(X)$ when they cover the same set of variables X. This analysis will help us to specify the requirement for the constraint R with which R can be integrated with a BN $G = (G_S, G_P)$. This will lead to our definition for structurally inconsistent constraint, and further help to articulate the problem we set to solve in this thesis.

Given a probability distribution $P(X)$ and a DAG G_S over the same set of variables X, we use $I(U,V,W)_P$ to denote that U and V are independent given W in $P(X)$, and we use $I(U,V,W)_{G_S}$ to denote that U and V are independent given W in G_S, where U, V and W are disjoint subsets of X. The connection between G_S and $P(X)$ can be made through the following definitions [29, 56, 59]:

Definition 3.1 (I-Map) If every independency implied by G_S holds in $P(X)$, then we say G_S is an *I-Map* of $P(X)$. That is, $I(U,V,W)_{G_S} \Rightarrow I(U,V,W)_P$.

Definition 3.2 (D-Map) If every independency implied by $P(X)$ holds in G_S, then we say G_S is a *D-Map* of $P(X)$. That is, $I(U,V,W)_P \Rightarrow I(U,V,W)_{G_S}$.

Definition 3.3 (P-Map) If G_S is both an I-Map and a D-Map of $P(X)$, then we say G_S is a *Perfect Map*, or *P-Map* of $P(X)$. That is, $I(U,V,W)_P \Leftrightarrow I(U,V,W)_{G_S}$.

Definition 3.4 (Minimal I-Map) The *minimal I-Map* of $P(X)$ is a DAG G_S such that removing any arc from G_S introduces independencies that do not hold in $P(X)$.

Nir Friedman et al. defined I-Map from a different perspective. According to [39], a DAG is said to be an I-Map of a probability distribution if the probability distribution satisfies all the local dependencies associated with the DAG. Based on this definition, Nir Friedman et al. have proposed and proved that a DAG is an I-Map of a probability distribution if and only if the probability distribution can be factored according to that DAG [39, 91]. Here we reproduce this theorem and its proof using our notational conventions.

Theorem 3.1 G_S is an I-Map of $P(X)$ if and only if $P(X)$ can be factored according to G_S, i.e., $P(X) = \prod_i P(X_i \mid \pi_i)$, where π_i is determined by G_S.

Proof:

[Sufficiency] Let π_i be the parents of X_i, D_i be the descendants of X_i, and N_i be the non-descendants of X_i. Then $\{X_1, \cdots, X_n\} = \{X_i\} \cup \pi_i \cup D_i \cup N_i$. We have:

$$P(X_i \mid N_i, \pi_i) = \frac{P(X_i, N_i, \pi_i)}{\sum_{X_i} P(X_i, N_i, \pi_i)},$$

where

$$P(X_i, N_i, \pi_i) = \sum_{D_i} P(X_i, N_i, \pi_i, D_i) = \sum_{D_i} \prod_{j=1}^{n} P(X_j \mid \pi_j)$$

$$= P(X_i \mid \pi_i) \prod_{X_j \in N_i} P(X_j \mid \pi_j) \prod_{X_k \in \pi_i} P(X_k \mid \pi_k) \sum_{D_i} \prod_{X_l \in D_i} P(X_l \mid \pi_l).$$

Since $\sum_{D_i} \prod_{X_l \in D_i} P(X_l \mid \pi_l) = 1$, we have:

$$P(X_i, N_i, \pi_i) = P(X_i \mid \pi_i) \prod_{X_j \in N_i} P(X_j \mid \pi_j) \prod_{X_k \in \pi_i} P(X_k \mid \pi_k).$$

By summing over all the instantiations of X_i, we have:

$$\sum_{X_i} P(X_i, N_i, \pi_i) = \sum_{X_i} P(X_i \mid \pi_i) \prod_{X_j \in N_i} P(X_j \mid \pi_j) \prod_{X_k \in \pi_i} P(X_k \mid \pi_k)$$

$$= \prod_{X_j \in N_i} P(X_j \mid \pi_j) \prod_{X_k \in \pi_i} P(X_k \mid \pi_k) \sum_{X_i} P(X_i \mid \pi_i).$$

Since $\sum_{X_i} P(X_i \mid \pi_i) = 1$, we have:

$$\sum_{X_i} P(X_i, N_i, \pi_i) = \prod_{X_j \in N_i} P(X_j \mid \pi_j) \prod_{X_k \in \pi_i} P(X_k \mid \pi_k).$$

Therefore,

$$P(X_i \mid N_i, \pi_i) = \frac{P(X_i, N_i, \pi_i)}{\sum_{X_i} P(X_i, N_i, \pi_i)} = \frac{P(X_i \mid \pi_i) \prod_{X_j \in N_i} P(X_j \mid \pi_j) \prod_{X_k \in \pi_i} P(X_k \mid \pi_k)}{\prod_{X_j \in N_i} P(X_j \mid \pi_j) \prod_{X_k \in \pi_i} P(X_k \mid \pi_k)} = P(X_i \mid \pi_i).$$

It means X_i is independent of any of its non-descendants given its parents. Thus G_S is an I-Map of P.

[Necessity] Assume, without loss of generality, $X_1, \cdots, X_n$ are in topological order according to G_S. By the chain rule we have:

$$P(X_1, \cdots, X_n) = \prod_{i=1}^{n} P(X_i \mid X_1, \cdots, X_{i-1}).$$

Now consider one of the factors $P(X_i \mid X_1, \cdots, X_{i-1})$. Let π_i be the parents of X_i. Since $X_1, \cdots, X_n$ are in topological order according to G_S, we have $\pi_i \subseteq \{X_1, \cdots, X_{i-1}\}$. Furthermore, none of X_i's descendants can possibly be in the set $\{X_1, \cdots, X_{i-1}\}$. Besides the parents of X_i, all the other variables in $X_1, \cdots, X_{i-1}$ are the non-descendants of X_i,

which is denoted as Z_i. We have $\{X_1, \cdots, X_{i-1}\} = \pi_i \cup Z_i$. Since G_S is an I-Map of $P(X)$, according to the local Markov property of (2.13), X_i is independent of any of its non-descendants given its parents. So we have:

$$P(X_i \mid X_1, \cdots, X_{i-1}) = P(X_i \mid Z_i, \pi_i) = P(X_i \mid \pi_i).$$

Applying this transformation to all the factors in the chain rule decomposition, the conclusion follows. □

Besides, since every dependency implied by $P(X)$ holds in G_S if and only if every independency implied by G_S holds in $P(X)$, we have the following theorem:

Theorem 3.2 Let $\mathbf{P}_{G_S}$ denotes the set of all JPDs that can be factored according to G_S. $P(X) \in \mathbf{P}_{G_S}$ if and only if every dependency implied by $P(X)$ holds in G_S.

It is interesting to note from Theorem 3.2 that, $\forall P(X) \in \mathbf{P}_{G_S}$, the dependencies implied by G_S do not necessarily hold in $P(X)$. That means G_S can have more dependencies than $P(X)$, but not less.

Next we are going to specify the requirement for the constraint R with which R can be integrated with a BN $G = (G_S, G_P)$. From Chapter 2 we know that, to guarantee that a constraint R can be integrated with a BN $G = (G_S, G_P)$, there must exist a JPD $P(X)$ which can both satisfy R and be factored according to G_S. According to this and Theorem 3.2 we have the following theorem:

Theorem 3.3 Let $\mathbf{P}_R$ denote the set of all JPDs that can satisfy constraint R, and let $\mathbf{P}_{G_S}$ denote the set of all JPDs that can be factored according to G_S. $\mathbf{P}_R \cap \mathbf{P}_{G_S} \neq \varnothing$ if and only if every dependency implied by R holds in G_S.

Proof:

[Sufficiency] Let X be the set of variables of G_S, and Y be the set of variables of R.

Case 1. When R and G_S have the same set of variables, i.e., $Y = X$, then $\mathbf{P}_R = \{R\}$. If every dependency implied by R holds in G_S, then by Theorem 3.2, $R \in \mathbf{P}_{G_S}$. Thus $\mathbf{P}_R \cap \mathbf{P}_{G_S} = \{R\} \neq \varnothing$.

Case 2. When the set of variables of R is a subset of that of G_S, i.e., $Y \subset X$, we can construct a JPD $P(X)$ such that $P(Y) = R(Y)$ and each variable in $X - Y$ is independent of the other variables in X. That is, $P(X)$ satisfies R, and $P(X)$ also has the same dependencies as R. Thus $P(X) \in \mathbf{P}_R$, and every dependency implied by $P(X)$ holds in G_S. Since $P(X)$ and G_S have the same set of variables, according to Case 1, $\mathbf{P}_R \cap \mathbf{P}_{G_S} = \{P\} \neq \varnothing$.

[Necessity] Let $P(X) \in \mathbf{P}_R \cap \mathbf{P}_{G_S}$, then $P(X) \in \mathbf{P}_R$ and $P(X) \in \mathbf{P}_{G_S}$. Since $P(X)$ satisfies R, every dependency in R also holds in $P(X)$. Since $P(X) \in \mathbf{P}_{G_S}$, according to Theorem 3.2, every dependency implied by $P(X)$ holds in G_S. Therefore, every dependency implied by R holds in G_S. □

Based on Theorem 3.3, we have our definition for structurally inconsistent constraint:

Definition 3.5 (Structurally Inconsistent Constraint) Given a BN $G = (G_S, G_P)$ over a set of variables X, and a probability constraint R over a set of variables Y, where $Y \subseteq X$, if every dependency implied by R holds in G_S, then we say R is *structurally consistent* with G_S, or R is a *structurally consistent constraint* for G_S. Otherwise if any dependency

implied by R does not hold in G_S, we say R is *structurally inconsistent* with G_S, or R is a *structurally inconsistent constraint* for G_S.

Based on Definition 2.4 and Definition 3.5, we can use the following way to differentiate structural inconsistencies from other inconsistencies between a BN $G = (G_S, G_P)$ and a set of constraints $\mathbf{R}$. $\forall R_i \in \mathbf{R}$, if $\mathbf{P}_{R_i} \cap \mathbf{P}_{G_S} = \varnothing$, then R_i has structural inconsistency with the BN. If $\mathbf{P}_{\mathbf{R}} = \varnothing$, then $\mathbf{R}$ has inconsistency among the constraints. As an example, constraints $R_1(A,B)$ and $R_2(A,C)$ in Figure 2.5 are inconsistent with each other, and constraint $R_3(A,B,C)$ in Figure 2.5 has structural inconsistency with the BN.

With the definition of structurally inconsistent constraint, the problem we set to solve in this thesis can be formally stated as follows:

Given a BN $G = (G_S, G_P)$ with JPD $P(X)$, and a set of consistent constraints $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$ in which some of the constraints have structural inconsistencies with G_S, construct a new BN $\tilde{G} = (\tilde{G}_S, \tilde{G}_P)$ with probability distribution $\tilde{P}(X)$ that meets the following conditions:

C1: *Constraint satisfaction*: $\forall R_j(Y^j) \in \mathbf{R}, \tilde{P}(Y^j) = R_j(Y^j)$;

C2: *Minimality*: I-divergence $I(\tilde{P}(X) \,||\, P(X))$ and the change from G_S to $\tilde{G}_S$ are as small as possible.

3.2 The Method to Identify Structural Inconsistencies

In the previous section we have established the theorem that any dependency that is implied by the constraint but does not hold in the BN will cause structural inconsistencies. In this section we will develop a method based on this theorem to determine if a set of constraints has structural inconsistencies with a BN. This is accomplished by first extracting dependency information from each constraint using the *independent test*, then checking in the BN to see if any of those dependencies are not captured by the BN structure based on the *d-separation* method. We will provide an example to show how this method works.

3.2.1 The Independence Test

To extract dependency information from a constraint, we can perform the independence test. The kind of independence test we use in this thesis is as follows. The 0^{th} order unconditional independence test is to check whether $R(Y_u, Y_v) = R(Y_u) \cdot R(Y_v)$ holds for each pair of variables Y_u and Y_v in constraint R. If the equality holds, Y_u and Y_v are independent, otherwise they are dependent. The j^{th} order independence test for R is to check whether the following equation holds for each pair of variables Y_u and Y_v in R:

$$R(Y_u, Y_v \mid W) = R(Y_u \mid W) \cdot R(Y_v \mid W) \qquad \textbf{(3.1)}$$

where W is a set of any j number of variables in R other than Y_u and Y_v.

Taking into consideration of the numerical precision during the test, we set a threshold of 10^{-4} in all our experiments when comparing $R(Y_u, Y_v \mid W)$ with

$R(Y_u \mid W) \cdot R(Y_v \mid W)$. That is, Y_u and Y_v are considered independent given W when the following equation holds:

$$\left| R(Y_u, Y_v \mid W) - R(Y_u \mid W) \cdot R(Y_v \mid W) \right| < 10^{-4} \quad \textbf{(3.2)}$$

All dependencies in a constraint can then be obtained by performing all independent tests according to (3.2) from 0^{th} order to $(n-2)^{th}$ order.

3.2.2 The d-separation Method

To check whether some dependency is captured by a BN structure, we can use the d-separation method [58]. This method can identify if two variables are dependent in a BN by looking at the connection types for the variables along the path between these two variables. In general, there are three types of connections in BNs.

The first type is *serial connection*, i.e. $X_u \rightarrow X_w \rightarrow X_v$, where X_u, X_v and X_w are three variables in the BN. If X_w is not observed, X_u and X_v are dependent. Information can be transmitted between X_u and X_v through X_w if X_w is not observed. If X_w is observed, X_u and X_v are independent. Information cannot be transmitted between X_u and X_v through X_w if X_w is observed. Observing X_w blocks the information path.

The second type is *diverging connection*, i.e. $X_u \leftarrow X_w \rightarrow X_v$, also known as common cause. If X_w is not observed, X_u and X_v are dependent. Information can be transmitted through X_w among children of X_w if X_w is not observed. If X_w is observed, X_u and X_v are independent. Information cannot be transmitted through X_w among children of X_w if X_w is observed. Observing X_w blocks the information path.

The third type is *converging connection*, i.e. $X_u \rightarrow X_w \leftarrow X_v$, also known as common effect. If neither X_w nor any of its descendants are observed, X_u and X_v are independent. Information cannot be transmitted through X_w among the parents of X_w. It leaks through X_w and its descendants. If X_w or any of its descendants is observed, X_u and X_v are dependent. Information can be transmitted through X_w among parents of X_w if X_w or any of its descendants are observed. Observing X_w or its descendants opens the information path.

A path between X_u and X_v is *blocked* by a set of variables W if either that path contains a variable X_w that is in W and the connection at X_w is either serial or diverging, or that the path contains a variable X_w such that X_w and its descendants are not in W and the connection at X_w is a converging connection. We say X_u and X_v are *d-separated* by W if all paths between X_u and X_v are blocked by W.

3.2.3 The InconsId Method

Based on Definition 3.5 for structurally inconsistent constraint, we propose a method to identify structural inconsistencies between a BN and a set of constraints. The method is named *InconsId*, which stands for inconsistency identification.

For each constraint R_i in $\mathbf{R}$, the InconsId method first extracts all the dependencies from R_i using the independence test, and checks whether each dependency holds in the BN using the d-separation method. If not, this dependency is added to a local dependency list $\mathbf{DL}_i$ in the format of $< X_u, X_v, W >$, which means X_u and X_v are dependent given a set of variables W in the constraint but independent in the BN. Here we assume the nodes are

in topological order according to the DAG of the BN, and X_u has lower index than X_v. Thus, if a link between X_u and X_v is to be added, it shall have the direction of $X_u \rightarrow X_v$. After all the independence tests are completed for R_i, if $\mathbf{DL}_i$ is still an empty list, it means all the dependencies implied by R_i hold in the BN. Thus R_i is classified as a structurally consistent constraint and is added to the global structurally consistent constraint set $\mathbf{R}^+$. Otherwise if there are still any items left in $\mathbf{DL}_i$, it means these dependencies are implied by R_i but do not hold in the BN. Thus R_i is classified as a structurally inconsistent constraint and is added to the global structurally inconsistent constraint set $\mathbf{R}^-$. Each item in $\mathbf{DL}_i$ is added to the global dependency list $\mathbf{DL}$ if it does not already exist in $\mathbf{DL}$.

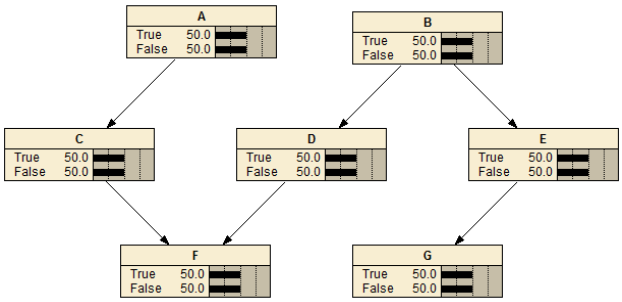
After all the constraints are processed, the method returns $\mathbf{R}^+$, $\mathbf{R}^-$ and $\mathbf{DL}$. The items in $\mathbf{DL}$ contain important information which will be used later in modifying the structure of the existing BN to overcome the structural inconsistencies. The following is the InconsId method.

InconsId ($G = (G_S, G_P)$, $\mathbf{R} = \{R_1, \cdots, R_m\}$)

1. $\mathbf{R}^+ = \varnothing$, $\mathbf{R}^- = \varnothing$, $\mathbf{DL} = \varnothing$;
2. For each constraint R_i in $\mathbf{R}$, do the following:
   2.1. $\mathbf{DL}_i = \varnothing$, n_i = the number of variables in R_i;
   2.2. If $n_i = 1$, go to step 2.3. Otherwise for each pair of variables X_u and X_v in R_i, do from 0^{th} order to $(n-2)^{th}$ order independence test for them according to (3.2). If the test fails, use the d-separation method to check in G_S whether X_u and X_v are dependent given a set of variables W in the constraints. If they are independent in G_S, add $< X_u, X_v, W >$ to $\mathbf{DL}_i$, with X_u having lower index than X_v in the BN;
   2.3. If $\mathbf{DL}_i = \varnothing$, add R_i to $\mathbf{R}^+$. Otherwise, add R_i to $\mathbf{R}^-$, and for each entry in $\mathbf{DL}_i$, add it to $\mathbf{DL}$ if it does not already exist in $\mathbf{DL}$;
3. Return $\mathbf{R}^+$, $\mathbf{R}^-$, and $\mathbf{DL}$.

The following example illustrates how the InconsId method works with the BN in Figure 3.1 and four constraints in Figure 3.2. For each constraint, the method checks whether all the dependencies existing in the constraint hold in the BN. For $R_1(B, D, F)$, $\mathbf{DL}_1 = \varnothing$, so it is added to $\mathbf{R}^+$. For $R_2(B, D, G)$, $\mathbf{DL}_2 = \{< D, G, \{B\} >\}$, so it is added to $\mathbf{R}^-$, and $< D, G, \{B\} >$ is added to $\mathbf{DL}$. For $R_3(D, E, G)$, $\mathbf{DL}_3 = \{< D, G, \{E\} >\}$, so it is added to $\mathbf{R}^-$, and $< D, G, \{E\} >$ is added to $\mathbf{DL}$. For $R_4(D, F, G)$, $\mathbf{DL}_4 = \varnothing$, so it is added

to $\mathbf{R}^{+}$. At the end, the method returns $\mathbf{R}^{+} = \{R_1(B,D,F), R_4(D,F,G)\}$, $\mathbf{R}^{-} = \{R_2(B,D,G), R_3(D,E,G)\}$, and $\mathbf{DL} = \{<D,G,\{B\}>, <D,G,\{E\}>\}$.



A	
true	false
0.500	0.500

B	
true	false
0.500	0.500

A	C	
	true	false
true	0.500	0.500
false	0.500	0.500

B	D	
	true	false
true	0.500	0.500
false	0.500	0.500

C	D	F	
		true	false
true	true	0.500	0.500
true	false	0.500	0.500
false	true	0.500	0.500
false	false	0.500	0.500

B	E	
	true	false
true	0.500	0.500
false	0.500	0.500

E	G	
	true	false
true	0.500	0.500
false	0.500	0.500

Figure 3.1 A 7 node BN and its CPTs

B	D	F	$R_1(B,D,F)$
true	true	true	0.320
true	true	false	0.320
true	false	true	0.080
true	false	false	0.080
false	true	true	0.080
false	true	false	0.080
false	false	true	0.020
false	false	false	0.020

B	D	G	$R_2(B,D,G)$
true	true	true	0.288
true	true	false	0.352
true	false	true	0.048
true	false	false	0.112
false	true	true	0.072
false	true	false	0.088
false	false	true	0.012
false	false	false	0.028

D	E	G	$R_3(D,E,G)$
true	true	true	0.072
true	true	false	0.088
true	false	true	0.288
true	false	false	0.352
false	true	true	0.012
false	true	false	0.028
false	false	true	0.048
false	false	false	0.112

D	F	G	$R_4(D,F,G)$
true	true	true	0.180
true	true	false	0.220
true	false	true	0.180
true	false	false	0.220
false	true	true	0.030
false	true	false	0.070
false	false	true	0.030
false	false	false	0.070

Figure 3.2 Constraints $R_1(B,D,F), R_2(B,D,G), R_3(D,E,G)$ and $R_4(D,F,G)$

3.3 Experiments

In this section we will analyze the time complexity of the InconsId method and conduct experiments to evaluate the execution time of the method. To analyze the time complexity of the InconsId method, we first consider a single constraint R. Let n be the number of variables in R, so the number of pairs of variables in R is $C_n^2 = n(n-1)/2$. For each pair of variables, the number of independence tests performed is $C_{n-2}^0 + C_{n-2}^1 + \cdots + C_{n-2}^{n-2} = 2^{n-2}$. If any of the independence tests fails, the method checks whether the dependency holds in the BN using the d-separation method, which can be implemented in linear time of the size of the BN [19]. Let $|V|$ be the number of nodes and $|E|$ be the number of links in the BN, the time complexity of the d-separation method is $O(|V|+|E|)$. So the time complexity of the InconsId method for a single constraint is $O(n^2 \cdot 2^n \cdot (|V|+|E|))$.

Since the InconsId method processes every constraint in the same way, the time complexity for a set of constraints $\mathbf{R} = \{R_1, \cdots, R_m\}$ will be m times of the time complexity for a single constraint, where m is the number of constraints in $\mathbf{R}$. That is, the total time complexity of the InconsId method is $O(\sum_{i=1}^{m} (n_i^2 \cdot 2^{n_i} \cdot (|V|+|E|)))$.

Based on the above analysis, we know that the execution time of the InconsId method is related to the number of variables in each constraint, the size of the BN, as well as the number of constraints in the constraint set. Next we will conduct experiments to evaluate the performance of the InconsId method and see empirically how its execution time is

affected by each of these factors. The experiments in this thesis are conducted based on Netica API from Norsys [89].

The first set of experiments is designed to evaluate the execution time of the InconsId method for a single constraint when the number of variables in the constraint varies from 2 to 10. The BN in this set of experiments has 25 nodes and 50 links. The result in Figure 3.3 shows the relation between the number of variables in the constraint and the execution time of the InconsId method when the size of the BN and the number of constraints are fixed.

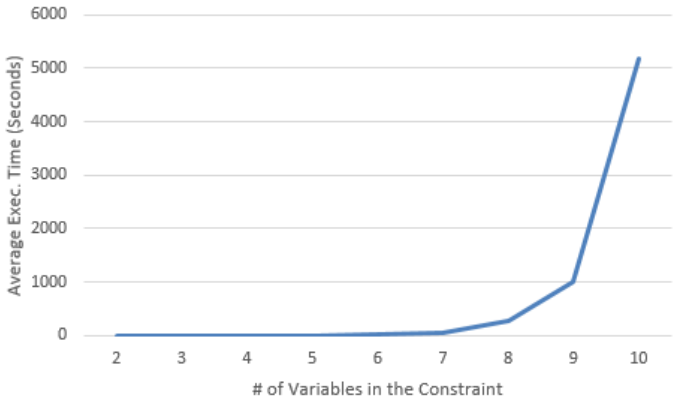


Figure 3.3 Result of Experiment 1 for the InconsId method

The second set of experiments is designed to evaluate the execution time of the InconsId method when the number of binary variables in the BN varies from 10 to 80. This set of experiments uses a single constraint with 6 variables. The result in Figure 3.4 shows the relation between the size of the BN and the execution time of the InconsId method when the number of variables in the constraint and the number of constraints are fixed.

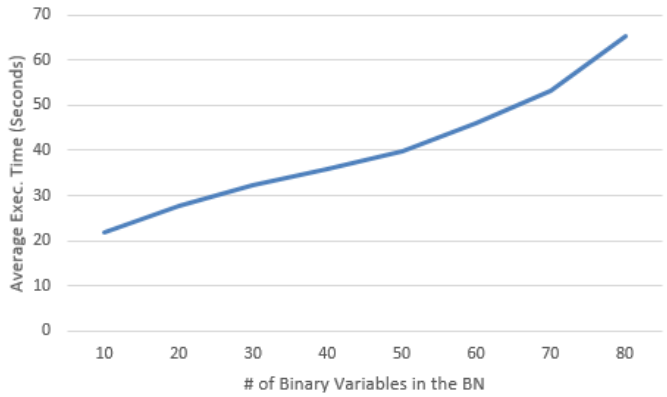


Figure 3.4 Result of Experiment 2 for the InconsId method

The third set of experiments is designed to evaluate the execution time of the InconsId method when the number of constraints varies from 2 to 10. Each constraint in the constraint set has 6 variables. The BN in this set of experiments has 25 binary variables. The result in Figure 3.5 shows the relation between the number of constraints in the constraint set and the execution time of the InconsId method when the number of variables in each constraint and the size of the BN are fixed.

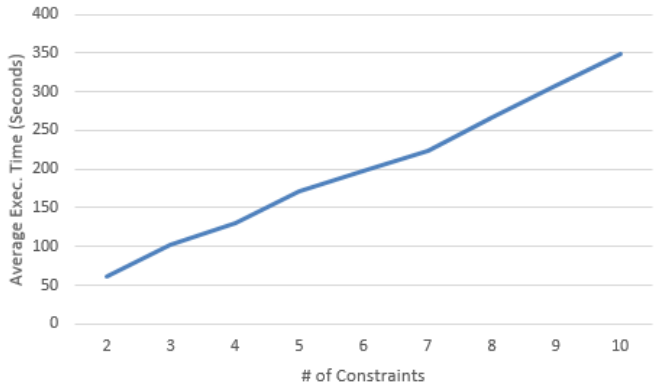


Figure 3.5 Result of Experiment 3 for the InconsId method

From the above experiment results we can see that the execution time of the InconsId method increases exponentially with the number of variables in the constraint, and increases linearly with either the size of the BN or the number of constraints. These experiment results are consistent with our theoretical analysis.

3.4 Summary

In this chapter we established the theorem that a constraint is structurally consistent with a BN if and only if every dependency implied by the constraint holds in the BN structure. Based on this theorem, we defined structurally inconsistent constraint which has dependency that does not hold in the BN structure, and described how to differentiate structural inconsistency from other inconsistencies. With the definition of structurally inconsistent constraint, we provided a formal statement for the problem we set to solve in this thesis.

Based on the established theorem and definition, we proposed the InconsId method to identify structural inconsistencies between a BN and a set of constraints. The method first extracts all the dependencies from each constraint using the independence test and then checks whether each dependency holds in the BN using the d-separation method. If any of these dependencies do not hold in the BN structure, the constraint will be classified as a structurally inconsistent constraint. We demonstrated how the InconsId method works through an example.

We analyzed the time complexity of the InconsId method and conducted experiments to evaluate its performance on the execution time. It was shown that the execution time of the InconsId method increases exponentially when the number of variables in the constraint increases, and increases linearly when either the size of the BN or the number of constraints increases.

4 Overcome Structural Inconsistencies – AddNode Methods

With our proposed methods in the last chapter, structural inconsistencies can be identified between a BN and a set of constraints. Our next goal is to overcome these identified structural inconsistencies by modifying the structure of the existing BN. In this chapter we will address this issue by adding nodes to the existing BN. This is done by adapting the virtual evidence method originally developed for dealing with uncertain evidence in BN reasoning. The class of methods proposed in this chapter is called AddNode methods.

In Section 4.1 we introduce the basic method of this class, named AddNode-Basic. This method overcomes the structural inconsistencies between a BN and a set of constraints by adding one node for each constraint. It executes a process similar to BN-IPFP which adds a virtual node for each constraint and repeatedly iterates over all the constraints until convergence. At the convergence, this extended BN satisfies all the constraints. Several variations of AddNode-Basic are proposed later to improve computational efficiency and minimize the changes to the existing BN structure.

In Section 4.2 we propose the AddNode+Merge method which adds only one virtual node for all the constraints. This is done by first merging all the constraints into a single constraint with IPFP, then using AddNode-Basic to integrate the merged constraint by adding only one node for it to the existing BN.

In Section 4.3 we propose the AddNode+D-IPFP method to integrate the structurally inconsistent constraints using the AddNode-Basic method while integrating the structurally consistent constraints using D-IPFP. Compared to the AddNode-Basic, the AddNode+D-IPFP avoids adding nodes for the structurally consistent constraints.

In Section 4.4 we propose the AddNode+Factorization method to factorize large constraints into smaller ones and add nodes only for the structurally inconsistent parts of the constraints. In this way the number of the parents as well as the size of the CPTs for the added nodes can be reduced.

In Section 4.5 we analyze each method theoretically for the class of AddNode methods, and conduct experiments to evaluate their performance.

4.1 The AddNode-Basic Method

Given a BN and a structurally inconsistent constraint, we can accommodate the identified structural inconsistencies in a way similar to Pearl's virtual evidence method. This is accomplished by first adding a node to the existing BN for the constraint and making this node the child of all the variables of that constraint, then setting the CPT of the node with the likelihood ratio calculated from the constraint using (2.19), and setting the state of the node to true. As the parents of the added node, all the variables covered by the constraint have converging connection with each other. Thus, they are dependent when the added node is set to true. With this added structure, the dependencies in the constraints that do not hold in the original BN now can be represented by the revised BN structure. Thus, the structural inconsistencies are overcome. Also, because the likelihood ratio is calculated using (2.19) and is used to construct the CPT of the added node, based on the virtual evidence method, the updated distribution will satisfy the constraint once the added node is set to true.

For multiple constraints $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$, after satisfying one constraint, the updated distribution may not satisfy the other constraints. Inspired by BN-IPFP-1, we

combine the above process for a single constraint with IPFP to iterate over all the constraints until convergence. Taking into account of the numerical precision, the method is considered to be at convergence when the following equation holds:

$$|\sum_{j=1}^{m}\delta(Q(Y^j), R_j(Y^j))| < 10^{-4} \quad \textbf{(4.1)}$$

where $Q(Y^j)$ is the marginal distribution on Y^j in the BN, and $\delta(Q(Y^j), R_j(Y^j))$ is the total variation between $Q(Y^j)$ and $R_j(Y^j)$. This rule also applies to the other AddNode methods for determining convergence. The following is the AddNode-Basic method.

AddNode-Basic ($G = (G_S, G_P)$, $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$)

1. $Q_0(X) = P(X)$, where $P(X)$ is the joint distribution of the given BN;

2. For each $R_j(Y^j)$ in $\mathbf{R}$, add a node V_j to G_S with variables in Y^j as its parents;

3. Starting with $k = 1$, repeat the following process until convergence:

   3.1. $j = 1 + (k-1) \bmod m$; $l = 1 + |(k-1)/m|$;

   3.2. Calculate the likelihood ratio from $Q_{k-1}(Y^j)$ and $R_j(Y^j)$ by (2.19):

$$L_{j,l}(Y^j) = \frac{R(y^j_{(1)})}{Q_{k-1}(y^j_{(1)})} : \frac{R(y^j_{(2)})}{Q_{k-1}(y^j_{(2)})} : \cdots : \frac{R(y^j_{(l)})}{Q_{k-1}(y^j_{(l)})}$$

   where $y^j_{(1)}, y^j_{(2)}, ..., y^j_{(l)} \in \mathbf{Y}^j$ are the state configurations of Y^j ;

   3.3. Construct the CPT of V_j with likelihood ratio $L_{j,l}(Y^j)$ by (2.20).

   The CPT entries when Y^j takes the instantiation of $y^j_{(t)}$ are calculated by:

$$P(V_j = true \mid y^j_{(t)}) = \frac{R(y^j_{(t)})}{Q_{k-1}(y^j_{(t)})} \Bigg/ \sum_{t=1}^{l} \frac{R(y^j_{(t)})}{Q_{k-1}(y^j_{(t)})}$$

$$P(V_j = false \mid y^j_{(t)}) = 1 - \left(\frac{R(y^j_{(t)})}{Q_{k-1}(y^j_{(t)})} \Bigg/ \sum_{t=1}^{l} \frac{R(y^j_{(t)})}{Q_{k-1}(y^j_{(t)})} \right);$$

   3.4. Set the state of V_j to true, and let $Q_k(X)$ denote the distribution of the updated BN;

   3.5. $k = k + 1$;

4. Return the updated BN as output.

The convergence and correctness of the AddNode-Basic method are established in Theorem 4.1 below.

Theorem 4.1 If constraints in $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$ are consistent with each other, then the AddNode-Basic method converges to JPD $Q^*(X)$, where $\forall R_i(Y^i) \in \mathbf{R}$, $Q^*(Y^i) = R_i(Y^i)$.

Proof:

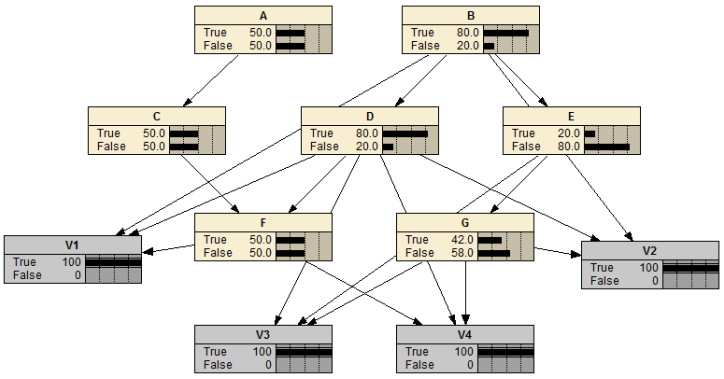
Consider the k^{th} iteration when constraint $R_j(Y^j)$ is selected to update the BN. $Q_k(X)$ obtained in step 3.4 of the AddNode-Basic method is the same as the one obtained by updating $Q_{k-1}(X)$ with $L_{j,l}(Y^j)$ using Pearl's virtual evidence method. According to Theorem 5 in [12], when this virtual evidence is applied to $Q_{k-1}(X)$, the updated distribution is exactly the same as what can be obtained using Jeffrey's rule after applying $R_j(Y^j)$. It has also been proven in [66] that after updating $Q_{k-1}(X)$ by $R_j(Y^j)$ using Jeffery's rule, the resulting distribution $Q_k(X)$ is the same as the I-projection of $Q_{k-1}(X)$ on $\mathbf{P}_{R_j(Y^j)}$. That is,

$$Q_k(X) = Q_{k-1}(X) \cdot \frac{R_j(Y^j)}{Q_{k-1}(Y^j)},$$

which is one step of IPFP. This means the AddNode-Basic method is the same as applying IPFP to a set of consistent constraints $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$. Therefore, it converges to joint probability distribution $Q^*(X)$, where $\forall R_i(Y^i) \in \mathbf{R}, Q^*(Y^i) = R_i(Y^i)$. □

The following example illustrates how the AddNode-Basic method works. Given a BN in Figure 3.1 and four constraints in Figure 3.2, some of which are inconsistent with the BN, AddNode-Basic first adds one node to the existing BN for each constraint. Then, for each constraint, it calculates the likelihood ratio based on the constraint and the

probability distribution of the current BN, then constructs the CPT of the node added for the constraint based on the likelihood ratio, and sets the state of the added node to true. This process is repeated until converging in 2 iterations. Figure 4.1 shows the resulting BN (a) together with the four CPTs of the virtual nodes (b). Figure 4.2 shows the marginal distributions in the resulting BN for the variables in each constraint. We can see that the resulting BN satisfies all the four constraints.



(a) Resulting BN after integrating the four constraints.

B	D	F	V1	
			true	false
true	true	true	0.320	0.680
true	true	false	0.320	0.680
true	false	true	0.080	0.920
true	false	false	0.080	0.920
false	true	true	0.080	0.920
false	true	false	0.080	0.920
false	false	true	0.020	0.980
false	false	false	0.020	0.980

B	D	G	V2	
			true	false
true	true	true	0.112	0.888
true	true	false	0.138	0.862
true	false	true	0.075	0.925
true	false	false	0.175	0.825
false	true	true	0.112	0.888
false	true	false	0.138	0.862
false	false	true	0.075	0.925
false	false	false	0.175	0.825

D	E	G	V3	
			true	false
true	true	true	0.050	0.950
true	true	false	0.050	0.950
true	false	true	0.200	0.800
true	false	false	0.200	0.800
false	true	true	0.050	0.950
false	true	false	0.050	0.950
false	false	true	0.200	0.800
false	false	false	0.200	0.800

D	F	G	V4	
			true	false
true	true	true	0.125	0.875
true	true	false	0.125	0.875
true	false	true	0.125	0.875
true	false	false	0.125	0.875
false	true	true	0.125	0.875
false	true	false	0.125	0.875
false	false	true	0.125	0.875
false	false	false	0.125	0.875

(b) Resulting CPTs of the added nodes.

Figure 4.1 Result after running the AddNode-Basic method

B	D	F	$Q^*(B,D,F)$
true	true	true	0.320
true	true	false	0.320
true	false	true	0.080
true	false	false	0.080
false	true	true	0.080
false	true	false	0.080
false	false	true	0.020
false	false	false	0.020

B	D	G	$Q^*(B,D,G)$
true	true	true	0.288
true	true	false	0.352
true	false	true	0.048
true	false	false	0.112
false	true	true	0.072
false	true	false	0.088
false	false	true	0.012
false	false	false	0.028

D	E	G	$Q^*(D,E,G)$
true	true	true	0.072
true	true	false	0.088
true	false	true	0.288
true	false	false	0.352
false	true	true	0.012
false	true	false	0.028
false	false	true	0.048
false	false	false	0.112

D	F	G	$Q^*(D,F,G)$
true	true	true	0.180
true	true	false	0.220
true	false	true	0.180
true	false	false	0.220
false	true	true	0.030
false	true	false	0.070
false	false	true	0.030
false	false	false	0.070

Figure 4.2 Marginal distributions of the resulting BN

4.2 The AddNode+Merge Method

The AddNode-Basic method adds one node for each constraint and iterates repeatedly over all the constraints until convergence. To reduce the number of added nodes and avoid computing the marginal $Q_{k-1}(X)$ in each iteration, we can first merge all the constraints into one constraint using IPFP, then add a single node to the existing BN for the merged constraint, and set the state of the node to true. In this way only one node is added to the existing BN, and the update to the BN only happens once. The following is the AddNode+Merge method.

AddNode+Merge ($G = (G_S, G_P)$, $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$)

1. $Q_0(Y) = P(Y)$ where $Y = Y^1 \cup \cdots \cup Y^m$, and $P(X)$ is the joint distribution of the given BN;
2. Apply IPFP on $Q_0(Y)$ with constraints in $\mathbf{R}$ until converging to $Q^*(Y)$;
3. Add a node V to G_S with variables in Y as its parents;
4. Calculate likelihood ratio $L(Y)$ from $Q_0(Y)$ and $Q^*(Y)$ by (2.19);
5. Construct CPT of V with likelihood ratio $L(Y)$ by (2.20);
6. Set the state of V to true to update the BN;
7. Return the updated BN as output.

The convergence and correctness of the AddNode+Merge method are established in Theorem 4.2 below.

Theorem 4.2 If constraints in $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$ are consistent with each other, then the AddNode+Merge method converges to the same JPD as the AddNode-Basic method.

Proof:

Since constraints in $\mathbf{R}$ are consistent with each other, the result of step 2 will converge to $Q^*(Y)$. After calculating $L(Y)$ from $Q_0(Y)$ and $Q^*(Y)$ in step 4 and updating $G = (G_S, G_P)$ with $L(Y)$ using Pearl's virtual evidence method in step 6, the method will converge because it is a special case of AddNode-Basic method with a single constraint. Let $Q^*(X)$ be the distribution of the resulting BN. According to Theorem 5 of [12] and Theorem 1 of [66], we have:

$$Q^*(X) = Q_0(X) \cdot \frac{Q^*(Y)}{Q_0(Y)}.$$

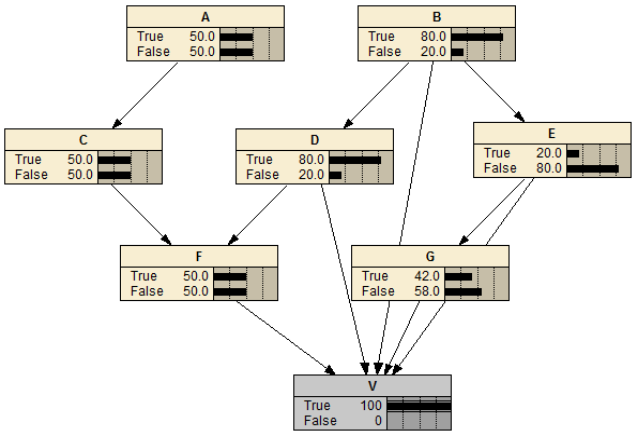
Also, according to Theorem 4 of [66], given an initial distribution $Q_0(X)$ and a set of consistent constraints $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$, applying IPFP on $Q_0(X)$ with m constraints in $\mathbf{R}$ is equivalent to modifying $Q_0(X)$ with a single constraint $Q^*(Y)$, where $Y = Y^1 \cup \cdots \cup Y^m$, and $Q^*(Y)$ is the converging distribution when applying IPFP on $Q_0(Y)$ with constraints in $\mathbf{R}$. Thus the AddNode+Merge method converges to the same JPD as the AddNode-Basic method. □

The following example illustrates how AddNode+Merge method works. Given a BN in Figure 3.1 and four constraints in Figure 3.2, it first runs IPFP to merge the constraints into one constraint $R'(B, D, E, F, G)$ as shown in Figure 4.3. Then it adds a node V to the existing BN, calculates the likelihood ratio based on $R'(B, D, E, F, G)$ and the probability distribution of the BN, constructs the CPT of V based on the likelihood ratio, and sets the state of V to true. Figure 4.4 shows the resulting BN (a) together with the resulting CPTs for the added node (b). Experiment shows that this BN satisfies all the four constraints. Also note that, the two BNs in Figure 4.1 (a) and 4.4 (a) are identical except the former has four added virtual nodes and the latter only has one.

B	D	E	F	G	$R'(B,D,E,F,G)$
true	true	true	true	true	0.029
true	true	true	true	false	0.035
true	true	true	false	true	0.029
true	true	true	false	false	0.035
true	true	false	true	true	0.115
true	true	false	true	false	0.141
true	true	false	false	true	0.115
true	true	false	false	false	0.141
true	false	true	true	true	0.005
true	false	true	true	false	0.011
true	false	true	false	true	0.005
true	false	true	false	false	0.011
true	false	false	true	true	0.019
true	false	false	true	false	0.045
true	false	false	false	true	0.019
true	false	false	false	false	0.045

B	D	E	F	G	$R'(B,D,E,F,G)$
false	true	true	true	true	0.007
false	true	true	true	false	0.009
false	true	true	false	true	0.007
false	true	true	false	false	0.009
false	true	false	true	true	0.029
false	true	false	true	false	0.035
false	true	false	false	true	0.029
false	true	false	false	false	0.035
false	false	true	true	true	0.001
false	false	true	true	false	0.003
false	false	true	false	true	0.001
false	false	true	false	false	0.003
false	false	false	true	true	0.005
false	false	false	true	false	0.011
false	false	false	false	true	0.005
false	false	false	false	false	0.011

Figure 4.3 The merged constraint $R'(B,D,E,F,G)$



(a) Resulting BN after integrating the four constraints.

B	D	E	F	G	V	
					true	false
true	true	true	true	true	0.029	0.971
true	true	true	true	false	0.035	0.965
true	true	true	false	true	0.029	0.971
true	true	true	false	false	0.035	0.965
true	true	false	true	true	0.115	0.885
true	true	false	true	false	0.141	0.859
true	true	false	false	true	0.115	0.885
true	true	false	false	false	0.141	0.859
true	false	true	true	true	0.005	0.995
true	false	true	true	false	0.011	0.989
true	false	true	false	true	0.005	0.995
true	false	true	false	false	0.011	0.989
true	false	false	true	true	0.019	0.981
true	false	false	true	false	0.045	0955
true	false	false	false	true	0.019	0.981
true	false	false	false	false	0.045	0955

B	D	E	F	G	V	
					true	false
false	true	true	true	true	0.007	0.993
false	true	true	true	false	0.009	0.991
false	true	true	false	true	0.007	0.993
false	true	true	false	false	0.009	0.991
false	true	false	true	true	0.029	0.971
false	true	false	true	false	0.035	0.965
false	true	false	false	true	0.029	0.971
false	true	false	false	false	0.035	0.965
false	false	true	true	true	0.001	0.999
false	false	true	true	false	0.003	0.997
false	false	true	false	true	0.001	0.999
false	false	true	false	false	0.003	0.997
false	false	false	true	true	0.005	0.995
false	false	false	true	false	0.011	0.989
false	false	false	false	true	0.005	0.995
false	false	false	false	false	0.011	0.989

(b) Resulting CPTs of the added node.

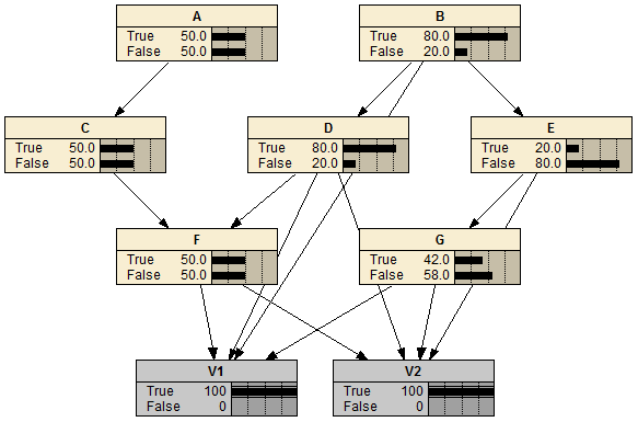
Figure 4.4 Result after running the AddNode+Merge method

The size of the CPT for the single added node grows exponentially with the number of distinct variables in the constraint set, which will impact the performance of the method. To solve the problem, we can combine the idea of the AddNode+Merge method with the AddNode-Basic method by first dividing the constraints in the constraint set into several groups, then merging the constraints in each group into one constraint which contains a moderate number of variables, and integrating these merged constraints with the existing BN using the AddNode-Basic method. The following example shows how it works. Given a BN in Figure 3.1 and four constraints in Figure 3.2, we first use IPFP to merge the first two constraints into constraint $R_1^{'}(B,D,F,G)$ and the last two constraints into constraint $R_2^{'}(D,E,F,G)$, which are shown in Figure 4.5. Then we use the AddNode-Basic method to integrate the two merged constraints $R_1^{'}(B,D,F,G)$ and $R_2^{'}(D,E,F,G)$ with the existing BN. Figure 4.6 shows the resulting BN (a) together with the resulting CPTs for the added nodes (b). Experiment shows that this BN satisfies all the four constraints.

B	D	F	G	$R_1^{'}(B,D,F,G)$
true	true	true	true	0.144
true	true	true	false	0.176
true	true	false	true	0.144
true	true	false	false	0.176
true	false	true	true	0.024
true	false	true	false	0.056
true	false	false	true	0.024
true	false	false	false	0.056
false	true	true	true	0.036
false	true	true	false	0.044
false	true	false	true	0.036
false	true	false	false	0.044
false	false	true	true	0.006
false	false	true	false	0.014
false	false	false	true	0.006
false	false	false	false	0.014

D	E	F	G	$R_2^{'}(D,E,F,G)$
true	true	true	true	0.036
true	true	true	false	0.044
true	true	false	true	0.036
true	true	false	false	0.044
true	false	true	true	0.144
true	false	true	false	0.176
true	false	false	true	0.144
true	false	false	false	0.176
false	true	true	true	0.006
false	true	true	false	0.014
false	true	false	true	0.006
false	true	false	false	0.014
false	false	true	true	0.024
false	false	true	false	0.056
false	false	false	true	0.024
false	false	false	false	0.056

Figure 4.5 The merged constraints $R_1^{'}(B,D,F,G)$ and $R_2^{'}(D,E,F,G)$



(a) Resulting BN after integrating the four constraints.

B	D	F	G	V1	
				true	false
true	true	true	true	0.144	0.856
true	true	true	false	0.176	0.824
true	true	false	true	0.144	0.856
true	true	false	false	0.176	0.824
true	false	true	true	0.024	0.976
true	false	true	false	0.056	0.944
true	false	false	true	0.024	0.976
true	false	false	false	0.056	0.944
false	true	true	true	0.036	0.964
false	true	true	false	0.044	0.956
false	true	false	true	0.036	0.964
false	true	false	false	0.044	0.956
false	false	true	true	0.006	0.994
false	false	true	false	0.014	0.986
false	false	false	true	0.006	0.994
false	false	false	false	0.014	0.986

D	E	F	G	V2	
				true	false
true	true	true	true	0.025	0.975
true	true	true	false	0.025	0.975
true	true	false	true	0.025	0.975
true	true	false	false	0.025	0.975
true	false	true	true	0.100	0.900
true	false	true	false	0.100	0.900
true	false	false	true	0.100	0.900
true	false	false	false	0.100	0.900
false	true	true	true	0.025	0.975
false	true	true	false	0.025	0.975
false	true	false	true	0.025	0.975
false	true	false	false	0.025	0.975
false	false	true	true	0.100	0.900
false	false	true	false	0.100	0.900
false	false	false	true	0.100	0.900
false	false	false	false	0.100	0.900

(b) Resulting CPTs of the added nodes.

Figure 4.6 Result after running AddNode-Basic with two merged constraints

4.3 The AddNode+D-IPFP Method

The AddNode-Basic method adds a virtual node for each constraint regardless it is structurally consistent or inconsistent. The change to the existing BN can be greatly reduced if the added nodes can be limited to the structurally inconsistent constraints only, and let the structurally consistent constraints be integrated using E-IPFP or D-IPFP. In this thesis for computational reason we will only use D-IPFP to integrate structurally

consistent constraints. The new method named AddNode+D-IPFP combines the AddNode-Basic method with D-IPFP to reduce the number of added nodes to the existing BN. After grouping constraints in $\mathbf{R}$ into $\mathbf{R}^+$ and $\mathbf{R}^-$ using the InconsId method, it repeats the process of integrating the structurally inconsistent constraints in $\mathbf{R}^-$ using the AddNode-Basic method, and integrating the structurally consistent constraints in $\mathbf{R}^+$ using D-IPFP until convergence. The following is the AddNode+D-IPFP method.

AddNode+D-IPFP ($G = (G_S, G_P)$, $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$)

1. $Q_0(X) = P(X)$, where $P(X)$ is the joint distribution of the given BN;
2. Run the InconsId method to group $\mathbf{R}$ into $\mathbf{R}^+$ and $\mathbf{R}^-$;
3. For each $R_j(Y^j)$ in $\mathbf{R}^-$, add a node V_j to G_S with variables in Y^j as its parents;
4. Starting with $k = 1$, repeat the following process until convergence:

   /*apply the AddNode-Basic method*/

   4.1. For each $R_j(Y^j)$ in $\mathbf{R}^-$, do the following:

      4.1.1. Calculate likelihood ratio $L_j(Y^j)$ from $Q_{k-1}(Y^j)$ and $R_j(Y^j)$ by (2.19);

      4.1.2. Construct CPT of V_j with likelihood ratio $L_j(Y^j)$ by (2.20);

      4.1.3. Set the state of V_j to true, and let $Q_k(X)$ denote the distribution of the updated BN;

      4.1.4. $k = k + 1$;

   /*apply the D-IPFP algorithm*/

   4.2. For each $R_t(Y^t)$ in $\mathbf{R}^+$, do the following:

      4.2.1. $Q_k^{'}(Y^t \mid S^t) = Q_{k-1}(Y^t \mid S^t) \cdot \frac{R_t(Y^t)}{Q_{k-1}(Y^t)} \cdot \alpha_k$,

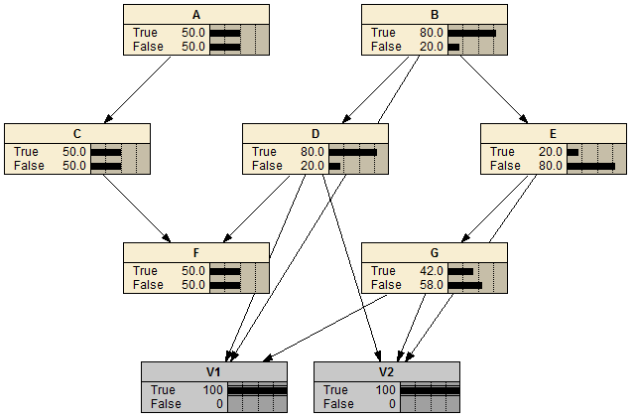
      where $S^t = (\bigcup_{X_i \in Y^t} \pi_i) \setminus Y^t$, and α_k is a normalization factor;

      4.2.2. $Q_k(X_i \mid \pi_i) = Q_k^{'}(X_i \mid \pi_i), \forall X_i \in Y^t$;

      4.2.3. $k = k + 1$;

5. Return the updated BN as output.

The following example shows how AddNode+D-IPFP method works. Given a BN in Figure 3.1 and four constraints in Figure 3.2, the method first runs the InconsId method to group the four constraints into $\mathbf{R}^{+} = \{R_1(B,D,F), R_4(D,F,G)\}$ and $\mathbf{R}^{-} = \{R_2(B,D,G)$, $R_3(D,E,G)\}$. Then it adds one node for each constraint in $\mathbf{R}^{-}$ to the existing BN. After that it runs step 3 of the AddNode-Basic method for $\mathbf{R}^{-}$ and one iteration of D-IPFP for $\mathbf{R}^{+}$ alternatively until convergence. Figure 4.7 shows the resulting BN (a) together with the updated CPTs for the existing nodes in the BN and the resulting CPTs for the added nodes (b). Experiment shows that this BN satisfies all the four constraints.



(a) Resulting BN after integrating the four constraints.

B	
true	false
0.800	0.200

B	D	
	true	false
true	0.800	0.200
false	0.800	0.200

E	G	
	true	false
true	0.420	0.580
false	0.420	0.580

B	D	G	V1	
			true	false
true	true	true	0.136	0.864
true	true	false	0.120	0.880
true	false	true	0.091	0.909
true	false	false	0.153	0.847
false	true	true	0.136	0.864
false	true	false	0.120	0.880
false	false	true	0.091	0.909
false	false	false	0.153	0.847

D	E	G	V2	
			true	false
true	true	true	0.050	0.950
true	true	false	0.050	0.950
true	false	true	0.200	0.800
true	false	false	0.200	0.800
false	true	true	0.050	0.950
false	true	false	0.050	0.950
false	false	true	0.200	0.800
false	false	false	0.200	0.800

(b) Resulting CPTs of the existing nodes and the added nodes.

Figure 4.7 Result after running the AddNode+D-IPFP method

4.4 The AddNode+Factorization Method

When integrating a structurally inconsistent constraint using the AddNode-Basic method, all the variables in the constraint are set as the parents of the added node. The size of the CPT of the added node grows exponentially with the number of variables in the structurally inconsistent constraint, which will affect the performance of the method. Therefore, we propose the AddNode+Factorization method to reduce the number of parents for the added nodes when integrating structurally inconsistent constraints with a large number of variables.

The method can factorize a large constraint into smaller ones and only add nodes for those that are structurally inconsistent with the existing BN. The factorization for each constraint is based on the interdependencies among its variables. Specifically, for each constraint $R_j(Y^j)$, the method orders its variables according to the topological order in G_S. Then for each variable Y_k^j in constraint $R_j(Y^j)$, the method finds the minimal subset $A_k^j \subseteq \{Y_1^j, \cdots, Y_{k-1}^j\}$ such that $R_j(Y_k^j \mid Y_1^j, \cdots, Y_{k-1}^j) = R_j(Y_k^j \mid A_k^j)$. This is similar to the process of constructing a minimal I-Map for $R_j(Y^j)$ [39]. Thus, we have $R_j(Y^j) = \prod_k R_j(Y_k^j \mid A_k^j)$. In this way, in order to satisfy $R_j(Y^j)$, we only need to make sure each factor $R_j(Y_k^j \mid A_k^j)$ is satisfied, or $R_j(\{Y_k^j\} \cup A_k^j)$ is satisfied. $R_j(\{Y_k^j\} \cup A_k^j)$ is called a *factorized constraint* and is added to $\mathbf{R}_f$.

After all constraints are factorized, the method groups the factorized constraints in $\mathbf{R}_f$ into structurally consistent constraint set $\mathbf{R}_f^+$ and structurally inconsistent constraint set $\mathbf{R}_f^-$ based on the dependencies on the dependency list. The method then integrates the

constraints in $\mathbf{R}_f^-$ with AddNode-Basic and integrates the constraints in $\mathbf{R}_f^+$ with D-IPFP. The following is the AddNode+Factorization method.

AddNode+Factorization ($G = (G_S, G_P)$, $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$)

1. $\mathbf{R}_f = \varnothing$, $\mathbf{R}_f^+ = \varnothing$, $\mathbf{R}_f^- = \varnothing$;
2. Run the InconsId method to get the dependency list **DL** ;
3. For each $R_j(Y^j)$ in $\mathbf{R}$, do the following:

   3.1. Order the variables in Y^j according to the topological order in G_S ;

   3.2. For each variable Y_k^j in Y^j , do the following:

   3.2.1. Find the minimal subset $A_k^j \subseteq \{Y_1^j, \cdots, Y_{k-1}^j\}$ such that

   $$R_j(Y_k^j \mid Y_1^j, \cdots, Y_{k-1}^j) = R_j(Y_k^j \mid A_k^j);$$

   3.2.2. $\forall R_f(Y^f) \in \mathbf{R}_f$, if $(\{Y_k^j\} \cup A_k^j) \not\subset Y^f$, then add $R_j(\{Y_k^j\} \cup A_k^j)$ to $\mathbf{R}_f$;

   3.2.3. $\forall R_f(Y^f) \in \mathbf{R}_f$, if $Y^f \subset (\{Y_k^j\} \cup A_k^j)$, then remove $R_f(Y^f)$ from $\mathbf{R}_f$;

4. For each $R_f(Y^f)$ in $\mathbf{R}_f$, if $\forall < X_u, X_v, W >$ in **DL** , $\{X_u, X_v\} \subseteq Y^f$, then add $R_f(Y^f)$ to $\mathbf{R}_f^-$, otherwise add $R_f(Y^f)$ to $\mathbf{R}_f^+$;
5. Run the AddNode-Basic method for $\mathbf{R}_f^-$, and run D-IPFP for $\mathbf{R}_f^+$ until convergence;
6. Return the updated BN as output.

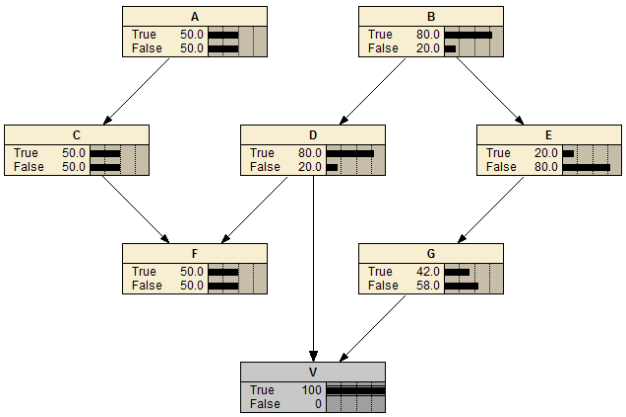
The convergence and correctness of the AddNode+Factorization method are established in Theorem 4.3 below.

Theorem 4.3 If constraints in $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$ are consistent with each other, then the AddNode+Factorization method converges to JPD $Q^*(X)$, where $\forall R_i(Y^i) \in \mathbf{R}$, $Q^*(Y^i) = R_i(Y^i)$.

Proof:

In step 3 of the AddNode+Factorization method, each constraint $R_j(Y^j)$ in $\mathbf{R}$ is factorized into $\prod_k R_j(Y_k^j \mid A_k^j)$. As long as each factor $R_j(Y_k^j \mid A_k^j)$ is satisfied, $R_j(Y^j)$ will be satisfied. Also, if $R_j(\{Y_k^j\} \cup A_k^j)$ is satisfied, $R_j(Y_k^j \mid A_k^j)$ will be satisfied. Step 3.2.2 adds $R_j(\{Y_k^j\} \cup A_k^j)$ to $\mathbf{R}_f$ if $\{Y_k^j\} \cup A_k^j$ is not a subset of the variable set of any constraints. Step 3.2.3 removes any constraint in $\mathbf{R}_f$ if its variable set is a subset of $\{Y_k^j\} \cup A_k^j$. These two steps guarantee that no constraint in $\mathbf{R}_f$ is satisfied more than once when later being integrated with the BN. Step 4 of the method groups the factorized constraints in $\mathbf{R}_f$ into structurally consistent constraint set $\mathbf{R}_f^+$ and structurally inconsistent constraint set $\mathbf{R}_f^-$ based on the dependencies on the dependency list. In this way the missing dependencies identified by the InconsId method in step 2 can be satisfied after the factorized constraints in $\mathbf{R}_f^-$ are integrated with the existing BN using the AddNode-Basic method in step 5. Since all the constraints factorized from $\mathbf{R}$ as well as all the dependencies in $\mathbf{R}$ are satisfied in the resulting BN at the end of step 5, all constraints in $\mathbf{R}$ are satisfied. Thus, the AddNode+Factorization method converges to joint probability distribution $Q^*(X)$, where $\forall R_i(Y^i) \in \mathbf{R}$, $Q^*(Y^i) = R_i(Y^i)$. □

The following example shows how the AddNode+Factorization method works. Given a BN in Figure 3.1 and four constraints in Figure 3.2, the method first runs the InconsId method to get $\mathbf{DL} = \{< D, G, \{B\} >, < D, G, \{E\} >\}$. After that it factorizes the four constraints into $R_1(B, D, F) = R_1(B) \cdot R_1(D) \cdot R_1(F)$, $R_2(B, D, G) = R_2(B) \cdot R_2(D) \cdot R_2(G \mid D)$, $R_3(D, E, G) = R_3(D) \cdot R_3(E) \cdot R_3(G \mid D)$ and $R_4(D, F, G) = R_4(D) \cdot R_4(F) \cdot R_4(G \mid D)$. At the end of step 3, $\mathbf{R}_f = \{R_1(B), R_1(F), R_2(D, G), R_3(E)\}$. Then it groups the constraints in $\mathbf{R}_f$ into structurally consistent constraints set $\mathbf{R}_f^+$ and structurally inconsistent constraints set $\mathbf{R}_f^-$ based on the dependencies in $\mathbf{DL}$. At the end of step 4, $\mathbf{R}_f^+ = \{R_1(B), R_1(F), R_3(E)\}$, $\mathbf{R}_f^- = \{R_2(D, G)\}$. Then it integrates $\mathbf{R}_f^-$ with the AddNode-Basic method and integrates $\mathbf{R}_f^+$ with D-IPFP until convergence. Figure 4.8 shows the resulting BN (a) together with the updated CPTs for the existing nodes in the BN and the resulting CPTs for the added nodes (b). Experiment shows that this BN satisfies all the four constraints.



(a) Resulting BN after integrating the four constraints.

B	
true	false
0.800	0.200

B	E	
	true	false
true	0.200	0.800
false	0.200	0.800

D	G	V	
		true	false
true	true	0.360	0.640
true	false	0.440	0.560
false	true	0.060	0.940
false	false	0.140	0.860

(b) Resulting CPTs of the existing nodes and the added nodes.

Figure 4.8 Result after running the AddNode+Factorization method

4.5 Experiments

In this section we will analyze the time performance for each AddNode method. We will also conduct experiments to empirically evaluate their performance, and see how their execution time are affected by the number of constraints and the size of the BN.

First we analyze the amount of changes the added nodes of the AddNode methods bring to the existing BN. Given a BN $G = (G_S, G_P)$ with all binary variables and a set of constraints $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$, let $R_i(Y^i)$ be any constraint in $\mathbf{R}$ and w_i be the number of variables in Y^i. Let $Y = Y^1 \cup Y^2 \cup \cdots \cup Y^m$ and w be the number of variables in Y. Let u be the number of inconsistent constraints in $\mathbf{R}$, $R_j(Y^j)$ be any inconsistent constraint in $\mathbf{R}$ and w_j be the number of variables in $R_j(Y^j)$. Let v be the number of inconsistent constraints in the set of the factorized constraints of $\mathbf{R}$, $R_k(Y^k)$ be any inconsistent constraint in the set of the factorized constraints of $\mathbf{R}$ and w_k be the number of variables in $R_k(Y^k)$. The changes to the existing BN made by the AddNode methods are summarized in Table 4.1.

Table 4.1 Summary of changes to the existing BN made by the AddNode methods

Method	# Added Nodes	# Added Links	# Added CPT Entries
AddNode-Basic	m	$\sum_{i}^{m} w_i$	$\sum_{i}^{m} 2^{w_i}$
AddNode+Merge	1	w	2^{w}
AddNode+D-IPFP	u	$\sum_{j}^{u} w_j$	$\sum_{j}^{u} 2^{w_j}$
AddNode+Factorization	v	$\sum_{k}^{v} w_k$	$\sum_{k}^{v} 2^{w_k}$

From the summary we can see that, among all the methods in the class of AddNode methods, AddNode+Merge adds the fewest nodes to the existing BN. AddNode+D-IPFP adds fewer nodes than AddNode-Basic when there are structurally consistent constraints. The number of nodes added by AddNode+Factorization depends on how many structurally inconsistent constraints are factorized from the constraint set. For the number of added links, compared to AddNode-Basic, AddNode+Merge adds fewer links when there are shared variables among the constraints. AddNode+D-IPFP adds fewer links when there are consistent constraints. AddNode+Factorization adds fewer links when there are structurally consistent parts in the constraints. For the number of added CPT entries, in general, AddNode+Merge adds the most CPT entries unless there are a significant number of shared variables among the constraints. To use AddNode+Merge in the case where the number of distinct variables in the constraint set is large, it is better to first divide all constraints into several groups according to some distance or relevance measure, then merge the constraints in each group into one constraint which contains a

moderate number of variables. Table 4.2 shows the modifications to the existing BN for the examples in this chapter.

Table 4.2 Modifications to the existing BN for the examples of AddNode methods

Method	# Added Nodes	# Added Links	# Added CPT Entries
AddNode-Basic	4	12	32
AddNode+Merge	1	5	32
AddNode+D-IPFP	2	6	16
AddNode+Factorization	1	2	4

Next we analyze the time performance for each method in the class of AddNode methods. For the AddNode-Basic method, it updates the belief of the entire BN at each iteration. The time complexity is equal to the BN inference method it uses for belief update. If it uses the Junction Tree method [46], the time complexity for one iteration of the AddNode-Basic method is exponential to the size of the largest clique in the junction tree of the existing BN. For the AddNode+Merge method, it updates the joint distribution of Y using IPFP and the time complexity for one iteration is exponential to $|Y|$. For the AddNode+D-IPFP method, the time complexity for the structurally inconsistent constraints is the same as that of the AddNode-Basic method. For the structurally consistent constraints, D-IPFP has the overhead of constructing and applying the structural constraint at each iteration, which is more computationally expensive than the AddNode-Basic method. For the AddNode+Factorization method, it has the overhead of

factorizing the constraints as well as constructing and applying the structural constraint for the structurally consistent parts of the constraints. Therefore, it is more computationally expensive than the AddNode-Basic method.

Finally, we will conduct experiments to evaluate the performance of the AddNode methods and see empirically how expensive they can be. The first set of experiments is designed to evaluate the execution time of each method when the number of constraints varies from 2 to 10. Each constraint in the constraint set has 3 variables. The summary for the constraints is in Table 4.3. The BN in this set of experiments has 15 binary variables. The result in Figure 4.9 shows the relation between the number of constraints in the constraint set and the execution time of each method when the size of the BN is fixed.

Table 4.3 Summary for the constraints in Experiment 1

# Constraints	# Distinct Variables in All Constraints	# Structurally Consistent Constraints	# Structurally Inconsistent Constraints	# Factorizable Constraints	Size of DL
2	5	1	1	1	4
4	8	2	2	1	5
6	10	2	4	1	10
8	11	2	6	1	15
10	11	2	8	1	23

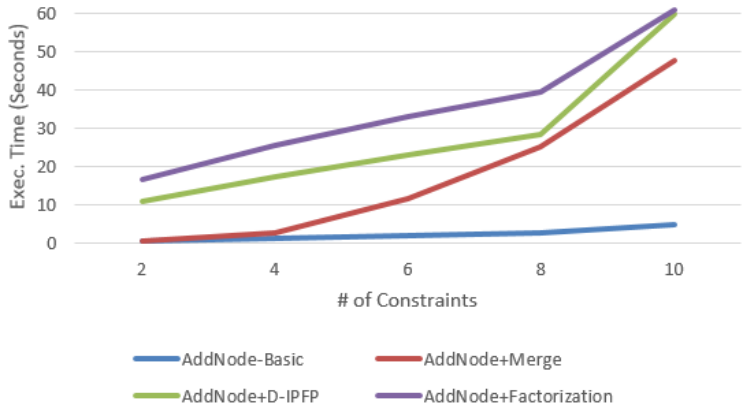


Figure 4.9 Result of Experiment 1 for the AddNode methods

The second set of experiments is designed to evaluate the execution time of each method when the number of binary variables in the BN varies from 20 to 80. This set of experiments uses 6 constraints, each of which has 3 variables. The dependency list in each experiment has 6 items. The result in Figure 4.10 shows the relation between the size of the BN and the execution time of each method when the number of constraints and the size of the dependency list are fixed.

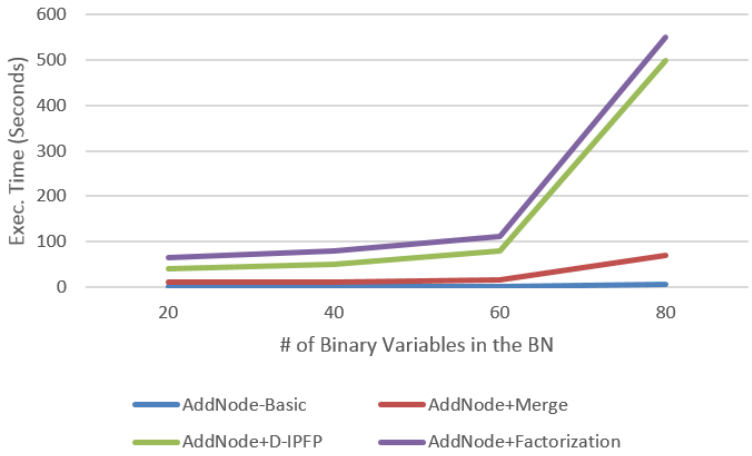


Figure 4.10 Result of Experiment 2 for the AddNode methods

From the above experiment results we can see that, the execution time of the AddNode-Basic method increases linearly when the number of constraints or the size of the BN increases. AddNode-Basic is faster than AddNode+Merge because in

AddNode+Merge the size of the CPT for the single added node grows exponentially with the number of distinct variables in the constraint set. AddNode-Basic is faster than AddNode+D-IPFP and AddNode+Factorization because the latter two methods involve IPFP for the structurally consistent constraints, and IPFP is the major contributing factor for the time performance of these two methods.

4.6 Summary

In this chapter we proposed the class of AddNode methods to overcome structural inconsistencies by adding nodes to the existing BN. These methods adapt the virtual evidence method originally developed for dealing with uncertain evidence in BN reasoning.

In Section 4.1 we introduced the basic method of this class, the AddNode-Basic method. For each constraint in the constraint set, it adds a node to the existing BN, makes the variables in the constraint as the parents of the added nodes and derives its CPT according to the likelihood rule of Pearl's virtual evidence method, then sets the state of the node to true. This process is iterated over all the constraints until convergence. After validating the concept of our idea with this basic method, we developed several variations of this method to balance the computational cost and solution quality in different situations, as well as to address some other concerns.

In Section 4.2 we introduced the AddNode+Merge method, which first merges all the constraints using IPFP, then adds a single node to the existing BN for the merged constraint. The main benefit of this method is to reduce the number of added nodes to one.

It can be used when it is computationally beneficial to merge small constraints before applying the AddNode-Basic method.

In Section 4.3 we introduced the AddNode+D-IPFP method, which integrates the structurally inconsistent constraints with the AddNode-Basic method and integrates the structurally consistent constraints with D-IPFP. In this way the number of nodes added to the existing BN can be reduced when there are structurally consistent constraints.

In Section 4.4 we introduced the AddNode+Factorization method, which first factorizes large constraints into smaller ones and adds nodes only for those that are structurally inconsistent with the given BN. The benefit of this method is two-folds. First, by replacing a large constraint with a number of smaller ones, the size of the CPTs for the added virtual nodes can be substantially reduced. Second, some of the small constraints from the factorization may be structurally consistent, which can be integrated without changing the BN structure.

In Section 4.5 we analyzed each method in the class of AddNode methods theoretically and conducted experiments to compare their performance in different situations. Experiments showed that compared to AddNode-Basic, other variations, while gaining some advantages, have their execution time increase much faster when the number of constraints or the size of the BN increases.

5 Overcome Structural Inconsistencies – AddLink Methods

In Chapter 4 we proposed the class of AddNode methods to overcome the structural inconsistencies by adding nodes to the existing BN. In this chapter we propose another class of methods, referred to as AddLink methods, to address the issue of structural inconsistencies.

In Section 5.1 we introduce the basic method of this class, named AddLink-Basic, to overcome the structural inconsistencies between a BN and a set of constraints. The idea of this method is to add one link for each dependency item on the dependency list in order to provide structural support for the missing dependencies.

In Section 5.2 we propose the AddLink-Prune method which seeks to minimize the number of links to be added to the existing BN. This is based on our observation that probabilistic dependencies are interrelated and many times adding one link may provide structural support for dependencies captured in several items on the dependency list. A truly global optimization requires searching a gigantic combinatorial space of all possible "missing" links of a given BN, which is computationally intractable for large BNs. Instead, our method focuses on the set of links that are added by our AddLink-Basic method, and tries to find a minimal subset that satisfies all the items on the dependency list.

In Section 5.3 we analyze the two AddLink methods and conduct experiments to evaluate their performance.

5.1 The AddLink-Basic Method

After running the InconsId method, all the dependencies that exist in the constraints but are missing in the BN are stored on the dependency list. The absence of these dependencies in the BN is the cause of the structural inconsistencies. To provide structural support for these missing dependencies, we can add a link $X_u \rightarrow X_v$ for each item $< X_u, X_v, W >$ on the dependency list to make up for the missing dependencies in the existing BN. Note here that as discussed in Subsection 3.2.3, nodes X_u and X_v are assumed to be in topological order according to the DAG of the BN, and X_u has lower index than X_v. After the links are added to the existing BN, there is no more structural inconsistency between the BN and the constraints. Then we can run either E-IPFP or D-IPFP to integrate the constraints with the updated BN. The following is the AddLink-Basic method.

AddLink-Basic ($G = (G_S, G_P)$, $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$)

1. Run the InconsId method with $G = (G_S, G_P)$ and **R** as input to obtain the dependency list **DL**;
2. For each item $< X_u, X_v, W >$ in **DL**, add a link $X_u \rightarrow X_v$ to G_S if it has not been added;
3. Run E-IPFP or D-IPFP with the modified BN and **R** as input;
4. Return the updated BN as output.

The following example illustrates how the AddLink-Basic method works. Given a BN in Figure 3.1 and six constraints in Figure 5.1, it first runs the InconsId method to get $\mathbf{DL} = \{< A, B, \varnothing >, < A, E, \varnothing >, < B, C, \varnothing >, < C, D, \varnothing >, < C, E, \varnothing >, < C, G, \varnothing >\}$. Then it adds links $A \to B$, $A \to E$, $B \to C$, $C \to D$, $C \to E$ and $C \to G$ to the existing BN to address the structural inconsistencies between the BN and the constraints. At the end, it runs E-IPFP to integrate the constraints with the updated BN. Figure 5.2 shows the resulting BN after running the AddLink-Basic method. Experiment shows that this BN satisfies all the six constraints. Besides, the I-divergence of the resulting JPD to the original JPD is 0.341.

A	B	$R_1(A,B)$
true	true	0.489
true	false	0.109
false	true	0.293
false	false	0.109

A	E	$R_2(A,E)$
true	true	0.340
true	false	0.258
false	true	0.227
false	false	0.175

B	C	$R_3(B,C)$
true	true	0.333
true	false	0.449
false	true	0.093
false	false	0.125

C	D	$R_4(C,D)$
true	true	0.171
true	false	0.256
false	true	0.229
false	false	0.344

C	E	$R_5(C,E)$
true	true	0.242
true	false	0.185
false	true	0.325
false	false	0.248

C	G	$R_6(C,G)$
true	true	0.229
true	false	0.197
false	true	0.308
false	false	0.265

Figure 5.1 Constraints $R_1(A,B), R_2(A,E), R_3(B,C), R_4(C,D), R_5(C,E)$ and $R_6(C,G)$

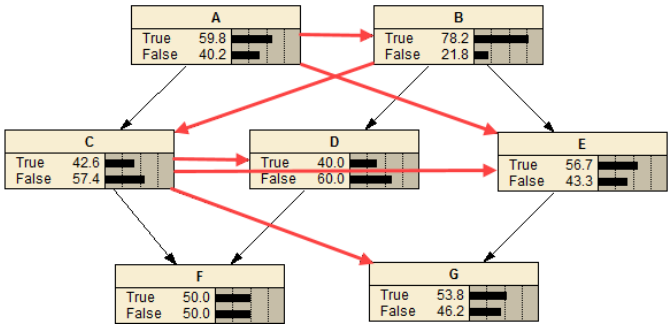


Figure 5.2 Resulting BN with added links colored red after running AddLink-Basic

5.2 The AddLink-Prune Method

The AddLink-Basic method adds one link for each item on the dependency list, which may introduce redundant links if one item is already covered by the link added for another item. To remove the redundancy and minimize the changes to the existing BN, we propose the AddLink-Prune method to search for the minimal set of links to add.

Informally speaking, this method works as follows. After adding a link between X_u and X_v for each item $< X_u, X_v, W >$ on the dependency list to a copy of the existing BN, the method uses depth-first search to find all the open paths between X_u and X_v in this BN for each dependency item $< X_u, X_v, W >$. A path is open between X_u and X_v if it is not blocked by W, which has been introduced in Subsection 3.2.2 for the d-separation method. Then the method creates a graph in a specific way with the set of added links on each open path as its vertices. Note that to avoid confusion, we use vertices and edges for the elements of this graph while we keep using nodes and links for the BN. At the end, the method uses depth-first branch-and-bound search [67] to find the path with minimum cost from the *Start* vertex to the *End* vertex in the graph. As will be shown shortly, a path from the *Start* vertex to the *End* vertex represents a solution to our problem as the added links in all the vertices along this path cover all the items on the dependency list. The cost of the path is measured by the total number of distinct added links in all the vertices along the path. Thus the path with minimum cost has the fewest links that can be added to the existing BN to remove all the items on the dependency list. Next we will explain how this method works with the BN in Figure 3.1 and the six constraints in Figure 5.1.

5.2.1 Identify the Candidate Link Sets

This method first runs the InconsId method to get the dependency list **DL** . In this example, $\mathbf{DL} = \{< A, B, \varnothing >, < A, E, \varnothing >, < B, C, \varnothing >, < C, D, \varnothing >, < C, E, \varnothing >, < C, G, \varnothing >\}$.

After that it creates a temp DAG by copying the DAG of the existing BN. Then for each item $< X_u, X_v, W >$ on the dependency list, it adds a link $X_u \to X_v$ to the temp DAG. In this example, links $A \to B$, $A \to E$, $B \to C$, $C \to D$, $C \to E$ and $C \to G$ are added to the temp DAG, which looks the same as the DAG in Figure 5.2.

Next, for each item $< X_u, X_v, W >$ on the dependency list, it uses depth-first search to find all the open paths from X_u to X_v in the temp DAG when variables in *W* are instantiated. The set of added links on the j^{th} open path for the i^{th} dependency item is denoted as L_i^j . As will be shown shortly, each L_i^j will serve as a vertex for the graph in which this method will search for the optimal solution. In this example, there is one open path for the first dependency item $< A, B, \varnothing >$, and the set of added links on this open path is denoted as $L_1^1 = \{A \to B\}$. There are three open paths for the second dependency item $< A, E, \varnothing >$, and the sets of added links on these open paths are denoted as $L_2^1 = \{A \to E\}$, $L_2^2 = \{C \to E\}$ and $L_2^3 = \{A \to B\}$. Similarly, the sets of added links on the open paths for the other dependency items are denoted as $L_3^1 = \{B \to C\}$, $L_3^2 = \{A \to B\}$, $L_4^1 = \{C \to D\}$, $L_4^2 = \{A \to B\}$, $L_4^3 = \{B \to C\}$, $L_5^1 = \{C \to E\}, L_5^2 = \{A \to E\}, L_5^3 = \{A \to B\}, L_5^4 = \{B \to C\}, L_6^1 = \{C \to G\}, L_6^2 = \{A \to B\}$, $L_6^3 = \{A \to E\}, L_6^4 = \{B \to C\}$ and $L_6^5 = \{C \to E\}$. Note that although all L_i^j s are singleton in this simple example, these vertices can contain multiple added links in general.

5.2.2 Construct a CLS Graph

In the next step of the method, a *CLS* (Candidate Link Set) graph is created. The CLS graph in Figure 5.3 is created for our example. We will use it to illustrate the process of how to create a CLS graph with all the candidate link sets obtained from Subsection 5.2.1.

This graph has a layered structure of $|\mathbf{DL}|+2$ levels, where $|\mathbf{DL}|$ denotes the size of the dependency list. Its first level has a single vertex *Start*, and the last level has a single vertex *End*. In our example the graph has 8 levels.

All vertices L_i^j are placed at the $(i+1)^{th}$ level of the graph, representing alternative ways of adding links to satisfy the i^{th} item in $\mathbf{DL}$. In our example, the vertex representing $L_1^1 = \{A \to B\}$ is placed at the second level of the graph, and the vertices representing $L_2^1 = \{A \to E\}$, $L_2^2 = \{C \to E\}$ and $L_2^3 = \{A \to B\}$ are placed at the third level of the graph, etc.

There is an edge from each vertex at the i^{th} level to each vertex at the $(i+1)^{th}$ level, so that a path from *Start* to *End* contains one vertex from each level of the graph. In this example, edges are added from *Start* to $L_1^1 = \{A \to B\}$, and from $L_1^1 = \{A \to B\}$ to $L_2^1 = \{A \to E\}$, $L_2^2 = \{C \to E\}$ and $L_2^3 = \{A \to B\}$, etc. Since each vertex at the $(i+1)^{th}$ level of the graph contains a set of added links that together can satisfy the i^{th} item in $\mathbf{DL}$, the union of the added links in all the vertices along the path from *Start* to *End* is able to satisfy all the items in $\mathbf{DL}$. That is, each path from *Start* to *End* provides a solution of what links to be added to the existing BN in order to overcome all the structural inconsistencies.

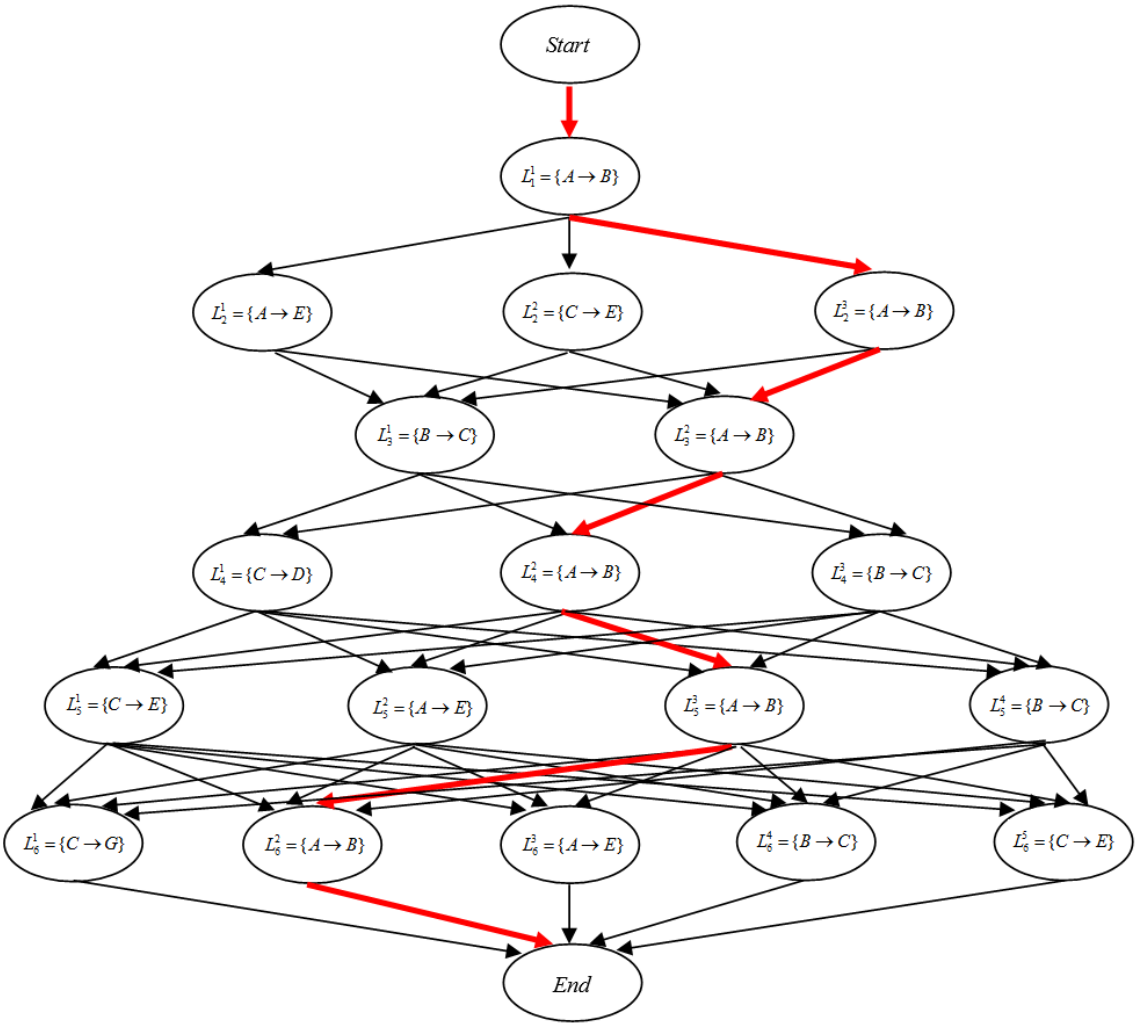


Figure 5.3 The CLS graph created with all the candidate link sets

5.2.3 The Cost of a Path in the CLS Graph

The cost of a path in the CLS graph is measured by the size of the union of the added links contained in each vertex along the path. In our example, the cost of the path $Start \rightarrow L_1^1 \rightarrow L_2^1 \rightarrow L_3^1 \rightarrow L_4^1 \rightarrow L_5^1 \rightarrow L_6^1 \rightarrow End$ is 6 because there are 6 distinct added links along this path, which are $A \rightarrow B$, $A \rightarrow E$, $B \rightarrow C$, $C \rightarrow D$, $C \rightarrow E$ and $C \rightarrow G$.

Our goal is then to find a solution path with minimum cost, i.e., a path containing the fewest distinct links to be added to the existing BN.

5.2.4 Search the Path with Minimum Cost

After the CLS graph is defined, the method uses depth-first branch-and-bound search to find the path with minimum cost from vertex *Start* to vertex *End* of the graph. The distinct links in the vertices on this path are the links to be added to the BN.

The bound is set to the minimum cost of all the paths found so far. In our example, the initial bound is set to 6, which is the size of the dependency list. After searching the first path $Start \rightarrow L_1^1 \rightarrow L_2^1 \rightarrow L_3^1 \rightarrow L_4^1 \rightarrow L_5^1 \rightarrow L_6^1 \rightarrow End$ whose cost is 6, the bound is not updated because the minimum cost of all the paths found so far is 6. After searching the second path $Start \rightarrow L_1^1 \rightarrow L_2^1 \rightarrow L_3^1 \rightarrow L_4^1 \rightarrow L_5^1 \rightarrow L_6^2 \rightarrow End$ whose cost is 5, the bound is updated to 5 because the minimum cost of all the paths found so far is 5.

Since each path from *Start* to *End* provides a solution of the necessary links to be added to overcome all the structural inconsistencies, we need to keep searching until the path with minimum cost is found in the CLS graph. In our example, the path with minimum cost is $Start \rightarrow L_1^1 \rightarrow L_2^3 \rightarrow L_3^2 \rightarrow L_4^2 \rightarrow L_5^3 \rightarrow L_6^2 \rightarrow End$, which is highlighted in red in Figure 5.3 and has the cost of 1. The set of added links on this path is $\{A \rightarrow B\}$.

We choose depth-first branch-and-bound search to find the optimal path because it can combine the space saving of the depth-first search with the heuristic information saved in the bound. Branch-and-bound helps cut the search space and guarantees the optimality of the final search result. Besides, since the depth of the CLS graph is $|\mathbf{DL}|+1$, depth-first search can be carried out with only limited space, which is desirable especially when the CLS graph is large. However, it may take a long time if the optimal path shows up at the very end of the search.

5.2.5 Integrate Constraints with the Updated BN

At the end, the method adds the distinct links on the path with minimum cost in the CLS graph to the existing BN, and runs E-IPFP or D-IPFP to integrate the constraints with the updated BN. In our example, the method adds $A \rightarrow B$ to the existing BN and runs E-IPFP to integrate the constraints with the updated BN. Figure 5.4 shows the resulting BN after running the AddLink-Prune method. Experiment shows that this BN satisfies all the six constraints. Besides, the I-divergence of the resulting JPD to the original JPD is 0.341, and the I-divergence of the resulting JPD to the resulting JPD of the AddLink-Basic method in Section 5.1 is 0.832.

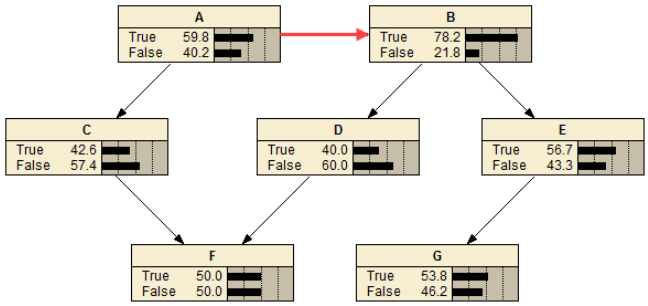


Figure 5.4 Resulting BN with added link colored red after running AddLink-Prune

5.2.6 Summary of the Method

As we can see from this example, compared to the AddLink-Basic method which adds six links to the existing BN, the AddLink-Prune method provides the same structural support for dependencies captured by the dependency list with only one added link. Also, the I-divergence of the resulting JPD to the original JPD of the AddLink-Prune method is the same as that of the AddLink-Basic method. Thus the AddLink-Prune method makes fewer changes to the original BN structure.

Besides, since the AddLink-Prune method focuses on the set of links that are added by the AddLink-Basic method, the search space only contains a set of six links for this example. If the method performs a truly global optimization, the search space will contain a set of fifteen links for this example. It will be computationally intractable when the size of the BN increases.

The following is the AddLink-Prune method:

AddLink-Prune ($G = (G_S, G_P)$, $\mathbf{R} = \{R_1(Y^1), \cdots, R_m(Y^m)\}$)

1. Run the InconsId method with $G = (G_S, G_P)$ and $\mathbf{R}$ as input to obtain dependency list $\mathbf{DL}$;
2. $G_S^t = G_S$. For each item $< X_u, X_v, W >$ in $\mathbf{DL}$, add a link $X_u \rightarrow X_v$ to G_S^t if it has not been added;
3. For each item $< X_u, X_v, W >$ in $\mathbf{DL}$, use depth-first search in G_S^t to find all the open paths from X_u to X_v when variables in W are observed. The set of added links on the j^{th} open path for the i^{th} dependency item is denoted as L_i^j ;
4. Create a CLS graph and use the depth-first branch-and-bound search to find the path with minimum cost from *Start* vertex to *End* vertex in the graph, where the cost of the path is measured by the number of distinct links in all the vertices along the path, and the bound is set to the minimum cost of all the paths found so far;
5. Add the distinct links on the path with minimum cost in the CLS graph to G_S , and run E-IPFP or D-IPFP to integrate $\mathbf{R}$ with the updated BN;
6. Return the updated BN as output.

5.3 Experiments

In this section we will analyze the two AddLink methods and conduct experiments to evaluate their performance. First, we analyze the changes brought to the existing BN by the AddLink methods. Both of the AddLink methods add extra links to the existing BN. The AddLink-Basic method adds one link for each dependency item if the link has not been added. Compared to the AddLink-Basic method, the AddLink-Prune method may add fewer links to the existing BN, which may also reduce the number of added CPT entries to the existing BN. In addition, the likelihood of increasing the maximum clique size of the BN may also be reduced with fewer added links. Thus, the resulting BN of the AddLink-Prune method may have better performance for BN reasoning. Table 5.1 shows the modifications to the existing BN for the examples in this chapter.

Table 5.1 Modifications to the existing BN for the examples of AddLink methods

Method	# Added Links	# Added CPT Entries	# Entries of the Largest CPT
AddLink-Basic	6	13	8
AddLink-Prune	1	1	4

Next we will conduct experiments to evaluate the performance of the AddLink methods and see empirically how expensive they can be. The input data in these experiments is the same as what is used in the experiments to evaluate the time performance of the AddNode methods in Section 4.5. We will use D-IPFP to integrate the constraints with the updated BN since D-IPFP works faster for large BNs.

The first set of experiments is designed to evaluate the execution time of each method when the number of constraints varies from 2 to 10. Each constraint in the constraint set has 3 variables. The BN in this set of experiments has 15 binary variables. The result in Figure 5.5 shows the relation between the number of constraints and the execution time of each method when the size of the BN is fixed. The size of the dependency list is also provided for each set of constraints.

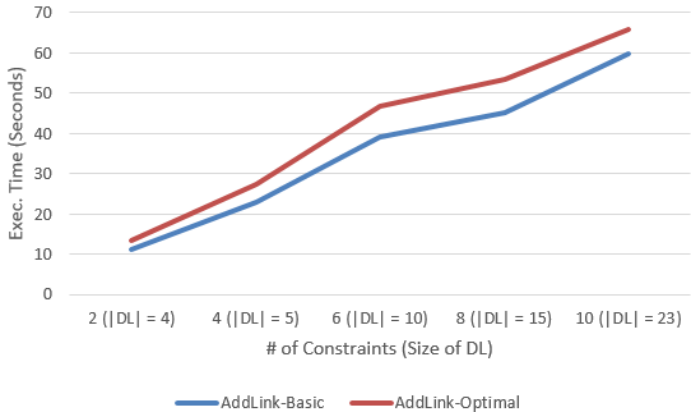


Figure 5.5 Result of Experiment 1 for the AddLink methods

The second set of experiments is designed to evaluate the execution time of each method when the number of binary variables in the BN varies from 20 to 80. This set of experiments uses 6 constraints, each of which has 3 variables. The size of the dependency list in each experiment is 6. The result in Figure 5.6 shows the relation between the size of the BN and the execution time of each method when the number of constraints and the size of the dependency list are fixed.

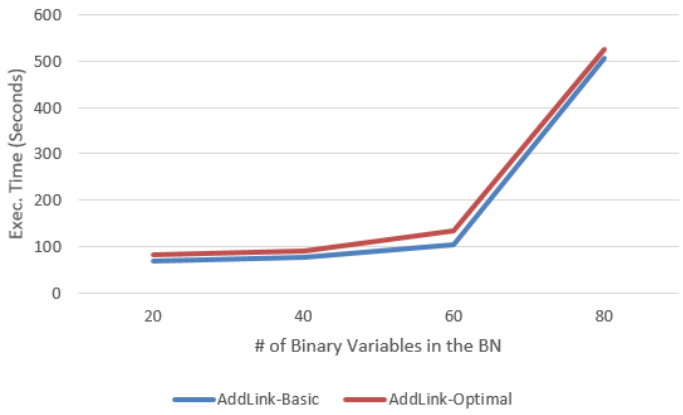


Figure 5.6 Result of Experiment 2 for the AddLink methods

From the experiment results we can see that the execution time of both methods increase linearly with the number of constraints and increase exponentially with the size of the BN. The difference of the execution time between the two methods is very small in both experiments. This is because both methods involve IPFP of similar complexity (same set of constrains and similar BN), and IPFP takes most of the computing time of these methods. Besides, the size of the dependency list is only six in the second set of experiments. This makes the CLS graph very small and the search for the optimal link set does not take much time in the AddLink-Prune method. If the size of the dependency list is large, the difference of the execution time between these two methods will be more obvious because the search for the added links in the AddLink-Prune method will take much more time compared to the AddLink-Basic method.

5.4 Summary

In this chapter we proposed the methods to overcome the structural inconsistencies by adding links to the existing BN. The class of methods is referred to as AddLink methods, which includes the AddLink-Basic method and the AddLink-Prune method.

The AddLink-Basic method adds one link for each item on the dependency list if the link has not been added. The added links provide structural support for the missing dependencies in the existing BN, and the constraints can then be integrated into the updated BN using E-IPFP or D-IPFP.

The AddLink-Prune method is developed to reduce the links to be added to the existing BN. It uses depth-first search to find all the candidate sets of added links and defines a CLS graph based on these sets. Then it uses depth-first branch-and-bound search to find the path with minimum cost in the graph, which represents the fewest links to be added to the existing BN in order to overcome all the structural inconsistencies.

In Section 5.3 we conducted experiments to evaluate the performance of the two AddLink methods. The experiment results showed that the execution time of both methods increase linearly with the number of constraints and increase exponentially with the size of the BN. The difference of the execution time between these two methods is very small in our experiments when the methods run under the same conditions.

6 Construct a Large BN from a Set of Small BNs

In this chapter we extend the framework and the methods we developed in the previous chapters to solve the problem of constructing a large BN from a set of small BNs.

In Section 6.1 we introduce the problem of constructing a large BN from a set of small BNs. To illustrate the problem, we use the Insurance BN in the BN repository [88] as an example. This BN is designed for evaluating car insurance risks. We split it into three small BNs, each for one kind of insurance cost. Meanwhile, a set of integration constraints is provided which represents the dependencies between the variables of the three small BNs.

In Section 6.2 we first formulate the problem of constructing a large BN from a set of small BNs as our knowledge integration problem, then using the Insurance network example we describe the process of merging small BNs into a large BN by applying the InconsId method and one variation of either the AddNode or AddLink methods.

In Section 6.3 we compare the performance of the AddNode and AddLink methods when applying them to merging small BNs into a large BN. We also compare the performance of the merged large BNs resulting from these methods.

6.1 The Problem of Constructing a Large BN from a set of Small BNs

Imagine that a group of experts are assigned a task to build a probabilistic model in the form of a BN for a certain domain. Each expert only has partial knowledge about the entire domain and is able to build a BN for the part of the domain that he or she is

familiar with. To integrate all their knowledge, the experts must communicate with each other in order to identify the dependencies among the small BNs they have built. This results in a set of probabilistic constraints that the merged BN should comply with. Now the problem is how to merge these small BNs into a large BN for the entire domain that satisfies these integration constraints.

To explain this problem with a concrete example, we take the Insurance network from the BN repository and split it into three small BNs, one for medical cost, one for liability cost and one for property cost. Each small BN is considered to be built by an expert of that sub-domain. To simplify the representation of the BN, we modified the original Insurance network to make all nodes as binary variables. Figure 6.1 shows the modified Insurance network whose CPTs are set manually. We will use this BN as the baseline BN to compare with the resulting BNs after merging the three small BNs with the AddNode and AddLink methods. Figure 6.2 shows the three smalls BNs, obtained by removing some links from the large BN. Their CPTs are updated automatically by Netica after those links are removed. In this example, the small BNs are disconnected and do not have any shared variables. We will discuss the potential issues when the BNs to be merged have shared variables later.

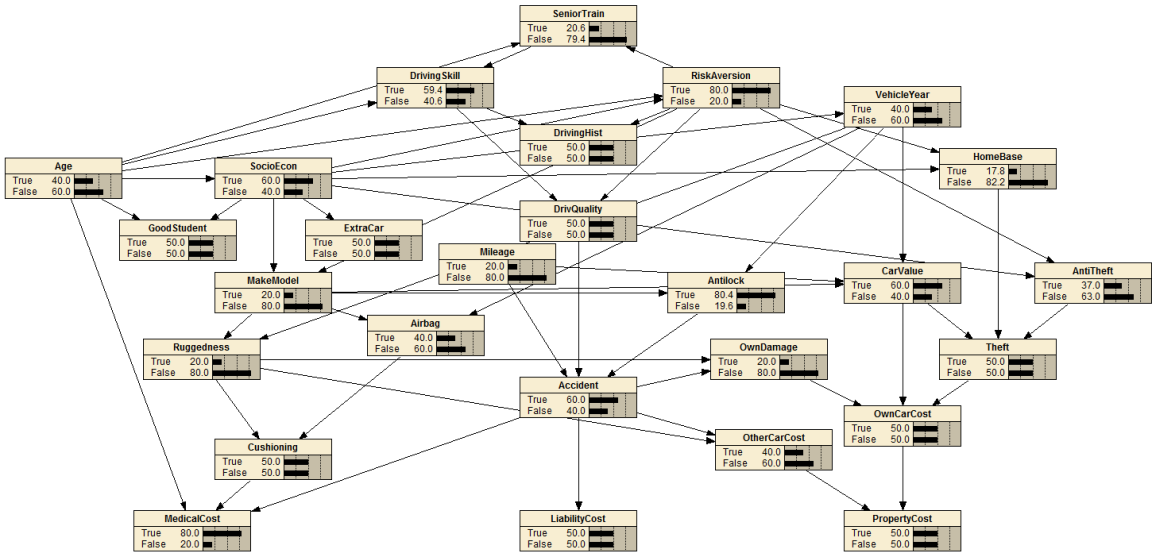
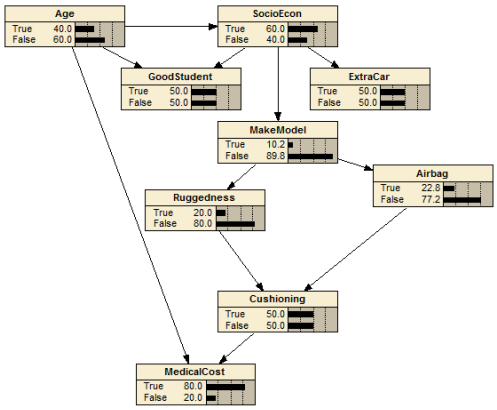
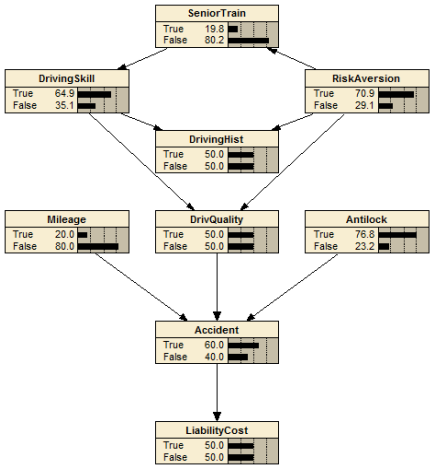


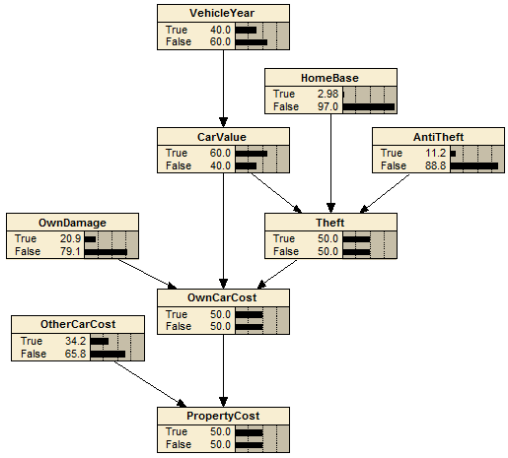
Figure 6.1 The modified Insurance network



(a) Small BN for medical cost.



(b) Small BN for liability cost.



(c) Small BN for property cost.

Figure 6.2 Three small BNs split from the Insurance network

Figure 6.3 shows the links that have been removed from Figure 6.1 when splitting it into three small BNs.

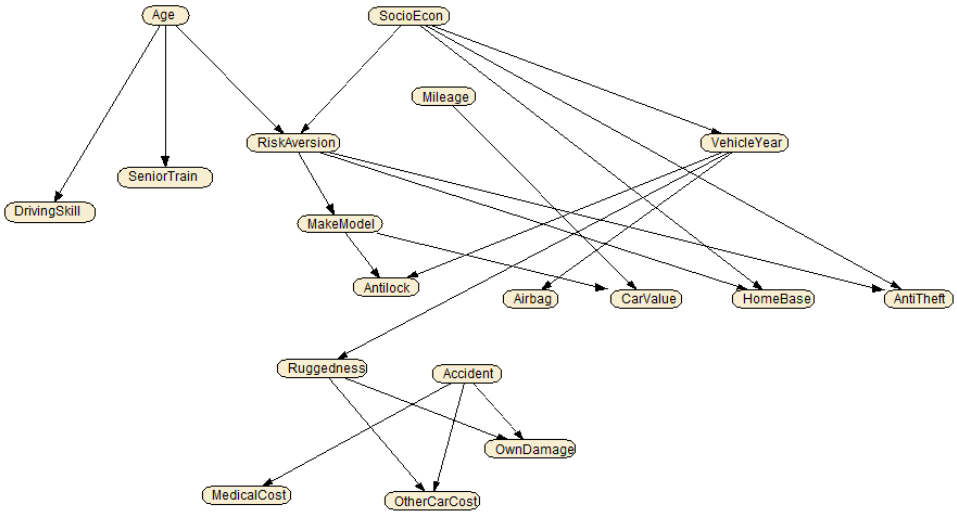


Figure 6.3 Links removed from the Insurance network when splitting it

In real applications, the dependencies between variables among these component BNs can be obtained from the communication and consensus building among the experts of the related subdomains and represented in the form of constraints called integration constraints. The quality of these constraints (e.g., their accuracy and thoroughness) will

affect the quality of the merged BN. Here we skip this step and assume a good set of constraints exist so that we can focus on the operation of merging the given small BNs. Therefore in our example we artificially generate ten constraints, given in Figure 6.4, each of which is a marginal probability distribution of four variables from the baseline BN in Figure 6.1. Moreover, as can be seen from Figures 6.3 and 6.4, each constraint encodes the direct dependencies among the four variables related by the adjacent links removed from the baseline BN. For example, R_1 corresponds to three adjacent links "Age $\rightarrow$ RiskAversion", "Age $\rightarrow$ SenoirTrain", and "Age $\rightarrow$ DrivingSkill", and it encodes the direct dependencies between the four variables involved. However, not all the dependencies and their strength lost with the split of the baseline BN are encoded in these ten constraints. For example, "RiskAversion" and "Ruggedness" are dependent given "MakeModel" in Figure 6.1. But this dependency is not encoded in any of the ten constraints. More constraints can be generated to encode all the removed dependencies and their strength. However, to make this example simple and focused, we will limit the dependencies to be recovered in the merged BN to those in the ten constraints. Also, in real-world situations it is common that not all dependencies among the small BNs can be identified at once. This setting will also allow us to see how the incomplete set of dependencies provided by the integration constraints will affect the integration quality.

Age	RiskAversion	SeniorTrain	DrivingSkill	R_1
true	true	true	true	0.0006
true	true	true	false	0.0078
true	true	false	true	0.2182
true	true	false	false	0.0573
true	false	true	true	0.0049
true	false	true	false	0.0657
true	false	false	true	0.0361
true	false	false	false	0.0095
false	true	true	true	0.0336
false	true	true	false	0.0609
false	true	false	true	0.2575
false	true	false	false	0.1639
false	false	true	true	0.0116
false	false	true	false	0.0211
false	false	false	true	0.0313
false	false	false	false	0.0199

SocioEcon	RiskAversion	HomeBase	AntiTheft	R_2
true	true	true	true	0.0016
true	true	true	false	0.0127
true	true	false	true	0.0521
true	true	false	false	0.4135
true	false	true	true	0.0634
true	false	true	false	0.0099
true	false	false	true	0.0405
true	false	false	false	0.0063
false	true	true	true	0.0283
false	true	true	false	0.0305
false	true	false	true	0.1257
false	true	false	false	0.1355
false	false	true	true	0.0227
false	false	true	false	0.0085
false	false	false	true	0.0355
false	false	false	false	0.0133

VehicleYear	Ruggedness	OtherCarCost	OwnDamage	R_3
true	true	true	true	0.0124
true	true	true	false	0.0148
true	true	false	true	0.0048
true	true	false	false	0.0481
true	false	true	true	0.0293
true	false	true	false	0.1035
true	false	false	true	0.0335
true	false	false	false	0.1536
false	true	true	true	0.0186
false	true	true	false	0.0222
false	true	false	true	0.0072
false	true	false	false	0.0721
false	false	true	true	0.0440
false	false	true	false	0.1552
false	false	false	true	0.0503
false	false	false	false	0.2304

Accident	OtherCarCost	OwnDamange	MedicalCost	R_4
true	true	true	true	0.0501
true	true	true	false	0.0125
true	true	false	true	0.1420
true	true	false	false	0.0355
true	false	true	true	0.0460
true	false	true	false	0.0115
true	false	false	true	0.2421
true	false	false	false	0.0605
false	true	true	true	0.0334
false	true	true	false	0.0083
false	true	false	true	0.0946
false	true	false	false	0.0236
false	false	true	true	0.0306
false	false	true	false	0.0077
false	false	false	true	0.1613
false	false	false	false	0.0403

SocioEcon	RiskAversion	MakeModel	VehicleYear	R_5
true	true	true	true	0.0065
true	true	true	false	0.0097
true	true	false	true	0.1856
true	true	false	false	0.2780
true	false	true	true	0.0323
true	false	true	false	0.0484
true	false	false	true	0.0158
true	false	false	false	0.0236
false	true	true	true	0.0262
false	true	true	false	0.0394
false	true	false	true	0.1016
false	true	false	false	0.1528
false	false	true	true	0.0148
false	false	true	false	0.0222
false	false	false	true	0.0172
false	false	false	false	0.0259

RiskAversion	MakeModel	Antilock	CarValue	R_6
true	true	true	true	0.0404
true	true	true	false	0.0269
true	true	false	true	0.0087
true	true	false	false	0.0058
true	false	true	true	0.3545
true	false	true	false	0.2365
true	false	false	true	0.0762
true	false	false	false	0.0508
false	true	true	true	0.0514
false	true	true	false	0.0343
false	true	false	true	0.0192
false	true	false	false	0.0128
false	false	true	true	0.0360
false	false	true	false	0.0240
false	false	false	true	0.0135
false	false	false	false	0.0090

MakeModel	VehicleYear	Antilock	Airbag	R_7
true	true	true	true	0.0026
true	true	true	false	0.0586
true	true	false	true	0.0008
true	true	false	false	0.0178
true	false	true	true	0.0639
true	false	true	false	0.0279
true	false	false	true	0.0194
true	false	false	false	0.0085
false	true	true	true	0.0648
false	true	true	false	0.1956
false	true	false	true	0.0149
false	true	false	false	0.0449
false	false	true	true	0.1899
false	false	true	false	0.2007
false	false	false	true	0.0436
false	false	false	false	0.0461

RiskAversion	MakeModel	Mileage	CarValue	R_8
true	true	true	true	0.0098
true	true	true	false	0.0065
true	true	false	true	0.0393
true	true	false	false	0.0262
true	false	true	true	0.0861
true	false	true	false	0.0574
true	false	false	true	0.3446
true	false	false	false	0.2299
false	true	true	true	0.0141
false	true	true	false	0.0094
false	true	false	true	0.0565
false	true	false	false	0.0377
false	false	true	true	0.0099
false	false	true	false	0.0066
false	false	false	true	0.0396
false	false	false	false	0.0264

SocioEcon	Mileage	Antilock	CarValue	R_9
true	true	true	true	0. 0578
true	true	true	false	0. 0386
true	true	false	true	0. 0141
true	true	false	false	0. 0094
true	false	true	true	0. 2315
true	false	true	false	0. 1544
true	false	false	true	0. 0564
true	false	false	false	0. 0376
false	true	true	true	0. 0386
false	true	true	false	0. 0257
false	true	false	true	0. 0094
false	true	false	false	0. 0063
false	false	true	true	0. 1544
false	false	true	false	0. 1030
false	false	false	true	0. 0376
false	false	false	false	0. 0251

MakeModel	VehicleYear	Accident	OwnDamage	R_{10}
true	true	true	true	0.0096
true	true	true	false	0.0383
true	true	false	true	0.0064
true	true	false	false	0.0255
true	false	true	true	0.0144
true	false	true	false	0.0575
true	false	false	true	0.0096
true	false	false	false	0.0383
false	true	true	true	0.0385
false	true	true	false	0.1537
false	true	false	true	0.0256
false	true	false	false	0.1024
false	false	true	true	0.0577
false	false	true	false	0.2306
false	false	false	true	0.0384
false	false	false	false	0.1536

Figure 6.4 Ten integration constraints for the small BNs

6.2 Merge the Small BNs into a Large BN

Now we formulate the problem of merging small BNs into a large BN as our knowledge integration problem so that our methods of identifying and overcoming structural inconsistencies can be applied. This can be done by simply forming a BN $\tilde{G}$ which contains the given small BNs as disconnected components and then integrating the constraints representing the interdependencies among the small BNs into $\tilde{G}$. In our example, $\tilde{G}$ contains the three disconnected components of Figure 6.2 (a), (b), and (c), and **R** contains the ten constraints. The integration of **R** into $\tilde{G}$ starts by applying the InconsId method, which results in a dependency list of 88 items. These items represent the dependencies that exist in the constraints but are missing in $\tilde{G}$. Note that among the ten constraints, R_9 and R_{10} are structurally consistent with $\tilde{G}$. The other eight constraints are structurally inconsistent with $\tilde{G}$.

Next we can use either the AddNode or AddLink methods to integrate $\mathbf{R}$ into $\tilde{G}$ and thus connect the three components into a connected large BN. The following are the results we get from these methods.

After applying the AddNode-Basic method, the small BNs are merged into the large BN in Figure 6.5. 10 nodes have been added in it, one for each constraint.

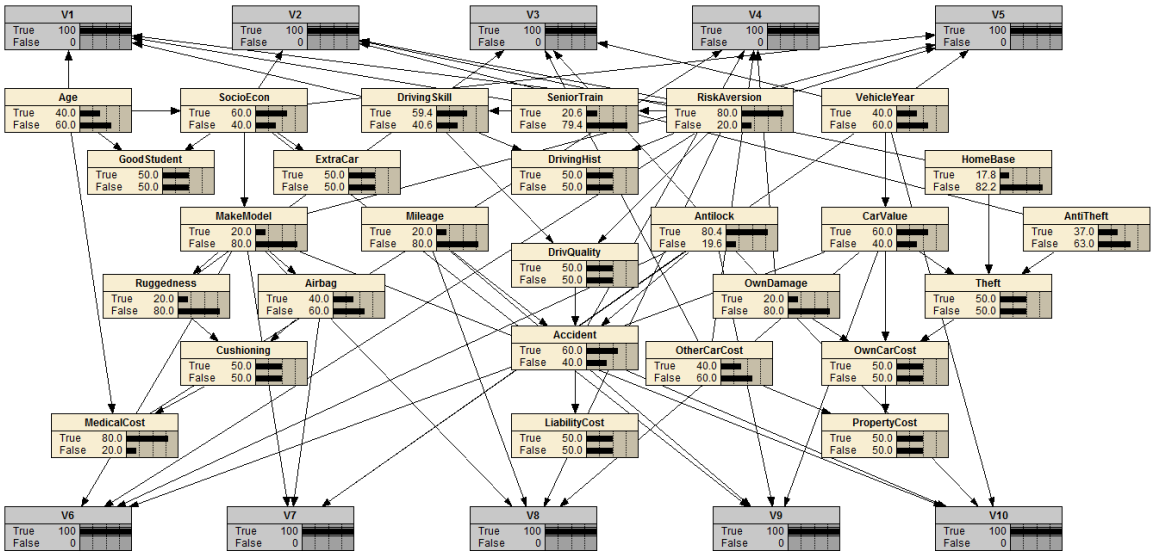


Figure 6.5 Result after running the AddNode-Basic method

Since the number of distinct variables is too large in the constraint set, it is not appropriate to merge all the constraints into one constraint with the AddNode+Merge method. To solve the problem, we combine the idea of the AddNode+Merge method with the AddNode-Basic method. We first divide the ten constraints into five groups, each of which contains two constraints. Then we merge the constraints in each group into one constraint, and use AddNode-Basic to merge the small BNs into a large BN with the merged constraints. The resulting BN is shown in Figure 6.6. Here 5 nodes have been added, one for each merged constraint.

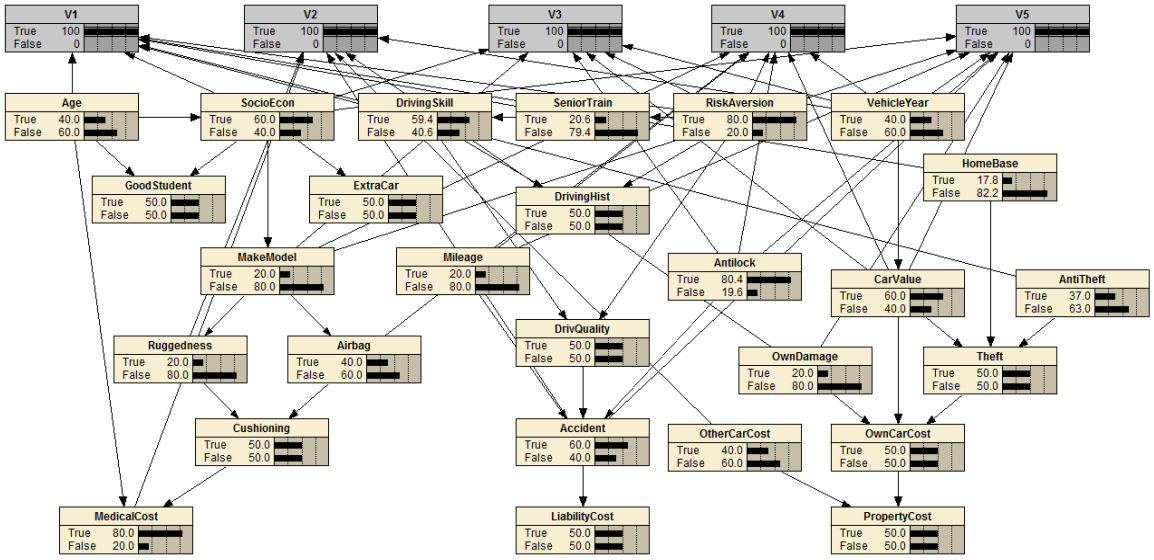


Figure 6.6 Result after running AddNode-Basic with the merged constraints

After applying the AddNode+D-IPFP method, the small BNs are merged into the large BN in Figure 6.7. 8 nodes have been added, one for each structurally inconsistent constraint. The 2 structurally consistent constraints are integrated using D-IPFP.

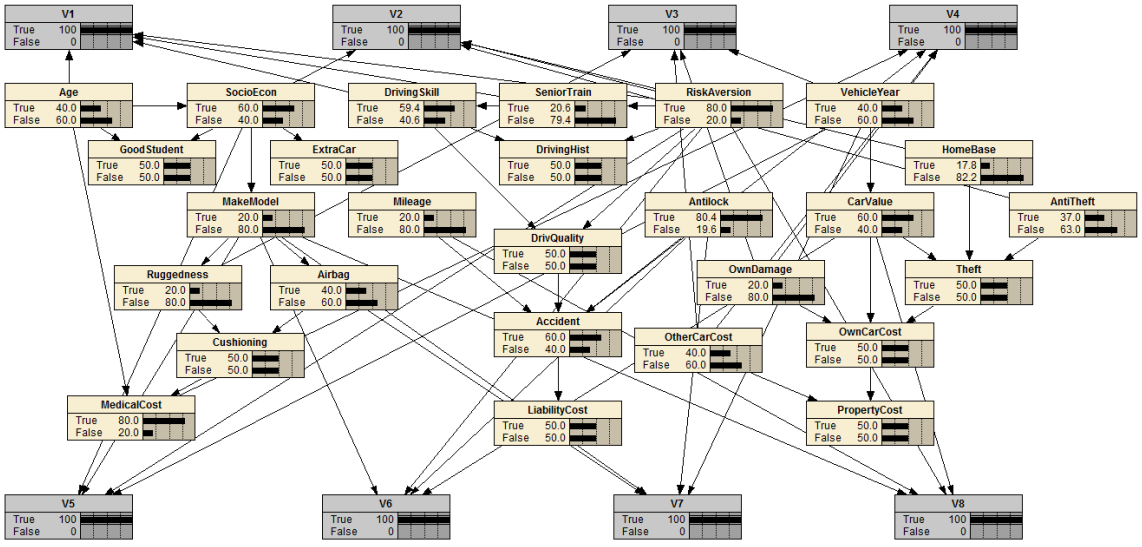


Figure 6.7 Result after running the AddNode+D-IPFP method

After applying the AddNode+Factorization method, the small BNs are merged into the large BN in Figure 6.8. The ten constraints are factorized into 13 constraints, out of which 10 are structurally inconsistent constraints and 3 are structurally consistent

constraints. Therefore, 10 nodes have been added in the resulting BN. The 3 structurally consistent constraints are integrated using D-IPFP.

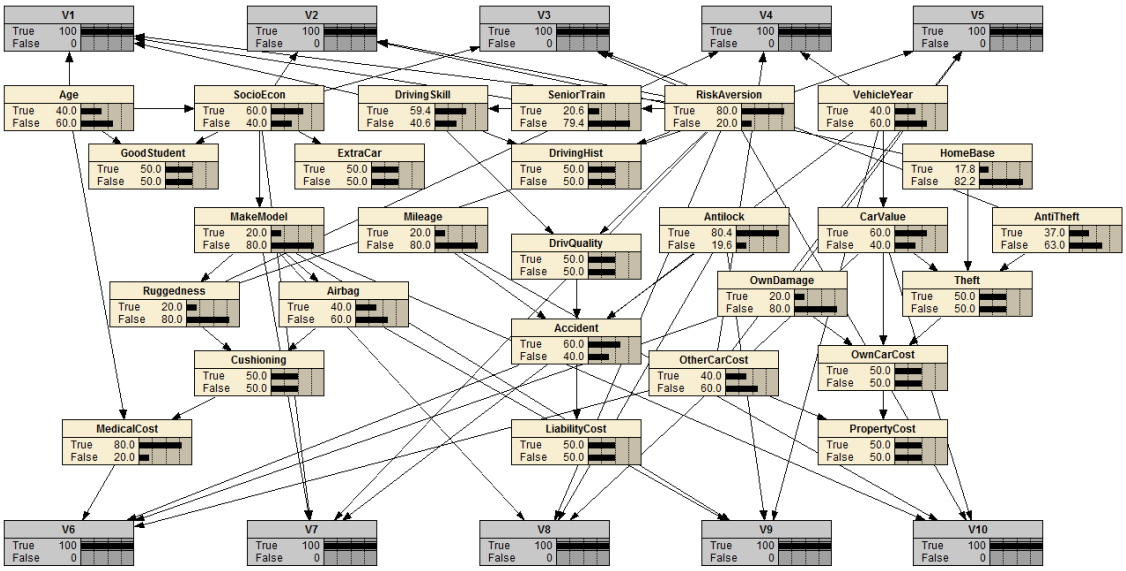


Figure 6.8 Result after running the AddNode+Factorization method

After applying the AddLink-Basic method, the small BNs are merged into the large BN in Figure 6.9. 21 links have been added in the resulting BN because there are 21 distinct pairs of X_u and X_v for all the $< X_u, X_v, W >$ in **DL**. The constraints are integrated using D-IPFP after adding links to the existing BN.

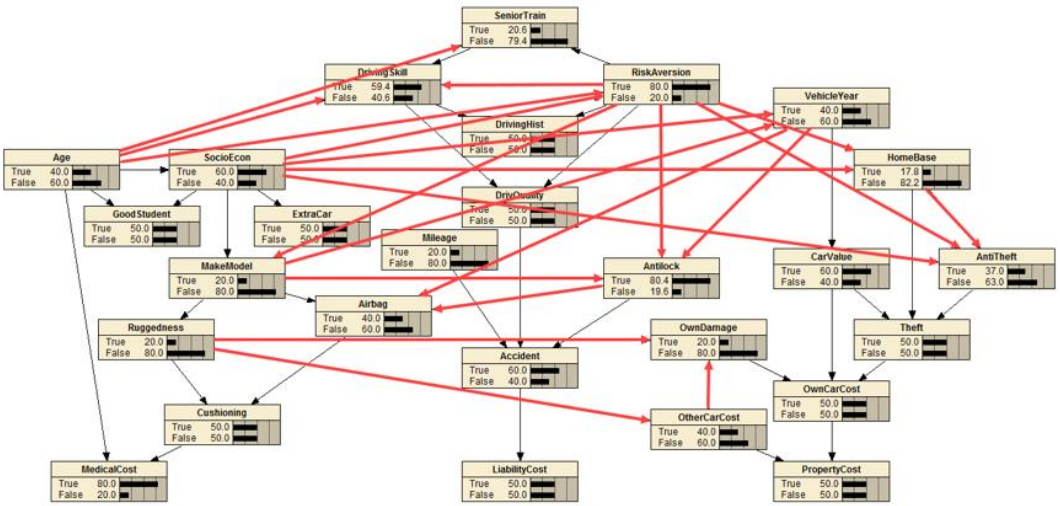


Figure 6.9 Result after running the AddLink-Basic method

After applying the AddLink-Prune method, the small BNs are merged into the large BN in Figure 6.10. 14 links have been added in the resulting BN. The constraints are integrated using D-IPFP after adding links to the existing BN.

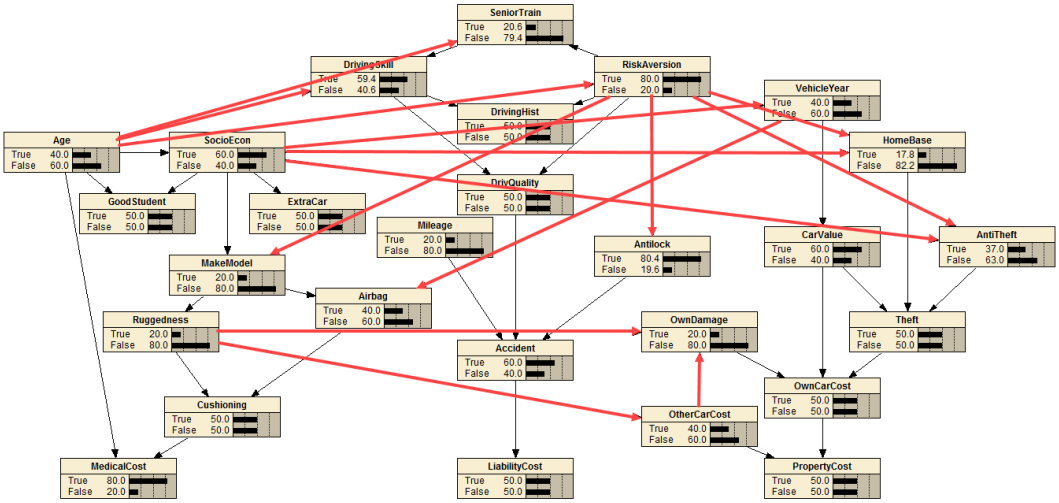


Figure 6.10 Result after running the AddLink-Prune method

From the above results we can see that a large BN is constructed from the three small BNs after applying each of the knowledge integration methods in our framework. Experiments show that each of these large BNs complies with the given constraints. After the integration, comprehensive car insurance analysis can be performed using the large BN for all kinds of insurance costs, including medical cost, liability cost, as well as property cost.

In this example there are no shared variables among the small BNs. If there are any, their marginal distributions may be inconsistent with each other. There are several solutions to deal with this issue. For example, a weight of trust can be assigned to each small BN and the marginal distribution in the small BN with higher weight will be adopted. Alternatively, we can first use SMOOTH to reach a compromise among the inconsistent marginal distributions, then apply it as a constraint during the integration.

As reviewed in Section 2.6, inter-subnet communication methods of MSBN can also be used to connect small BNs of subdomains to support probabilistic reasoning over a large domain. However, their communication methods impose very restrictive requirements on the small BNs, i.e., the shared variables among the small BNs need to be identical and all parents of a shared variable must appear in one subnet. If these requirements are met, then the small BNs can be organized into a hypertree which represents how they communicate with each other. The communication is done through exchanging beliefs of the small BNs over their shared variables, each of which needs to be a d-sepset. Such restrictions have limited the application of MSBN when there are inconsistencies among the small BNs. In contrast, our approach does not impose any restrictions on these small BNs. It works as long as a good set of integration constraints that captures the dependencies among these BNs can be obtained. This great flexibility allows our methods to have a wider scope of applications in automatically constructing a large probabilistic knowledge base from several smaller knowledge bases represented as BNs.

6.3 Comparison of the AddNode and AddLink Methods

In this section we compare the AddNode and AddLink methods based on their performance in integrating the three small BNs with the ten constraints in Section 6.2. The differences between the resulting BNs and the existing BN are summarized in Table 6.1. From the summary we can see that the AddNode methods add extra nodes to the existing BN. The number of links and CPT entries added by the AddNode methods is usually higher than those added by the AddLink methods.

Table 6.1 Modifications to the existing BN by the AddNode and AddLink methods

Method	# Added Nodes	# Added Links	# Added CPT Entries
AddNode-Basic	10	40	160
AddNode+Merge	5	34	640
AddNode+D-IPFP	8	32	108
AddNode+Factorization	10	36	128
AddLink-Basic	0	21	42
AddLink-Prune	0	14	21

Next we compare the time performance for the AddNode and AddLink methods. Figure 6.11 shows the execution time of each method when performing the integration task in Section 6.2. It can be seen that the execution time of AddNode+Merge is highest because the sizes of the CPTs for the added nodes are too large after the constraints are merged. The execution time of the other AddNode methods are lower than that of the AddLink methods. This is because the two AddLink methods involve IPFP for all constraints while AddNode+D-IPFP and AddNode+Factorization only involve IPFP for some constraints, and IPFP is the major contributing factor for the time performance of these methods.

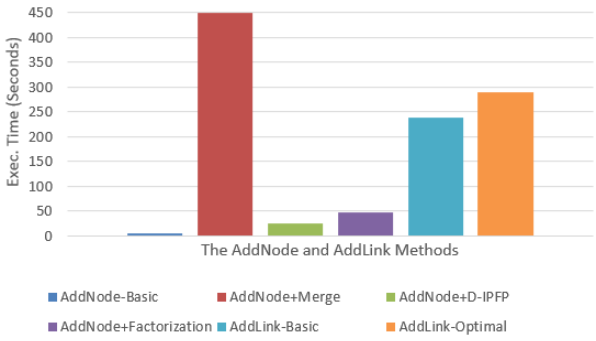
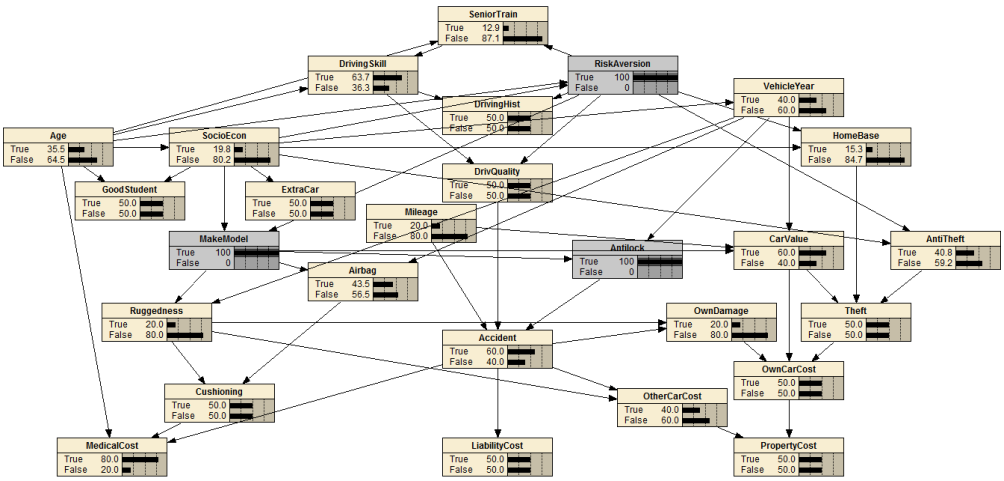
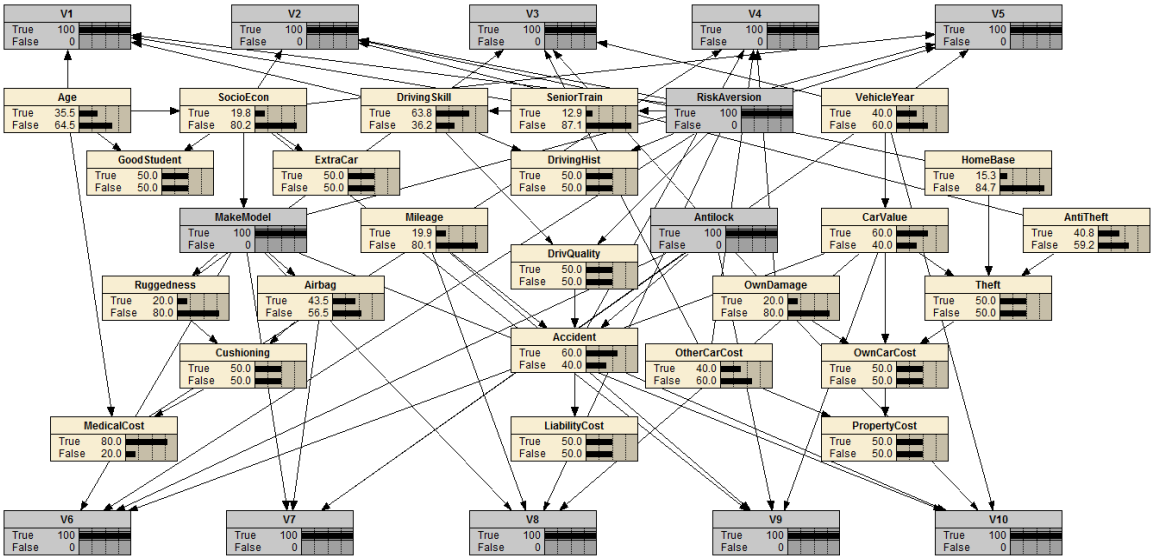


Figure 6.11 Execution time of the AddNode and AddLink methods

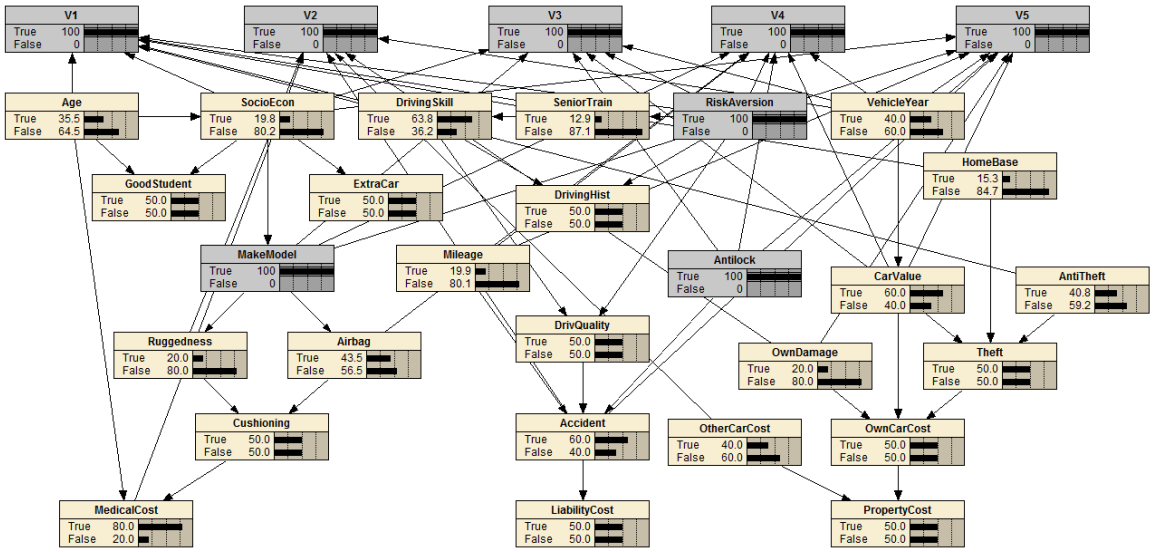
Lastly, we compare the performance of the probabilistic inference tasks for the large BNs merged by the AddNode and AddLink methods in Section 6.2. The inference task is also performed on the baseline BN shown in Figure 6.1. The first inference task is about belief update with three evidences: "MakeModel = true", "RiskAversion = true" and "Antilock = true". Figure 6.12 shows their inference results for task 1 for each of these BNs.



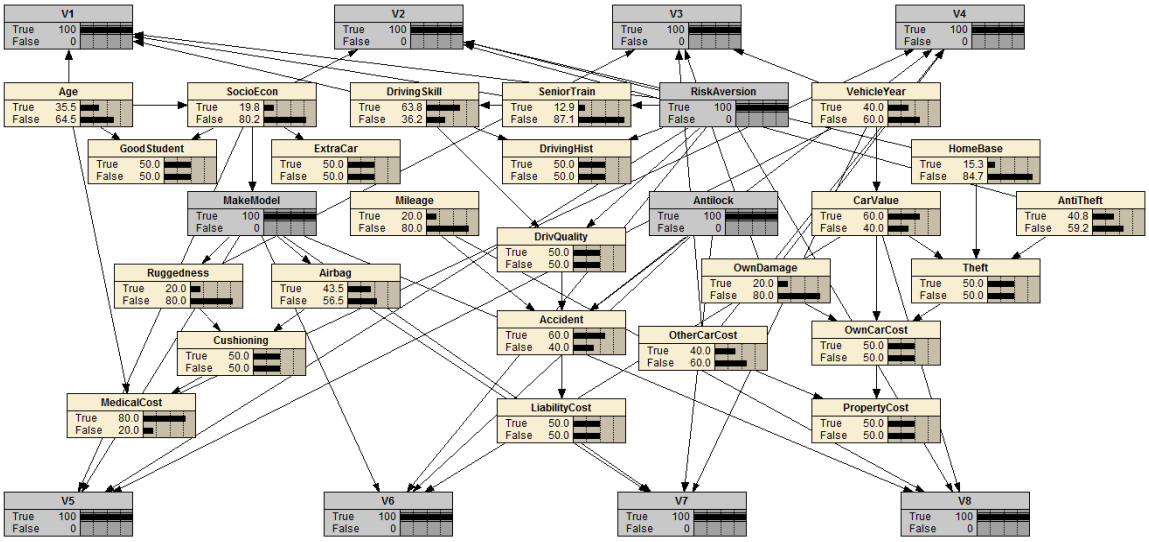
(a) Inference result of the baseline BN.



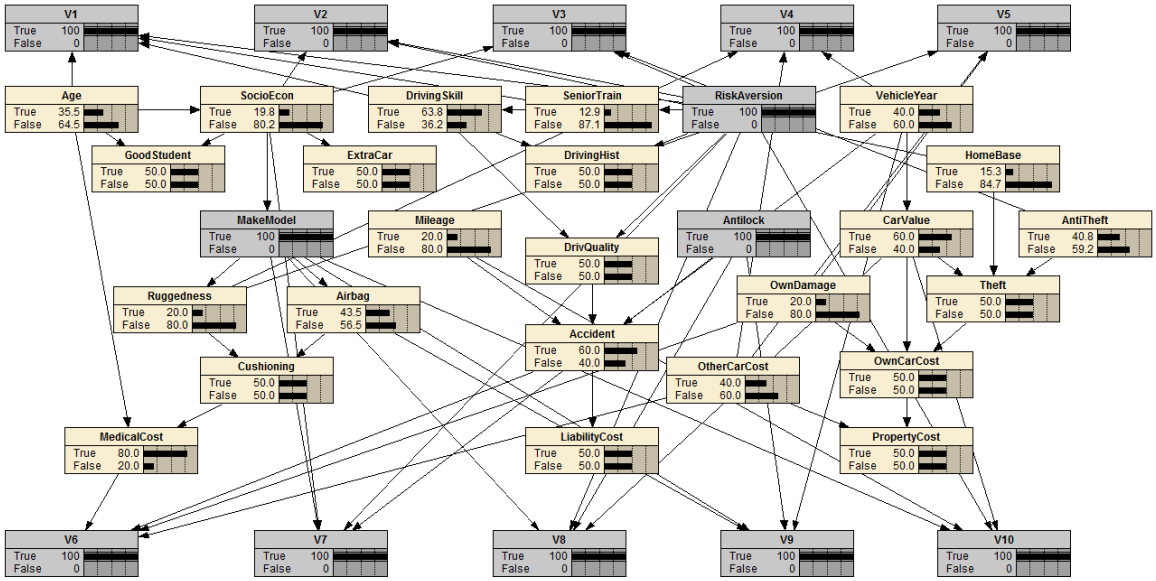
(b) Inference result of the BN merged by AddNode-Basic.



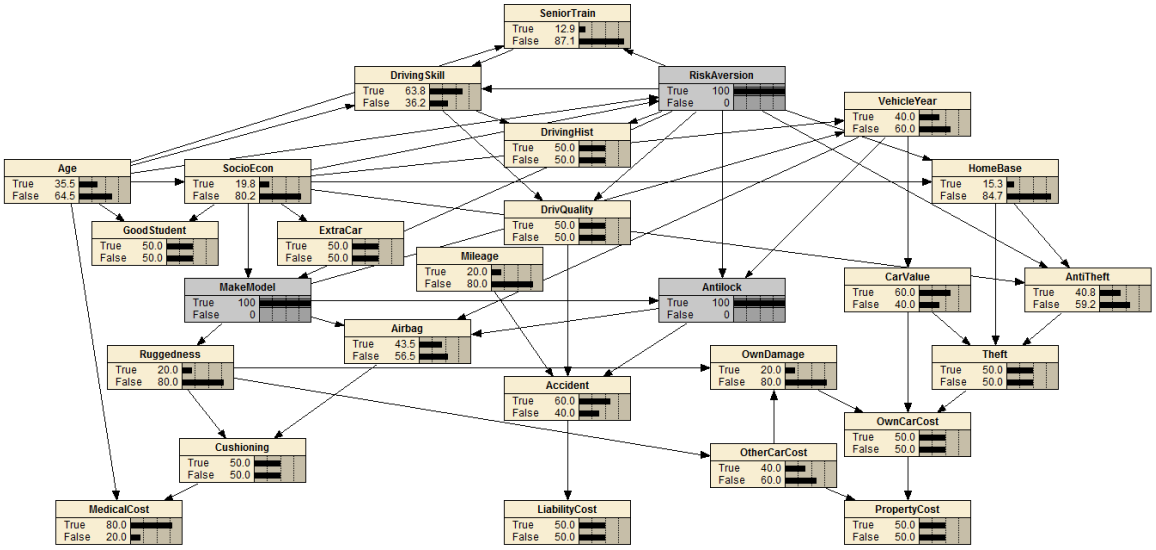
(c) Inference result of the BN merged by AddNode+Merge.



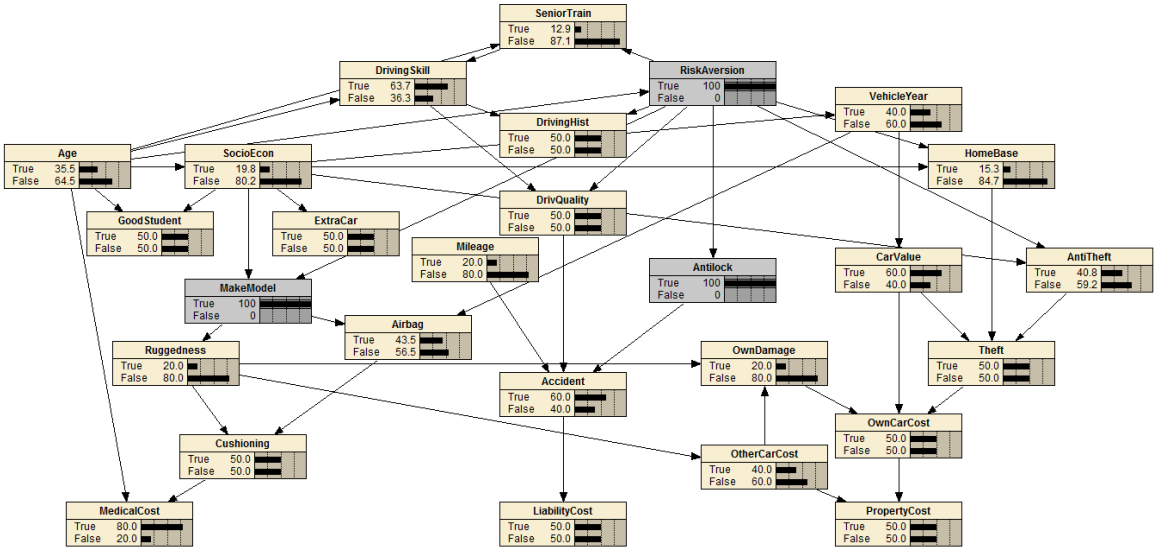
(d) Inference result of the BN merged by AddNode+D-IPFP.



(e) Inference result of the BN merged by AddNode+Factorization.



(f) Inference result of the BN merged by AddLink-Basic.

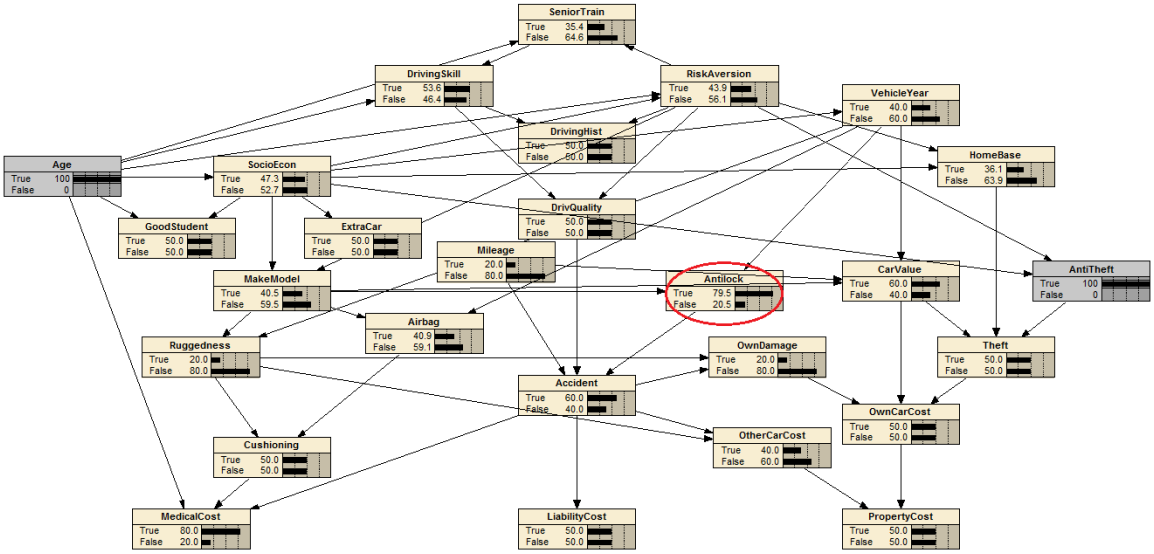


(g) Inference result of the BN merged by AddLink-Prune.

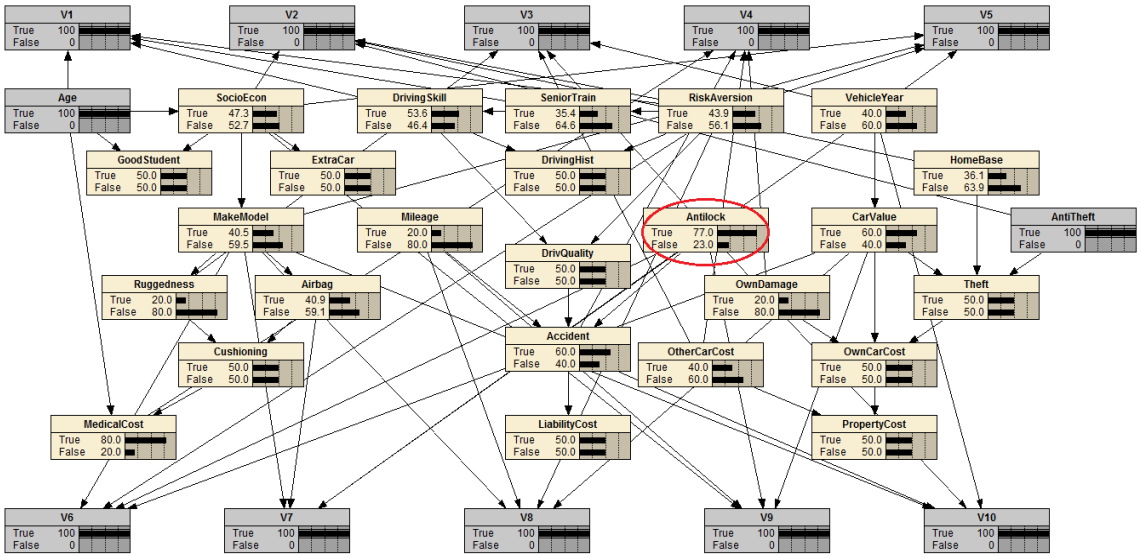
Figure 6.12 Inference results of the baseline BN and the merged BNs for task 1

As can be seen from Figure 6.12, the beliefs for the same node in the baseline BN and the merged BNs are very similar to each other given the three findings. However, since there are still some discrepancies between the baseline BN and the merged BNs, the beliefs for some nodes may be different when given other findings. This can be shown by entering findings of "True" for "Age" and "AntiTheft" for the baseline BN and all the

merged BNs for task 2. After the findings are entered, the baseline BN gives the belief of (0.795, 0.205) for "Antilock" while all the other merged BNs give the belief of (0.770, 0.230) for this node. Figure 6.13 shows the different inference results for the baseline BN and the resulting BN merged by AddNode-Basic.



(a) Inference result of the baseline BN.



(b) Inference result of the BN merged by AddNode-Basic.

Figure 6.13 Inference results of the baseline BN and a merged BN for task 2

The difference is caused by the different strengths of dependency between “Antilock” and “MakeModel” for the baseline BN and the merged BNs. We can enforce the strength of this dependency in Figure 6.13(b) to be the same as that in Figure 6.13(a) by adding an additional integration constraint, which is shown in Figure 6.14. Figure 6.15 shows the resulting BN merged by AddNode-Basic after integrating the ten integration constraints in Figure 6.4 together with the additional integration constraint in Figure 6.14. We can see that the belief for “Antilock” in Figure 6.15 is the same as that in Figure 6.13(a).

RiskAversion	MakeModel	VehicleYear	Antilock	R_{11}
true	true	true	true	0.0251
true	true	true	false	0.0076
true	true	false	true	0.0377
true	true	false	false	0.0114
true	false	true	true	0.2335
true	false	true	false	0.0537
true	false	false	true	0.3503
true	false	false	false	0.0805
false	true	true	true	0.0361
false	true	true	false	0.0109
false	true	false	true	0.0542
false	true	false	false	0.0164
false	false	true	true	0.0268
false	false	true	false	0.0062
false	false	false	true	0.0403
false	false	false	false	0.0093

Figure 6.14 One extra integration constraint for the small BNs

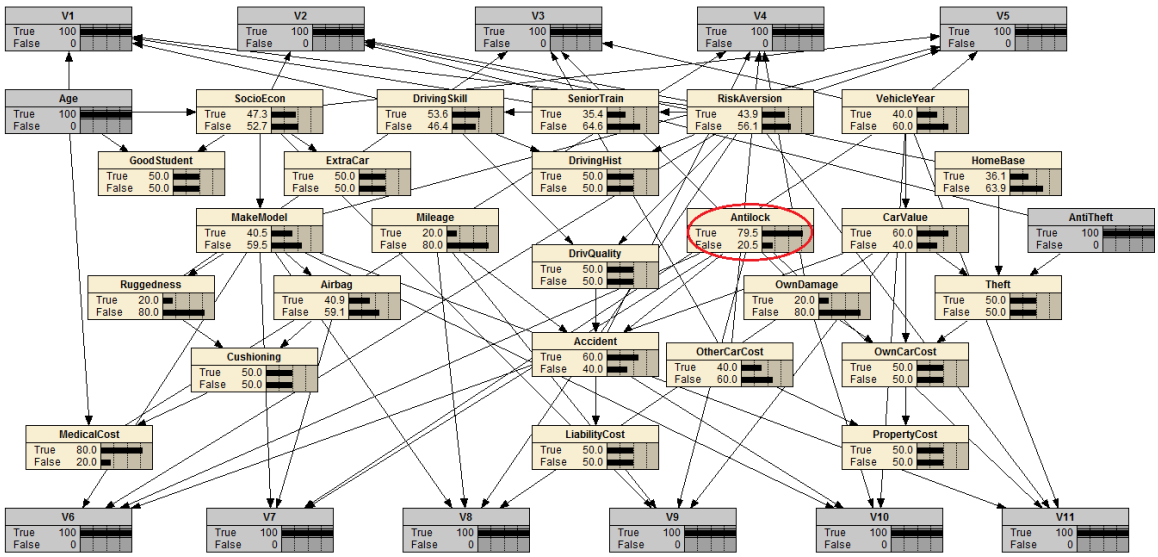


Figure 6.15 Inference result of the merged BN by AddNode-Basic

Inference task 3 is to query the belief of node "Age" after entering findings of "True" for "MakeModel", "RiskAversion" and "Antilock". All merged BNs give the belief of (0.355, 0.645) for "Age". We also compare the time performance of these BNs for task 3. Because of the fluctuation in the results when measuring the time performance, we repeat each inference task 100 times to get its mean, max, min and standard deviation values. The results of task 3 are given in Table 6.2 below.

From the results we can see that the inference performance for each resulting BN is very close to each other. It is also very close to the time performance of the baseline BN. It may be because the junction trees generated from these BNs are very similar to each other.

Table 6.2 Time performance of the baseline BN and the merged BNs for task 3

Method	Mean (ms)	Max (ms)	Min (ms)	SD (ms)
N/A(Baseline BN)	7.34	18.00	6.00	2.11
AddNode-Basic	7.83	26.00	6.00	2.99
AddNode+Merge	7.34	24.00	6.00	2.13
AddNode+D-IPFP	7.28	16.00	6.00	1.54
AddNode+Factorization	7.14	13.00	6.00	1.32
AddLink-Basic	7.60	21.00	6.00	2.95
AddLink-Prune	7.07	26.00	6.00	2.73

6.4 Summary

In this chapter we applied our framework and the related methods to solve the problem of constructing a large BN from a set of small BNs.

In Section 6.1 we illustrated the problem using the modified Insurance network. We split this BN into three small BNs, one for each kind of insurance cost. A set of integration constraints were generated to capture the important dependencies among these small BNs. The goal is to merge the small BNs into a large BN which complies with the integration constraints.

In Section 6.2 we merged the small BNs into a large BN using our proposed methods. We first extracted the dependency information from the constraints using the InconsId method. Then we merged the small BNs into a large BN using both the AddNode and AddLink methods. The results of the Insurance network example showed that, with the help of the integration constraints for the missing dependencies among the small BNs, these BNs can be successfully merged into a large BN.

In Section 6.3 we compared the AddNode and AddLink methods using the integration results in Section 6.2. It was shown that the AddLink methods usually brought fewer changes to the existing BN than the AddNode methods in the number of added nodes, added links, and added CPT entries. For the time performance of each method in this application, besides the AddNode+Merge method which has the highest execution time because the sizes of the CPTs are too large after the constraints are merged, the other AddNode methods have lower execution time than the AddLink methods. This is largely due to the heavier IPFP computation involved in the AddLink methods. The inference results for the baseline BN and the resulting BNs are very similar to each other for task 1.

The inference results showed some difference between the baseline BN and the resulting BN merged by the AddNode-Basic method for task 2 because of some discrepancies in these two BNs, which can be overcome with an additional integration constraint. The time performance of the probabilistic inference task for each resulting BN is very close to each other. It is also very close to the time performance of the baseline BN. It may be because the junction trees generated from these BNs are very similar to each other.

The work reported in this chapter represents our first attempt to extend our framework to other knowledge base engineering problems. There are other applications that may be possible and worthy of exploring by extending our framework. Another area that needs to be investigated further is about the integration constraints, especially how their quality and completeness affect the quality of the merged BN.

7 Conclusion and Future Work

Integrating pieces of new knowledge into an existing knowledge base is essential for developing and maintaining the reliability and accuracy of the knowledge base. In this thesis we focused on the issue of knowledge integration for probabilistic knowledge represented as low dimensional distributions (also called constraint) which has structural inconsistency with the existing knowledge base represented as a BN. Existing works typically solve this issue by removing the inconsistency from the constraint during the integration. However, when the constraint is more up-to-date or comes from a more reliable source, the new dependency relations it brings should be respected. In such situations it is necessary and beneficial to modify the structure of the existing BN so that the constraint, including the new dependency relations it brings, can be integrated into the BN in its entirety. Therefore, this thesis is set to develop and demonstrate techniques to completely integrate the constraint that has structural inconsistency with the BN.

First, we established the theorem that a constraint is structurally consistent with a BN if and only if every dependency implied by the constraint holds in the BN structure. Based on this theorem, we provided a formal definition for structurally inconsistent constraint and developed a method named InconsId to identify structural inconsistencies between a BN and a set of constraints. Both theoretical analysis and experiments showed that the execution time of InconsId increases exponentially with the number of variables in the constraint and increases linearly with either the size of the BN or the number of constraints.

Second, we proposed two classes of methods, i.e., the class of AddNode methods and the class of AddLink methods, to overcome the identified structural inconsistencies. These methods are shown to be able to successfully modify the structure of the existing BN and in turn to successfully integrate the constraints into the BN in their entirety. The class of AddNode methods overcomes the structural inconsistencies by adding nodes to the existing BN. Besides AddNode-Basic, we also developed several variations of it, i.e., AddNode+Merge, AddNode+D-IPFP and AddNode+Factorization, to balance the computational cost and solution quality and to address other concerns. Experiments showed that compared to AddNode-Basic, other variations, while gaining some advantages, have their execution time increase much faster with the number of constraints or the size of the BN. The class of AddLink methods overcomes the structural inconsistencies by adding links to the existing BN. Besides AddLink-Basic, we also developed AddLink-Prune to minimize the number of links to be added to the existing BN. Experiments showed that significant reduction of the number of added links may be achieved by the AddLink-Prune method. Experiment also showed that the execution time of both of the AddLink methods increase linearly with the number of constraints and increase exponentially with the size of the BN.

Lastly, we applied the developed framework and the related methods to the task of constructing a large BN from a set of small BNs which represent its subdomains. Experiments showed that this problem can be formulated as our knowledge integration problem. With the help of a set of integration constraints which reflect the dependencies between variables among the small BNs, a large BN that has the smaller BNs as its components can be constructed using our AddNode or AddLink methods.

With the capability of identifying structural inconsistencies and overcoming them by modifying the structure of the existing BN in a principled way, our work pioneers the research in the area of integrating structurally inconsistent constraints with BNs. By lifting the structural restrictions on the inputs, our work can be applied to a wide range of knowledge integration problems such as KB merging.

Besides the promising results we achieved so far, there are still several areas that can be investigated in the future.

First, some of the methods in our framework can be further improved. For example, for the InconsId method, the execution time increases exponentially with the number of variables in the constraint. This can be improved by finding out how to skip some of the independence tests for the variables based on their connection types in the BN. For the AddNode methods, they can be optimized by adding substructures within or between constraints in order to reduce the size of the CPT for the added nodes. For the AddLink-Prune method, for computational efficiency, currently the scope of search for the added links is limited to those created from the dependency list. If some heuristic information can be identified during the search, the scope of the candidate links can be expanded to all the potential links that do not exist in the given BN. For the AddNode+D-IPFP method, we have proved its effectiveness through the experiment in this thesis. Formal proof of its convergence is still evading us for now and it is worth finding it.

Second, for the methods of overcoming structural inconsistencies, some of them can be combined when needed. In Section 4.2 we have shown how to combine AddNode-Basic with AddNode+Merge when the size of the CPT for the added node is too large after merging all the constraints into one constraint. Similarly, these methods can be

combined in other ways to balance the computational cost and solution quality, as well as to address other concerns. For example, AddNode+Merge can be combined with AddNode+D-IPFP by adding only one node for the structurally inconsistent constraints after merging them into one constraint. AddNode+Merge can also be combined with AddNode+Factorization by merging the factorized structurally inconsistent constraints into one constraint and integrating the factorized structurally consistent constraints using D-IPFP. Additionally, the AddNode methods can also be combined with the AddLink methods when necessary. Some good strategy of how to combine these methods can be developed based on the specific situation of the existing BN and the set of constraints. Also of great interest is to develop guiding principle or heuristics for determining when to use the AddNode methods and when to use the AddLinks methods and when to combine them.

Third, in this thesis we have only considered constraints whose variables are all in the existing BN. In real-world situations, it is very likely that new variables will be introduced by the constraints. In order to integrate constraints with variables not in the existing BN, the methods in our framework need to be extended. If only some of the variables in the constraint are not in the existing BN, we can treat the constraint as a structurally inconsistent constraint, and add nodes or links for it to connect the new variables with the existing BN. If none of the variables in the constraint is in the existing BN, we need to find the connection between this constraint and the existing BN through other constraints that have shared variables with it. Otherwise, a stand-alone BN will be built for this constraint after the integration.

Fourth, we have illustrated how to apply our framework in constructing a large BN from a set of small BNs with an example in Chapter 6. The small BNs in this example do not have any overlaps in their variables. With some enhancement to deal with the possible inconsistent marginal distributions for the shared variables, our framework can be further applied to grow a small BN into a large one when other small BNs that have overlaps with the existing small BN are available. To deal with the possible inconsistent marginal distributions for the shared variables, we can either assign a weight of trust to each small BN, then adopt the marginal distribution in the small BN that has the highest weight; or we can first use SMOOTH to reach a compromise among the inconsistent marginal distributions, then apply it as a constraint during the integration.

Lastly, it is of great value if our framework can be extended to solve data science problems such as knowledge update with large datasets. The large number of variables and cases in the datasets may bring several challenges when our framework is applied. For example, how to effectively pre-process such big data in order to convert the datasets into constraints? How to parallelize our methods so distributed data-parallelism patterns such as MapReduce can be applied to solve the scalability issue during the knowledge update with large datasets?

With the research in the above areas to be investigated, our framework and related methods will have better performance and greater flexibility in solving knowledge integration problems with BNs. It would be of great interest to see more accurate and more reliable knowledge models to be developed for real-world KB integration tasks using our framework.

References

[1] Auer, S.; Bizer, C.; Kobilarov, G.; Lehmann, J.; Cyganiak, R.; Ives, Z. 2007. DBpedia: A Nucleus for a Web of Open Data. In Proceedings of the 6th International Semantic Web Conference and 2nd Asian Semantic Web Conference, pp. 722-735.

[2] Barber, D. 2012. Bayesian Reasoning and Machine Learning. Cambridge University Press.

[3] Beal, M.J.; Falciani, F.; Ghahramani, Z.; Rangel, C.; Wild, DL. 2005. A Bayesian Approach to Reconstructing Genetic Regulatory Networks with Hidden Factors. Bioinformatics, 21(3): 349-356.

[4] Benferhat, S. and Tabia, K. 2014. Reasoning with Uncertain Inputs in Possibilistic Networks. In Proceedings of the Fourteenth International Conference on Principles of Knowledge Representation and Reasoning, pp. 538-547.

[5] Bloemeke, M. 2002. Agent-Encapsulated Bayesian Networks and The Rumor Problem. Technical Reports, TR 2002-006, Depart of Computer Science, University of South Carolina.

[6] Bock, H.H. 1989. A Conditional Iterative Proportional Fitting (CIPF) Algorithm with Applications in the Statistical Analysis of Discrete Spatial Data. Bull. ISI, Contributed papers of 47th Session in Paris, vol. 1, pp. 141-142.

[7] Bollacker, K.; Evans, C.; Paritosh, P.; Sturge, T.; Taylor, J. 2008. Freebase: A Collaboratively Created Graph Database for Structuring Human Knowledge. In

Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, pp. 1247-1250.

[8] Bollacker, K.; Tufts, P.; Pierce, T.; Cook, R. 2007. A Platform for Scalable, Collaborative, Structured Information Integration. In Proceedings of the 6th International Workshop on Information Integration on the Web.

[9] Bradford, J.R.; Needham, C.J.; Bulpitt, A.J.; Westhead, D.R. 2006. Insights into Protein–Protein Interfaces using a Bayesian Network Prediction Method. Journal of Molecular Biology, 362(2): 365-386.

[10] Buntine, W. 1991. Theory Refinement on Bayesian Networks. In Proceedings of the Seventh Conference on Uncertainty Artificial Intelligence, pp. 52-60.

[11] Carlson, A.; Betteridge, J.; Kisiel, B.; Settles, B.; Jr, E. R. H.; Mitchell, T. M. 2010. Toward an Architecture for Never-Ending Language Learning. In Proceedings of the Twenty-Fourth Conference on Artificial Intelligence, pp. 1306-1313.

[12] Chan, H. and Darwiche, A. 2005. On the Revision of Probabilistic Beliefs using Uncertain Evidence. Artificial Intelligence, 163(1): 67-90.

[13] Charniak, E. 1991. Bayesian Networks without Tears. AI Magazine, vol. 12, pp. 50-64.

[14] Chong, C. and Klüppelberg, C. 2018. Contagion in Financial Systems: A Bayesian Network Approach. SIAM Journal on Financial Mathematics, 9(1): 28-53.

[15] Cooper, G. F. and Herskovits, E. 1992. A Bayesian Method for the Induction of Probabilistic Networks from Data. Machine Learning, vol.9, pp. 309-347.

[16] Cramer, E. 2000. Probability Measures with Given Marginals and Conditionals: I-projections and Conditional Iterative Proportional Fitting. Statistics and Decisions, vol. 18, pp. 311-329.

[17] Csiszár, I. 1967. Information-type Measures of Difference of Probability Distributions and Indirect Observation. Studia Sci. Math. Hungar., vol. 2, pp. 299-318.

[18] Csiszar, I. 1975. I-divergence Geometry of Probability Distributions and Minimization Problems. The Annuals of Probability, 3(1): 146-158.

[19] Darwiche, A. 2009. Modeling and Reasoning with Bayesian Networks. Cambridge University Press.

[20] de Campos, C. P. and Ji, Q. 2010. Properties of Bayesian Dirichlet Scores to Learn Bayesian Network Structures. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, pp. 431-436.

[21] Deming, W.E. and Stephan, F.F. 1940. On a Least Square Adjustment of a Sampled Frequency Table When the Expected Marginal Totals are Known. Ann. Math. Statist. 11(4): 427-444.

[22] Deshpande, O.; Lambda, D.; Tourn, M.; Das, S.; Subramaniam, S.; Rajaraman, A.; Harinarayan, V.; Doan, A. 2013. Building, Maintaining and Using Knowledge Bases: A Report from the Trenches. In Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, pp.1209-1220.

[23] Ding, Z. 2005. BayesOWL: A Probabilistic Framework for Uncertainty in Semantic Web. PhD thesis, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County.

[24] Dong, X. L.; Berti-Equille, L.; Srivastatva, D. 2009. Integrating Conflicting Data: the Role of Source Dependence. In Proceedings of the VLDB Endowment, 2(1): 550-561.

[25] Dong, X.; Gabrilovich, E.; Heitz, G.; Horn, W.; Lao, N.; Murphy, K.; Strohmann, T.; Sun, S.; Zhang, W. 2014. Knowledge Vault: A Web-scale Approach to Probabilistic Knowledge Fusion. In Proceedings of the 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, New York, USA.

[26] Fantl, J. and McGrath, M. 2009. Knowledge in an Uncertain World. Oxford University Press.

[27] Fienberg, S.E. 1970. An Iterative Procedure for Estimation in Contingency Tables. Ann. Math. Statist. 41(3): 907-917.

[28] Friedman, N.; Linial, M.; Nachman, I.; Pe'er, D. 2000. Using Bayesian Networks to Analyze Expression Data. Journal of Computational Biology, 7(3-4): 601-620.

[29] Geiger, D.; Verma T.; Pearl, J. 1990. Identifying Independence in Bayesian Networks. Networks, 20(5): 507-534.

[30] Heckerman, D. 2008. A Tutorial on Learning with Bayesian Networks. Springer.

[31] Heckerman, D.; Geiger, D.; Chickering, D. M. 1995. Learning Bayesian Networks: The Combination of Knowledge and Statistical Data. Machine Learning, 20(3): 197-243.

[32] Hermelen, F. van; Lifschitz, V.; Porter, B. 2008. Handbook of Knowledge Representation. Elsevier.

[33] Hesar, A. S. 2013. Structure Learning of Bayesian Belief Networks Using Simulated Annealing Algorithm. Middle-East Journal of Scientific Research, 18(9): 1343-1348.

[34] Hesar, A.S.; Tabatabaee, H.; Jalali, M. 2012. Structure Learning of Bayesian Networks Using Heuristic Methods. In International Proceedings of Computer Science & Information Technology, vol. 45.

[35] Imoto, S; Higuchi, T; Goto H, Tashiro, K; Kuhara, S; Miyano, S. 2003. Combining Microarrays and Biological Knowledge for Estimating Gene Networks via Bayesian Networks. In Proceedings of the Computational Systems Bioinformatics, vol. 2, pp. 104-113.

[36] Jeffrey, R. 1983. The Logic of Decisions, 2nd Edition. University of Chicago Press.

[37] Jensen, V. 2001. Bayesian Networks and Decision Graphs. Springer.

[38] Jordan, M. I. 2004. Graphical Models. Statistical Science (Special Issue on Bayesian Statistics), 19(1): 140-155.

[39] Koller D. and Friedman N. 2009. Probabilistic Graphical Models: Principles and Techniques. MIT Press.

[40] Koski, T.J.T. and Noble, J.M. 2012. A Review of Bayesian Networks and Structure Learning. Mathematica Applicanda, 40(1): 53-103.

[41] Krause, P. and Clark, D. 1993. Representing Uncertain Knowledge: An Artificial Intelligence Approach. Kluwer Academic Publishers.

[42] Kruithof, R. 1937. Telefoonverkeersrekening. De Ingenieur, vol 52, pp. 15-25.

[43] Kullback S. and Leibler R.A. 1951. On Information and Sufficiency. Ann. Math. Statist., vol. 22, pp. 79-86.

[44] Lam, W. and Bacchus, F. 1994. Learning Bayesian Belief Networks: An Approach Based on the MDL Principle. Computational Intelligence, 10(4): 269-293.

[45] Lam, W. and Bacchus, F. 1994. Using New Data to Refine a Bayesian Network. In Proceedings of the Tenth Conference on Uncertainty in Artificial Intelligence, pp. 383-390.

[46] Lauritzen, S.L. and Spiegelhalter, D.J. 1988. Local Computation with Probabilities in Graphic Structures and Their Applications in Expert Systems. In J. Royal Statistical Soc. Series B, 50(2): 157-224.

[47] Lehmann, J.; Isele, R.; Jakob, M.; Jentzsch, A; Kontokostas, D.; Mendes, P. N.; Hellmann S.; Morsey, M.; Kleef, P. van; Auer, S.; Bizer, C. 2014. DBpedia - A Large-scale, Multilingual Knowledge Base Extracted from Wikipedia. Semantic Web Journal.

[48] Maleki, M. and Cruz-Machado, V. 2013. Supply Chain Performance Monitoring using Bayesian Network. International Journal of Business Performance and Supply Chain Modelling, 5(2): 177-197.

[49] Margaritis, D. 2003. Learning Bayesian Network Model Structure from Data. PhD thesis, School of Computer Science, Carnegie-Mellon University, Pittsburgh, PA.

[50] Murray, K. 1995. Learning as Knowledge Integration. PhD thesis, Department of Computer Science, The University of Texas at Austin.

[51] Neapolitan, R. E. 2012. Probabilistic Reasoning in Expert Systems: Theory and Algorithms. CreateSpace Independent Publishing Platform.

[52] Nickel, M.; Murphy, K.; Tresp V.; Gabrilovich, E. 2016. A Review of Relational Machine Learning for Knowledge Graphs. In Proceedings of the IEEE, 104(1): 11-33.

[53] O'Gorman, B.; Babbush, R.; Perdomo-Ortiz, A.; Aspuru-Guzik, A.; Smelyanskiy, V. 2015. Bayesian Network Structure Learning Using Quantum Annealing. The European Physical Journal Special Topics, 224(1): 163-188.

[54] Pan, R. 2006. Semantically-Linked Bayesian Networks: A Framework for Probabilistic Inference over Multiple Bayesian Networks. PhD thesis, Department of Computer Science and Electrical Engineering, University of Maryland, Baltimore County.

[55] Pan, R.; Peng, Y.; Ding, Z. 2006. Belief Update in Bayesian Networks Using Uncertain Evidence. In Proceedings of the IEEE International Conference on Tools with Artificial Intelligence (ICTAI-2006), Washington, DC.

[56] Pearl, J. 1987. The Logic of Representing Dependencies by Directed Graphs. In Proceedings of the Sixth National Conference on Artificial Intelligence, vol. 1, pp. 374-379.

[57] Pearl, J. 1990. Jeffery's Rule, Passage of Experience, and Neo-Bayesianism. Knowledge Representation and Defeasible Reasoning, pp. 245-265, Kluwer Academic Publishers.

[58] Pearl, J. 1995. Causal Diagrams for Empirical Research. Biometrika, 82(4):669-710.

[59] Pearl, J. 1997. Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, 2nd Edition. Morgan Kaufman, San Mateo, CA.

[60] Pearl, J. 2009. Causality: Models, Reasoning and Inference, 2nd Edition. Cambridge University Press.

[61] Pearl, J. and Russell, S. 2003. Bayesian Networks. The Handbook of Brain Theory and Neural Networks, pp. 157-160, MIT Press.

[62] Pe'er, D. 2005. Bayesian Network Analysis of Signaling Networks: A Primer. Science's STKE, 2005(281).

[63] Peng, Y. and Ding, Z. 2005. Modifying Bayesian Networks by Probability Constraints. In Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence, Edinburgh.

[64] Peng, Y. and Zhang, S. 2010. Integrating Probability Constraints into Bayesian Nets. In Proceedings of the 9th European Conference on Artificial Intelligence, Lisbon, Portugal.

[65] Peng, Y.; Ding, Z.; Zhang, S.; Pan, R. 2012. Bayesian Network Revision with Probabilistic Constraints. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, pp. 317-337.

[66] Peng, Y.; Zhang, S.; Pan, R. 2010. Bayesian Network Reasoning with Uncertain Evidences. International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, 18(5): 539-564.

[67] Poole, D. and Mackworth, A. 2017. Artificial Intelligence: Foundations of Computational Agents, 2nd Edition. Cambridge University Press.

[68] Rüschendorf, L. 1995. Convergence of the Iterative Proportional Fitting Procedure. Ann. Statist., 23(4): 1160-1174.

[69] Russell, S. and Norvig, P. 2009. Artificial Intelligence: A Modern Approach. Prentice-Hall.

[70] Spirtes, P.; Glymour G.; Scheines, R. 1993. Causation, Prediction, and Search. Springer-Verlag New York.

[71] Sun, Y. and Peng, Y. 2016. Inconsistent Knowledge Integration with Bayesian Networks. The 29th International FLAIRS Conference, Key Largo, Florida, USA.

[72] Sun, Y. and Peng, Y. 2016. Modify Bayesian Network Structure with Inconsistent Constraints. In Proceedings of the ISCA 29th International Conference on Computer Applications in Industry and Engineering, Denver, CO, USA.

[73] Suzuki, J. 1996. Learning Bayesian Belief Networks Based on the Minimum Description Length Principle: An Efficient Algorithm Using the B&B Technique. Machine Learning, Proceedings of the Thirteenth International Conference, pp. 462-470.

[74] Tessier, C.; Chaudron, L.; Müller, H.-J. 2001. Conflicting Agents: Conflict Management in Multi-Agent Systems. Kluwer Academic Publishers.

[75] Thimm, M. 2009. Measuring Inconsistency in Probabilistic Knowledge Bases. In Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, pp. 530-537.

[76] Valtorta, M.; Kim, Y.; Vomlel, J. 2002. Soft Evidential Update for Probabilistic Multiagent Systems. International Journal of Approximate Reasoning, 29(1): 71-106.

[77] Verma, T. and Pearl, J. 1991. Equivalence and Synthesis of Causal Models. In Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence, pp. 255-270.

[78] Vomlel, J. 1999. Methods of Probabilistic Knowledge Integration. PhD thesis, Department of Cybernetics, Faculty of Electrical Engineering, Czech Technical University.

[79] Vomlel, J. 2004. Integrating Inconsistent Data in a Probabilistic Model. Journal of Applied Non-Classical Logics, 14(3): 1-20.

[80] Wagner, C. 2002. Probability Kinematics and Commutativity. Philosophy of Science, 69(2): 266-278.

[81] Xiang, Y. 2002. Probabilistic Reasoning in Multi-agent Systems: A Graphical Models Approach. Cambridge University Press.

[82] Xiang, Y. and Lesser, V. 2000. Justifying multiply sectioned Bayesian networks. In Proceedings of the 4th International Conference on Multi-Agent Systems, Boston, MA.

[83] Xiang, Y. and Lesser, V. 2003. On the Role of Multiply Sectioned Bayesian Networks to Cooperative Multiagent Systems. IEEE Transactions on Systems, Man, and Cybernetics, Part A, 33(4):489-501.

[84] Xiang, Y.; Poole, D.; Beddoes, M. P. 1993. Multiply Sectioned Bayesian Networks and Junction Forests for Large Knowledge Based Systems. Computational Intelligence, 9(2): 171-220.

[85] Yuan, C.; Malone, B.; Wu, X. 2011. Learning Optimal Bayesian Networks Using A* Search. In Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence (IJCAI-11), pp. 2186-2191, Helsinki, Finland.

[86] Zhang, S. and Peng, Y. 2008. An Efficient Method for Probabilistic Knowledge Integration. In Proceedings of the 20th IEEE International Conference on Tools with Artificial Intelligence, Dayton, Ohio.

[87] http://wiki.dbpedia.org

[88] http://www.bnlearn.com/bnrepository/

[89] http://www.norsys.com/

[90] https://en.wikipedia.org/wiki/Knowledge_Graph

[91] https://www.cs.mcgill.ca/~dprecup/courses/Prob/Lectures/prob-lecture03.pdf

[92] https://www.wikipedia.org