Approximation of General CPTs with NIN-AND Tree Models

by

Qian Jiang

A Thesis
Presented to
The University of Guelph

In partial fulfillment of requirements
for the degree of
Master of Science
in
Computer Science

Guelph, Ontario, Canada
© Qian Jiang, November, 2016
Approximation of General CPTs with NIN-AND Tree Models

Qian Jiang
University of Guelph, 2016

A Bayesian network (BN) is a probabilistic graphical model representing causal dependence and conditional independence among domain variables. The strength of dependence is quantified by conditional probability tables (CPTs). The space complexity of BNs is exponential on the number of parents per variable, which limits the effectiveness of knowledge acquisition and inference. Non-impeding noisy-AND tree (NAT) models give a highly expressive approximate representation of BNs. Approximating CPTs in BNs by NAT models can significantly reduce the space complexity and speed up BN inference. In this thesis, we generalize the CPT approximation by NAT models. Leaky causes, representing all causes that are not explicitly modeled, exist in CPTs from real world BNs. We approximate CPTs with persistent leaky causes (PLCs) by explicitly modeling such leaky causes. Experimental results show that approximating general CPTs by NAT models is reasonably accurate. Explicitly modeling PLCs further improves the accuracy in CPTs with PLCs.
Acknowledgments

I would like to sincerely express my gratitude to my advisor Dr. Yang Xiang for his invaluable guidance and encouragement. Besides, I would like to thank my advisory committee member Dr. Fangju Wang for the helpful suggestion towards my research. Furthermore, I would like to thank everyone in the School of Computer Science. It has been a wonderful experience studying and working with such amazing people. Last but not least, I would like to thank my family and friends, especially my parents for their support and love.
## Table of Contents

List of Tables vii

List of Figures viii

List of Abbreviations ix

### 1 Introduction
1.1 Overview .................................................. 1
1.2 Thesis Structure ............................................. 4

### 2 Background
2.1 Basic Concepts of Graph Theory ......................... 5
  2.1.1 Directed Graphs ........................................ 6
  2.1.2 Directed Acyclic Graphs ............................... 7
2.2 Basic Concept of Probability ............................... 8
  2.2.1 Frequency Probability and Bayesian Probability .... 8
  2.2.2 Properties of Probability ............................ 9
  2.2.3 Conditional Independence ............................. 10
2.3 Bayesian Networks .......................................... 11
  2.3.1 Introduction to Bayesian Networks ................. 11
  2.3.2 Inference with Bayesian Networks ................. 14
2.4 Space-efficient Models .................................... 17
  2.4.1 Terminologies .......................................... 18
  2.4.2 Leaky Causes ........................................... 20
  2.4.3 Noisy-OR Models ...................................... 22
  2.4.4 Binary NIN-AND Tree Models ......................... 25
  2.4.5 Noisy-MAX Models ..................................... 33
  2.4.6 Multi-valued NIN-AND Tree Models ................... 36
  2.4.7 Other Existing Models ................................ 44
2.5 Approximating Binary CPTs with NAT Models ............... 47
  2.5.1 Partial PCI Pattern Identification .................... 47
  2.5.2 PCI Pattern Based Search Trees and Candidate NAT Search 48
  2.5.3 Parameter Search by Gradient Descent ............... 50
  2.5.4 Experimental Result of Binary CPT Approximation .... 51
2.6 Bayesian Network Inference with NAT Models ............... 52
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>JPD of tampering, fire and alarm</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>JPD of fire, alarm and leaving</td>
<td>15</td>
</tr>
<tr>
<td>2.3</td>
<td>Updated JPD of fire, alarm and leaving</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>CPT with a persistent leaky cause</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>CPT with a non-persistent leaky cause</td>
<td>22</td>
</tr>
<tr>
<td>2.6</td>
<td>CPT without leaky causes</td>
<td>22</td>
</tr>
<tr>
<td>2.7</td>
<td>CPT of fever, cold and flu</td>
<td>25</td>
</tr>
<tr>
<td>2.8</td>
<td>CPT with only single-causals specified</td>
<td>32</td>
</tr>
<tr>
<td>2.9</td>
<td>Full CPT derived from the NAT model</td>
<td>33</td>
</tr>
<tr>
<td>2.10</td>
<td>CPT of multi-valued flu fever and cold</td>
<td>37</td>
</tr>
<tr>
<td>2.11</td>
<td>CPT of the multi-valued NAT in Figure 2.11</td>
<td>43</td>
</tr>
<tr>
<td>2.12</td>
<td>The number of NATs with ( n ) causes</td>
<td>49</td>
</tr>
<tr>
<td>3.1</td>
<td>Value-pair interactions of NATs in Figure 3.2</td>
<td>73</td>
</tr>
<tr>
<td>3.2</td>
<td>Pairwise causal probability comparison by NAT models</td>
<td>74</td>
</tr>
<tr>
<td>4.1</td>
<td>Experiments of approximating general CPTs without PLCs</td>
<td>87</td>
</tr>
<tr>
<td>4.2</td>
<td>Experiments of approximating general CPTs with PLCs</td>
<td>88</td>
</tr>
<tr>
<td>4.3</td>
<td>Summary of the selected real world BNs and CPTs</td>
<td>90</td>
</tr>
<tr>
<td>4.4</td>
<td>Groups of target CPTs from real world with PLCs</td>
<td>90</td>
</tr>
<tr>
<td>4.5</td>
<td>Performance summary of NPLC-App and NPLC-Exh</td>
<td>93</td>
</tr>
<tr>
<td>4.6</td>
<td>Performance summary of NPLC-App and NMAX</td>
<td>95</td>
</tr>
<tr>
<td>4.7</td>
<td>Performance summary of PLC-App and PLC-Exh</td>
<td>96</td>
</tr>
<tr>
<td>4.8</td>
<td>Performance summary of PLC-App, NPLC-App and NMAX</td>
<td>97</td>
</tr>
<tr>
<td>4.9</td>
<td>Performance of PLC-App, NPLC-App, and NMAX over real world CPTs</td>
<td>98</td>
</tr>
</tbody>
</table>
## List of Figures

2.1 (a) An undirected graph. (b) A directed graph. (c) A hybrid graph. (d) A directed acyclic graph .......................... 6  
2.2 A BN example: fire diagnosis ........................................ 12  
2.3 A simple BN of three variables .................................... 14  
2.4 Causes of *alarm* with a leaky cause ............................ 21  
2.5 A causal interaction .................................................. 24  
2.6 A binary dual NIN-AND gate ........................................ 27  
2.7 A binary direct NIN-AND gate ...................................... 28  
2.8 A binary NAT .......................................................... 30  
2.9 A multi-valued NIN-AND dual gate ................................. 38  
2.10 A multi-valued NIN-AND direct gate .............................. 39  
2.11 A multi-valued NAT .................................................. 41  
2.12 A simple BN of five causes ......................................... 44  
2.13 The BN representation utilizing CSI ............................. 45  
2.14 An intercausal cancellation model ................................. 46  
2.15 A PST for *n* = 3 connected to NATs with direct leaf gates .................................................. 50  
2.16 A 3-dimensional gradient descent ................................... 51  
2.17 Hybrid graph for MF of a gate ...................................... 52  
2.18 (a) A 4-gate NAT topology. (b) MF graph of (a) ............... 54  

3.1 Value-pair interaction identification: (a) *Tight closure* rule. (b) *Sided loose closure* rule ................................. 64  
3.2 (Sub)NATs over *c*<sub>0</sub>, *c*<sub>i</sub> and *c*<sub>j</sub> .......................................................... 72  
3.3 Framework of extracting value-pair interactions .................. 82  

4.1 The number of well-defined PCI bits relative to *e*<sup>1</sup> ........ 91  
4.2 The number of well-defined PCI bits relative to *e*<sup>2</sup> ........ 91  
4.3 The number of well-defined PCI bits relative to *e*<sup>3</sup> ........ 92  
4.4 Euclidean distance (left) and runtime (right) for NPLC-App and NPLC-Exh over Batch 2.1 ................................. 92  
4.5 Euclidean distance (left) and runtime (right) for NPLC-App and NMAX over Batch 2.1 ...................................... 94  
4.6 Euclidean distance (left) and runtime (right) for NPLC-App and NMAX over Batch 3.1 ...................................... 94  
4.7 Euclidean distance (left) and runtime (right) for NPLC-App and NMAX over Batch 3.2 ...................................... 94
4.8 Euclidean distance (left) and runtime (right) for PLC-App and PLC-Exh over Batch 4.1 ......................................................... 96
4.9 Euclidean distance (left) and runtime (right) for PLC-App, NPLC-App and NMAX over Batch 5.1 ........................................... 97
### List of Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BN</td>
<td>Bayesian Network</td>
</tr>
<tr>
<td>CPD</td>
<td>Conditional Probability Distribution</td>
</tr>
<tr>
<td>CPT</td>
<td>Conditional Probability Table</td>
</tr>
<tr>
<td>DAG</td>
<td>Directed Acyclic Graph</td>
</tr>
<tr>
<td>JPD</td>
<td>Joint Probability Distribution</td>
</tr>
<tr>
<td>NAT</td>
<td>Non-Impeding Noisy-AND Tree</td>
</tr>
<tr>
<td>NIN-AND Tree</td>
<td>Non-Impeding Noisy-AND Tree</td>
</tr>
<tr>
<td>PCI</td>
<td>Pairwise Causal Interaction</td>
</tr>
<tr>
<td>PLC</td>
<td>Persistent Leaky Cause</td>
</tr>
<tr>
<td>PST</td>
<td>PCI Pattern Based Search Tree</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Overview

In Artificial Intelligence (AI), an agent should be capable to make decisions rationally upon its action over the environment. Thus, an agent must handle the basic knowledge and understanding of the environment. However, the environment is usually uncertain for an agent because it is random or partially observed. The perception of the environment for an agent might be uncertain. The activity of guessing the state of environment from knowledge and observation is referred to as inference or reasoning. Traditional AI systems are rule-based systems. Since a rule-based intelligent agent cannot concisely represent uncertain environment by certain rules [10], probability theory is applied to quantify the uncertainty. Joint probability distribution (JPD) is applied to help agents to handle the uncertain knowledge, but it is intractable facing to complex problems. The reason is that JPDs are exponential on the number of total environment variables.

Bayesian networks (BNs) were introduced by Pearl [8], which combine probability theory and graph theory to model uncertain knowledge and enable efficient inference. A BN is a probabilistic graphic model consisting of three components.
They are a set of variables, a directed acyclic graph (DAG) as the graphic structure and a set of conditional probability tables (CPTs). Each node in the DAG represents a variable from the variable set. A directed link connects a pair of node. In each link, the head node is called child or effect and the tail node is its parent or cause. A node with its parents is called a BN family. A BN models causal dependence and conditional independence among domain variables. The strength of dependence is quantified by a CPT for each variable. Each CPT is associated with a node. The complexity of a CPT is exponential on the number of parents of the corresponding node. Thus, the complexity of a BN is linear on the number of variables, but it grows exponentially on the number of parents per node. Therefore, comparing to JPDs, BNs show a more compact representation of knowledge.

BNs significantly reduce the complexity of knowledge representation, but the size per CPT is still exponential on the number of parents per node. Several techniques have been proposed to deal with this issue. Causal independence models are one type of such techniques, among which noisy-OR [9] is the most well known. Noisy-OR models reduce the complexity from exponential on the number of causes to linear over binary variables. Noisy-OR is extended to noisy-MAX which is applied over multi-valued variables. However, these two models have the same limitation. They assume that causal interactions among variables are only reinforcing. In real world, causes of an effect might undermine each other, which means causal interactions can be undermining. To overcome this limitation, Non-Impeding Noisy-AND (NIN-AND) tree models [18] have been presented over binary variables and generalized to multi-variables [17]. NIN-AND tree (NAT) models encode both undermining and
reinforcing causal interactions as well as their mixture. A NAT model consists of a
NAT topology (simplified as NAT) and a set of single-causal probabilities (simplified
as single-causals). The complexity of NAT models is linear on the number of causes.
In addition, the recent work enables NAT models over multi-valued variables to speed
up BN inference [21].

The technique to approximate binary CPTs by NAT models has been developed
[23]. Given a binary target CPT, the binary CPT approximation starts with defining
the causal interaction between each pair of causes. Then, a set of NATs is found as
candidates based on the set of causal interactions. For each candidate NAT, gradient
descent is used to search for single-causals, where a set of NAT models is obtained.
At last, a NAT model with the smallest error from the target CPT is selected to be
the approximation. The study shows that approximating binary CPTs with NAT
models is more accurate than by noisy-OR models.

In this research, we first extend the CPT approximation by NAT models to
multi-valued variables. A *Leaky cause* represents all causes that are not explicitly
modeled. Leaky causes exist in CPTs from real world BNs. The first method deals
with CPTs without leaky causes. However, in real world, leaky causes might be
always active in target CPTs. When they are always active, we call them *persistent leaky causes*. To deal with this issue, we develop a second technique to approximate
CPTs with persistent leaky cause by NAT models. The experimental results show
that the accuracy of CPT approximation by NAT model is improved comparing to
noisy-MAX. Moreover, for a target CPT with a persistent leaky cause, explicitly
modeling such a leaky cause is more accurate than not modeling it.
1.2 Thesis Structure

This thesis is structured as follows. In Chapter 2, necessary background knowledge is reviewed. Chapter 3 studies the methods to approximate general CPTs with NAT models. In Chapter 4, experimental design and results are demonstrated. At last, Chapter 5 summarizes the research, evaluates the limitations and discusses the future work.
Chapter 2

Background

This chapter reviews necessary background knowledge related to this thesis followed by related previous work. Basic concepts of graph theory and probability theory are reviewed in Section 2.1 and 2.2. In Section 2.3, basic concepts of BNs and inference in BNs are introduced. Section 2.4 discusses leaky causes and related models including NAT models. Section 2.5 presents the approximation of binary CPTs with NIN-AND Tree Models. Finally, BN inference by NAT models is studied in Section 2.6.

2.1 Basic Concepts of Graph Theory

In Computer Science, a graph is a visualization tool for intuitively modeling complex problems. A graph consists of two elements including a set $V$ of nodes or vertices and a set $E$ of edges connecting nodes in pairs. If two nodes are linked by an edge, they are said to be adjacent. Hence, a graph can be described by the notation $G = (V, E)$. In this thesis, nodes in a graph are represented by ovals with labels within it.

Edges in a graph can be either directed or undirected. If all edges in a graph
are directed, the graph is called a directed graph (Figure 2.1 (b), (d)). Otherwise, the graph is an undirected graph (Figure 2.1 (a)) or a hybrid graph (Figure 2.1 (c)). The graphs in BNs are directed graphs, so we focus on directed graphs here.

![Figure 2.1](image.png)

**Figure 2.1**: (a) An undirected graph. (b) A directed graph. (c) A hybrid graph. (d) A directed acyclic graph

A sequence of nodes is called a *path* if every node in the sequence is adjacent to its neighbor node or nodes. We denote a path by surrounding the sequence of nodes with ⟨⟩. In Figure 2.1 (a), for instance, ⟨a, b, c, d, e⟩ is a path. A path is a *cycle* if there are two or more distinct nodes in the path and the first node and the last node are identical. We denote a cycle by surrounding the sequence of nodes with ⟨⟩. In Figure 2.1 (a), ⟨b, c, d, e, b⟩ is an example of a cycle.

### 2.1.1 Directed Graphs

As shown above, a directed graph consists of a set of nodes and a set of directed edges. Each edge connects an ordered pair of nodes. Figure 2.1 (b) is an example of a directed graph. There is a directed edge from a to b. Node a is *tail* and node b is...
We also refer to $a$ as a parent of $b$ and $b$ as a child of $a$.

A path in a directed graph is a directed path if each node in the sequence, other than the first and the last, is the parent of one neighbor node and the child of the other neighbor node. We denote a directed path by surrounding the sequence of nodes with $()$. In Figure 2.1 (b), $(a, b, e, d, c)$ is a directed path. Node $a$ is called an ancestor of $c$, and $c$ is a descendant of $a$. A cycle in a directed graph is directed if each node in the sequence is the parent of one neighbor and the child of the other neighbor node. We denote a direct cycle by surrounding the sequence of nodes with $()$. In Figure 2.1 (b), $(b, e, d, c, b)$ is a directed cycle.

### 2.1.2 Directed Acyclic Graphs

A directed acyclic graph (DAG) is a directed graph with no directed cycles. As we can see, in Figure 2.1 (b), $(b, e, d, c, b)$ is a directed cycle. Hence, Figure 2.1 (b) is not a DAG. However, comparing Figure 2.1 (d) with Figure 2.1 (b), we found the directions between node $b$ and $c$ are opposite in these two graphs. The cycle $(b, e, d, c, b)$ found in Figure 2.1 (d) is undirected. No more directed cycles can be found in the graph. Therefore, Figure 2.1 (b) is a DAG. DAGs are applied as a graphical representation of dependencies and conditional independencies in Bayesian networks.
2.2 Basic Concept of Probability

2.2.1 Frequency Probability and Bayesian Probability

Because of partial observation and the uncertain environment, an agent may not obtain all aspects of the environment. Propositional logic is limited to briefly deal with the uncertainty [10]. Probability theory is one way to handle this problem. We introduces two common ways of probability interpretations.

*Frequentist probability*, a standard probability interpretation based on frequency, is objective and can only be obtained by observing enough independent cases. Frequency probability can be approximated by the ratio \( P(A) \approx \frac{N_A}{N} \), where \( N_A \) represents the number of trials in which event \( A \) occurs and \( N \) represents the total number of trials. When the number of trials approaches infinity, the true probability will be converged such that \( P(A) = \lim_{N \to \infty} \frac{N_A}{N} \). However, frequentist probability has its limitation. When some trials cannot be repeated, the probability is undefined. For example, the probability of winning the 2016 NBA final cannot be observed by repeating experiments.

*Bayesian probability* is subjective, which is different from frequentist probability. Bayesian probability theory tells us that the belief of an agent depends on its knowledge about the event. Thus, it is not necessary to observe enough experimental results. For instance, to decide the probability of obtaining a head when flip a fair coin, denoted by \( P(H) \), frequentist repeats trials enough times to approximate the probability. From Bayesian interpretation, the probability can be obtained based on the knowledge of a fair coin. The chance of getting a head is the same of getting a
tail, which means $P(H) = 0.5$.

In this thesis, we adopt Bayesian probability interpretation.

### 2.2.2 Properties of Probability

In this subsection, we introduce properties of Bayesian. We give an example to explain the terminologies. Define three variables *fire*, *alarm* and *smoke*. Define the value of each variable as *true (t)* and *false (f)*. Define the domain of variables $D = \{fire, alarm, smoke\}$. Define $F = \{fire\}$ as a subset of $D$.

A probability such as $P(fire = t)$ is an *unconditional probability*. A probability distribution such as $P(F)$ is an *unconditional distribution*. The unconditional probability $P(D)$ over the entire domain is referred to as *joint probability distribution* (JPD). Each joint probability represents one combination of values of all variables involved. Since $F$ is a subset of $D$, distribution $P(F)$ is a *marginal* distribution of $P(D)$. Moreover, probabilities can also be informed with observation. For example, the evidence that *smoke* in the building is *true* increase the chance that the building is on *fire*. We can use a *conditional probability* such as $P(fire = t | smoke = t)$ to represent the probability that the building is on *fire* given the condition that there is *smoke* in the building. The distribution $P(F | smoke = t)$ represents the belief of *fire* given the observation that *smoke* is true. It is called *conditional probability distribution* (CPDs).

The probability of an event is between 0 and 1, such as $0 \leq P(fire = t) \leq 1$. The *sum* rule explains that the sum of a probability distribution equals to 1, such as $P(F) = P(fire = t) + P(fire = f) = 1$. Based on the *negation* rule, such as
\[ P(\text{fire} \neq t | \text{smoke} = t) = 1 - P(\text{fire} = t | \text{smoke} = t), \]
we can derive the belief of a variable with one specific value. Given two events, according to the \textit{marginalization} rule, such as \[ P(\text{fire} = t) = \sum_{\text{smoke} = t} P(\text{fire} = t, \text{smoke} = t), \]
the distribution of one event can be derived. One important rule in Bayesian probability calculation is the \textit{product} rule shown as follows,

\[ P(\text{fire} = f, \text{alarm} = t) = P(\text{fire} = f | \text{alarm} = t)P(\text{alarm} = t). \]

From product rule, a useful property called \textit{Bayes’s} rule is derived. Bayes’s rule is used to calculate the belief of hypothesis \( H \) given the evidence of observation \( O \), denoted by \( P(H|O) \), which cannot be observed. Given the belief \( P(H) \), \( P(O) \) and \( P(O|H) \), we can derive \( P(H|O) = P(O|H) \frac{P(H)}{P(O)}. \)

\section{2.2.3 Conditional Independence}

Although JPDs give the impression that every variable is dependent on every others, that is usually not the case. For example, when people in a building observe a \textit{fire} in the building, they will be \textit{leaving} the building. The belief on \textit{leaving} is dependent on \textit{fire}. However, if we slightly change the observation, this may be different. If a \textit{fire} triggers an \textit{alarm} in a building, and people will be \textit{leaving} the building when they hear the \textit{alarm}. Therefore, when people hear the \textit{alarm}, they will be \textit{leaving} the building, no matter the building is on \textit{fire} or not. In this case, \textit{leaving} is \textit{conditionally independent} on \textit{fire} given \textit{alarm}, denoted by \( I(\text{fire, alarm, leaving}) \), and the equation

\[ P(\text{leaving}|\text{alarm}, \text{fire}) = P(\text{leaving}|\text{alarm}) \]
holds. It indicates when alarm is observed, the information given by fire no longer affects leaving. With the definition of conditional independence and the product rule, the chain rule is derived. For example,

\[
P(\text{leaving, alarm, fire}) = P(\text{leaving} | \text{alarm, fire}) \times P(\text{alarm} | \text{fire}) \times P(\text{fire})
\]

\[
= P(\text{leaving} | \text{alarm}) \times P(\text{alarm} | \text{fire}) \times P(\text{fire}).
\]

With the chain rule, a JPD can be expressed as the product of conditional probability tables.

2.3 Bayesian Networks

In this section, we will introduce what Bayesian networks are, and inference of Bayesian networks.

2.3.1 Introduction to Bayesian Networks

A Bayesian network (BN) [8] is a probabilistic graphic model that represents causal dependence and conditional independence among domain variables. A BN consists of three main elements including a set of variables \( V \), a connected DAG \( G \) and a set of probability distributions \( P \) which are represented by conditional probability tables (CPTs). Therefore, a BN can be denoted by a triplet \((V, G, P)\). In the DAG of a BN, each node stands for a variable from variable set \( V \). With the structure of DAGs, the conflict is avoided that an event is caused by itself. Edges in DAGs represent causal dependencies among variables. Each variable is conditionally independent on its ancestors given its parents. Each node in the DAG is associated with a CPT from
$P$, which shows CPDs of the variable conditioned by its parents. In other words, a CPT consists of multiple CPDs, one for each value combination of the parent variables. Here is an example of BN named fire diagnosis shown in Figure 2.2.

Here is an example of BN named fire diagnosis shown in Figure 2.2.

Six binary variables tampering, fire, alarm, smoke, leaving and report are defined in the problem. Each of them has two possible values true ($t$) and false ($f$). In the BN, tampering and fire can trigger the alarm in the building. When the building is on fire, there may be smoke. When people in the building hear Alarm, they will be leaving the building. After that, the manager will write a report because of leaving. It is defined that leaving is conditionally independent on tampering or fire given alarm. It means that, given the information of alarm, the belief of leaving no longer needs the observation of tampering or fire. Moreover, report is conditionally independent on alarm given leaving, which means if leaving is observed, the information of alarm has no longer affects the belief of report.
We can also observe that in Figure 2.2, the CPT of alarm only contains the conditional probabilities when \( \text{alarm} = t \). The reason is that according to the sum rule, we can derive the probability

\[
P(\text{alarm} = f | \text{tampering}, \text{fire}) = 1 - P(\text{alarm} = t | \text{tampering}, \text{fire}).
\]

With these CPTs, we can obtain the JPD by the chain rule. For instance,

\[
P(\text{alarm} = t, \text{tampering} = t, \text{fire} = t)
= P(\text{alarm} = t | \text{tampering} = t, \text{fire} = t)P(\text{tampering} = t)P(\text{fire} = t)
= 0.5 \times 0.02 \times 0.01 = 0.0001.
\]

An example of JPD with variables alarm, tampering and fire is shown in Table 2.1.

<table>
<thead>
<tr>
<th>alarm</th>
<th>tampering</th>
<th>fire</th>
<th>( P(\text{tampering, fire, alarm}) )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( f )</td>
<td>( f )</td>
<td>( f )</td>
<td>0.97010298</td>
</tr>
<tr>
<td>( f )</td>
<td>( f )</td>
<td>( t )</td>
<td>0.000098</td>
</tr>
<tr>
<td>( f )</td>
<td>( t )</td>
<td>( f )</td>
<td>0.00297</td>
</tr>
<tr>
<td>( f )</td>
<td>( t )</td>
<td>( t )</td>
<td>0.0001</td>
</tr>
<tr>
<td>( t )</td>
<td>( f )</td>
<td>( f )</td>
<td>0.00009702</td>
</tr>
<tr>
<td>( t )</td>
<td>( f )</td>
<td>( t )</td>
<td>0.009702</td>
</tr>
<tr>
<td>( t )</td>
<td>( t )</td>
<td>( f )</td>
<td>0.01683</td>
</tr>
<tr>
<td>( t )</td>
<td>( t )</td>
<td>( t )</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

The number of parameters in a JPD with all variables in a problem is \( O(\kappa^\mu) \), where \( \mu \) denotes the number of variables, each of which has up to \( \kappa \) possible values. In our example of problem, the full JPD has \( 2^6 = 64 \). However, with the BN representation, the JPD is factorized into \( \mu \) CPTs, and each of them has up to \( \kappa^n \) parameters, where \( n \) denotes the the largest number of parents. This means that the
number of parameters in each CPT is $O(\kappa^n)$. The number of defined parameters is reduced from $O(\kappa^\mu)$ to $O(\mu \kappa^n)$. For instance, in our example, the total number of defined probability in all CPTs is $1 + 1 + 4 + 2 + 2 + 2 = 12$, which is much smaller than 64. Hence, the structure of BNs gives a more compact representation than JPDs in complicated problems.

2.3.2 Inference with Bayesian Networks

*Inference* is the activity of guessing the state of environment from knowledge and observation. Calculating the posterior probability distributions after observing the evidence, referred to as *belief updating*, is a common assignment in inference. There are many ways to deal with it. In this subsection, we introduce two examples of BN inference in Figure 2.3 which is a subnetwork of Figure 2.2. The CPT of *alarm*

![Figure 2.3: A simple BN of three variables](image)

<table>
<thead>
<tr>
<th>fire</th>
<th>$P(\text{alarm} = t \mid \text{fire})$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$f$</td>
<td>0.9802</td>
</tr>
<tr>
<td>$t$</td>
<td>0.017098</td>
</tr>
</tbody>
</table>

is already marginalized using the equation $P(\text{alarm} = t \mid \text{fire}) = \sum_{\text{smoke}} P(\text{alarm} = t \mid \text{fire, smoke})$. In Figure 2.3, we can see that *fire* and *leaving* are conditionally independent on *alarm*, which means that $I(\text{fire, alarm, leaving})$ holds. The first example updates the belief by a JPD in Table 2.2 and the second example updates...
the belief by message passing using CPTs [14].

<table>
<thead>
<tr>
<th>fire</th>
<th>alarm</th>
<th>leaving</th>
<th>P(fire, alarm, leaving)</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>f</td>
<td>f</td>
<td>0.972099907</td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>t</td>
<td>0.000973073</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td>f</td>
<td>0.002031242</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td>t</td>
<td>0.014895778</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>f</td>
<td>0.000197802</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>t</td>
<td>0.000000198</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>f</td>
<td>0.00117624</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
<td>0.00862576</td>
</tr>
</tbody>
</table>

[Belief Updating with JPDs]

Suppose that the evidence fire = t is observed, and we want to know the probability of people leaving the building. Processes are executed below.

According to the observation, we start with updating the JPD by keeping probabilities of fire = t unchanged and setting probabilities of fire = f to 0. Then, based on the sum rule, normalization is executed by dividing the unchanged probabilities by their sum. The result JPD after normalization is shown in Table 2.3.

<table>
<thead>
<tr>
<th>fire</th>
<th>alarm</th>
<th>leaving</th>
<th>P(fire, alarm, leaving)</th>
</tr>
</thead>
<tbody>
<tr>
<td>f</td>
<td>f</td>
<td>f</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>f</td>
<td>t</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td>f</td>
<td>0</td>
</tr>
<tr>
<td>f</td>
<td>t</td>
<td>t</td>
<td>0</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>f</td>
<td>0.0197802</td>
</tr>
<tr>
<td>t</td>
<td>f</td>
<td>t</td>
<td>0.00000198</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>f</td>
<td>0.117624</td>
</tr>
<tr>
<td>t</td>
<td>t</td>
<td>t</td>
<td>0.862576</td>
</tr>
</tbody>
</table>

We want to know the probability of leaving, so we collect data by marginaliza-
We finally have the result

\[
P(\text{leaving}|\text{fire} = t) = \sum_{\text{alarm}} P(\text{leaving},\text{alarm}|\text{fire} = t) = [0.8625958, 0.1374042],
\]

which means if the building is on fire, the probability of people leaving the building is over 0.86.

In this example, to obtain such a JPD, \(2^3 - 1 = 7\) probabilities are defined. Belief updating on this JPD including normalization and marginalization also needs updating all these probabilities. As we mentioned before, the complexity of a full JPD is \(O(\kappa^\mu)\), which is exponential on the number of variables. As the domain of the problem grows, the full JPD and inference is intractable. Therefore, belief updating upon JPDs are not efficient in complex problems.

[Belief Updating By Message Passing]

Belief can also be updated using message passing which avoids updating JPDs. Instead, we update distributions which contain only a small set of variables and then deliver their impact to other distributions through message passing. The graphical dependence structure will decide how small each distribution is and which other distributions it may affect. In this example, we use message passing over CPTs. To explain this idea, we use the same belief updating example with JPDs. To obtain the probability \(P(\text{leaving}|\text{fire} = t)\) by message passing, it needs all CPTs.

We pass the message with the direction fire → alarm → leaving based on Figure 2.3. The observation is at node fire. Let the node fire send a message to alarm that \(P(fire = t|fire = t) = 1\). Upon receiving the message, the node alarm
computes its own message and send it to the node *leaving*:

\[
P(\text{alarm} | \text{fire} = t) = P(\text{alarm} | \text{fire} = t)P(\text{fire} = t | \text{fire} = t) = [0.9802, 0.0198].
\]

The computation is based on the distribution \( P(\text{alarm} | \text{fire}) \) and the incoming message \( P(\text{fire} = t | \text{fire} = t) \). When receiving the message \( P(\text{alarm} | \text{fire} = t) \), node *leaving* obtains

\[
P(\text{leaving} | \text{fire} = t) = P(\text{leaving} | \text{alarm})P(\text{alarm} | \text{fire} = t) = [0.8625958, 0.1374042].
\]

This means when observing that the building is on *fire*, the probability of people *leaving* the building is over 0.86.

Through this example, we can see that belief updating by message passing only needs the total number of parameters in all CPTs in the BN, which is \( 1 + 2 + 2 = 5 \). In general, message passing only needs \( O(\mu \kappa^n) \) probabilities, which is linear on the number of variables. In addition, belief updating by message passing obtains the same result of that by JPDs. Therefore, belief updating by message passing is more efficient than by JPDs. More information about message passing in BNs can be found in [14].

### 2.4 Space-efficient Models

BNs represent a more compact structure that has a smaller space complexity. However, the size of CPTs is still exponential on the number of parents per node in BNs. Thus, it is still difficult to generate CPTs in complex BNs. Researchers have developed space-efficient models to reduce the number of parameters of CPTs.
such as context-specific independence by Boutilier et al. [1]. Causal independence models are one type of techniques proposed for this aim as well. CPTs in causal independence models are distributed based on causal independence among variables. Several causal independence models have been proposed, such as well-known noisy-OR attributed by Pearl [9] which can be traced back to Good [4]. Recursive noisy-OR by Lemmer and Gossink [6] develops a formulation of multi-causal interactions. To adapt noisy-OR to multi-variables, Noisy-MAX has been developed by Henrion [5] and Galan and Diez [2]. Noisy-AND by Galan and Diez [3] is another extension of noisy-OR. Binary NIN-AND tree by Xiang and Jia [18] handles different types of causal-interactions and their combinations. Multi-valued NIN-AND tree by Xiang [17] adapts NIN-AND tree to multi-valued variables. There also exist other techniques such as tensor-decomposition models by Vomlel and Tichavsky [12] and intercausal cancellation model by Woudenberg et al. [13]. In the following subsections, we begin with introducing some important terminologies. Then, we present what leaky causes are. Next, we review the causal independence models noisy-OR and binary NAT models. After that, we study noisy-MAX and multivalued NIN-AND trees. Last but not least, other models will be discussed including context-specific independence and intercausal cancellation models.

2.4.1 Terminologies

Before we introduce related models, we would like to present some terminologies that we use in this thesis. They are mostly shown in [17].

An uncertain cause is a cause that can lead to an effect but does not always do
so, which means the probability $P$ is $0 < P < 1$. For example, fire is an uncertain cause of alarm. Denote an effect by $e$ and a set of $n$ causes by $C = \{c_1, ..., c_n\}$. We use $\mathcal{C}$ to represent all causes in $C$. We use $e$ to represent alarm and $c_1$ to represent fire. Assume that each variable with $k$ values ($k \geq 2$) is indexed from 0 to $k - 1$, where a superscript means a higher intensity. They are called graded variables. For binary effect $e$, it has domain $D_e = \{e^0, e^1\}$, where $e^0$ means that $e$ is inactive or absent and $e^1$ means that $e$ is active or present. For example, $e^0$ means alarm = false, and $e^1$ means alarm = true. The domain of binary cause $c_i$ is $D_i = \{c^0_i, c^1_i\}$. For instance, $c^0_1$ means fire = false, and $c^1_1$ means fire = true. When $e$ is multi-valued, it has domain $D_e = \{e^0, e^1, ..., e^n\}$, where $e^0$ indicates the inactive value and $e^1, ..., e^n$ indicate active values. The domain of multi-valued cause $c_i$ is $D_i = \{c^0_i, c^1_i, ..., c^m_i\}$. For instance, medicine $M_1$ and $M_2$ can cure a disease $D$. We use $e$ to represent recovering from $D$, $c_1$ to represent taking $M_1$ and $c_2$ to represent taking $M_2$. Value $e^0$ means not recovering from $D$ at all; $e^1$ means partially recovered; and $e^2$ means fully recovered. Value $c^0_1$ means not taking $M_1$; $c^1_1$ means taking low dosage of $M_1$ and $c^2_1$ means taking high dosage of $M_1$. For simplicity, an active value might be written as $e^+$ or $c^+_i$.

A causal event refers to an event that a cause $c_i$ produces an effect $e$ to occur. Whether a causal event is success or failure depends on whether $e$ is active or inactive; whether a causal event is single-causal or multi-causal depends on the number of active causes; and whether a causal event is simple or congregate depends on the range of effect values. The causal event $e^0 \leftarrow c^v_i$, where $(v_i > 0)$ ($c_i$ is active) and every other cause except $c_i$ is inactive, is a simple single-causal failure. The causal
event $e \geq e^k \leftarrow c^{v_1}_{i_1}, ..., c^{v_2}_{i_2} (k, v_1, v_2 > 0)$ is a congregate multi-causal success. The probability of a causal event is referred to as a causal probability. A causal probability relates to conditional probability as

$$P(e^k \leftarrow c^{v_i}_i) = P(e^k \mid c^{v_i}_i, c^{0}_y \text{ for each } c_y \in C \text{ where } y \neq i).$$

With the disease example mentioned in the last paragraph, the causal probability $P(e^2 \leftarrow c^{2}_1)$ means the probability of fully recovering from $D$ when taking high dosage of $M_1$ and not taking $M_2$. The causal probability $P(e \geq e^1 \leftarrow c^{2}_1, c^{1}_2)$ means the probability of either partially or fully recovering from $D$ when taking high dosage of $M_1$ and low dosage of $M_2$.

Causal independence explains that the impact on the effect of one cause is independent of the impact of other causes. For example, investing on projects earns money. Investments on two different projects do not affect each other. The rate of return of one project does not affect the rate of return of the other project. Therefore, we say investments on two different projects are independent.

### 2.4.2 Leaky Causes

In the theory of causal independence models, when all causes of an effect are inactive, the effect is inactive. However, in reality, the effect might still be active when all observed causes are inactive, which means that there are causes not explicitly modeled [5]. For example, in the CPT shown in Table 2.4 which is extracted from the fire diagnosis in Figure 2.2 in the previous section, when the two causes tampering and fire are inactive, the effect alarm is still active (0.0001). Other causes such
Table 2.4: CPT with a persistent leaky cause

| tampering | fire | $P(\text{alarm} = t|\text{tampering}, \text{fire})$ |
|-----------|------|----------------------------------|
| $t$       | $t$  | 0.5                              |
| $t$       | $f$  | 0.85                             |
| $f$       | $t$  | 0.99                             |
| $f$       | $f$  | 0.0001                           |

as *manufacturing defect* may trigger *alarm*. Since the probability of them being active is very low, these causes are not explicitly modeled in the CPT of *alarm*.

A *leaky cause* represents all causes that are not explicitly modeled. For example in Figure 2.4, the variable *other* is the leaky cause. When it is relevant, we denote

![Figure 2.4: Causes of alarm with a leaky cause](Image)

the leaky cause by $c_0$ and the remaining causes by $c_1, ..., c_n$. A leaky cause may or may not be persistent. A *persistent leaky cause* is always active and a *non-persistent leaky cause* is not always active. For instance, *manufacturing defect* is usually a non-persistent leaky cause. However, if there is something wrong with the machines that make *alarms* in the factory, *manufacturing defect* would be a persistent leaky cause. As shown in Table 2.4, there exists a persistent leaky cause. A non-persistent leaky cause behaves in the same way as other normal causes that can be active or inactive. The only difference is it does not have an explicit name. Therefore, we can
add such a normal cause to model the non-persistent leaky cause. Table 2.5 is a CPT with a non-persistent leaky cause other. A CPT without leaky cause is shown in Table 2.6. In this thesis, both non-persistent leaky causes and persistent leaky causes will be considered.

Table 2.5: CPT with a non-persistent leaky cause

| tampering | fire | other | P(alarm = t| tampering, fire, other) |
|-----------|------|-------|----------------------------------|
| t         | t    | t     | 0.501                            |
| t         | f    | t     | 0.855                            |
| f         | t    | t     | 0.991                            |
| f         | f    | t     | 0.0001                           |
| t         | t    | f     | 0.5                              |
| t         | f    | f     | 0.85                             |
| f         | t    | f     | 0.99                             |
| f         | f    | f     | 0                                |

Table 2.6: CPT without leaky causes

| tampering | fire | P(alarm = t| tampering, fire) |
|-----------|------|-------------------|
| t         | t    | 0.5               |
| t         | f    | 0.85              |
| f         | t    | 0.99              |
| f         | f    | 0                 |

2.4.3 Noisy-OR Models

Noisy-OR [4, 9] is one of the most well-known binary causal models for estimating CPTs. Noisy-OR models have been shown to be equivalent to dual NIN-AND gates in NAT models [18] (will be introduced later). Therefore, for simplicity and consistency, we present noisy-OR models using the same terminologies as those for NAT models. A CPT with $n$ binary variables has $2^n$ probabilities. Noisy-OR models reduce the space complexity down to linear on the number of causes $n$ with the following three
assumptions.

1. All causes of the effect are modeled including a leaky cause if necessary.

This assumption ensures that the probability of the effect equals 0 when all modeled causes are inactive, meaning $P(e^+|c^-) = 0$.

2. Failure conjunction

*Failure conjunction* is defined as $(e^- \leftrightarrow c_1^+, \ldots, c_n^+) = (e^- \leftrightarrow c_1^+) \land \cdots \land (e^- \leftrightarrow c_n^+)$. The conjunction cause fails to produce the effect when each cause has failed to produce the effect.

3. Failure independence

*Failure independence* is the most significant assumption of the noisy-OR. The probability of the joint event is the product of probability of single events. It is defined that

$$P((e^- \leftrightarrow c_1^+) \land \cdots \land (e^- \leftrightarrow c_n^+)) = P(e^- \leftrightarrow c_1^+) \ast \cdots \ast P(e^- \leftrightarrow c_n^+).$$

Failures of causes to produce the effect are independent of each other.

With the assumptions of noisy-OR, the probability that all active causes fail to cause the effect is the product of probabilities that each active cause fails to produce the effect. Using the negation rule mentioned in the previous section, we can derive

$$P(e^+ \leftarrow c_1^+, \ldots, c_n^+) = 1 - P(e^- \leftarrow c_1^+, \ldots, c_n^+)$$

$$= 1 - (1 - P(e^+ \leftarrow c_1^+)) \ast \cdots \ast (1 - P(e^+ \leftarrow c_n^+)).$$

From the equation, we find that with assumptions of noisy-OR, only single-causal probabilities (simplified as single-causals) such as $P(e^+ \leftarrow c_i^+)$ are needed to derive
CPTs. We give a simple example with the causal interaction in Figure 2.5. Assume that cold and flu are the two binary causes of the effect fever, where cold and flu follow the noisy-OR assumption. Each variables has the value true (t) or false (f).

Two single-causals are $P(fever^+ \leftarrow cold^+) = 0.7$ and $P(fever^+ \leftarrow flu^+) = 0.8$, where $fever^+$, $cold^+$ and $flu^+$ mean $fever = t$, $cold = t$ and $flu = t$, respectively.

![Figure 2.5: A causal interaction](image)

In order to obtain the CPT of $fever$, 4 probabilities $P(fever^+|cold^+,flu^+)$, $P(fever^+|cold^+,flu^-)$, $P(fever^+|cold^-,flu^+)$ and $P(fever^+|cold^-,flu^-)$ should be obtained, and the remaining probabilities can be derived by the negation rule. However, only two of them are given, which are

$$P(fever^+|cold^+,flu^-) = P(fever^+ \leftarrow cold^+) = 0.7$$

and

$$P(fever^+|cold^-,flu^+) = P(fever^+ \leftarrow flu^+) = 0.8.$$

Since it is mentioned that cold and flu are the only two causes of fever, when cold and flu are both inactive, the probability of fever equals 0, which means

$$P(fever^+|cold^-,flu^-) = 0.$$ 

As this point, $P(fever^+|cold^+,flu^+) = P(fever^+ \leftarrow cold^+,flu^+)$ is the only unknown probability. It can be derived as follows:

$$P(fever^+|cold^+,flu^+) = 1 - P(fever^- \leftarrow cold^+,flu^+)$$

24
\[
1 - P(\text{fever}^- \leftarrow \text{cold}^+) \times P(\text{fever}^- \leftarrow \text{flu}^+)
= 1 - (1 - P(\text{fever}^+ \leftarrow \text{cold}^+)) \times (1 - P(\text{fever}^+ \leftarrow \text{flu}^+))
= 1 - (1 - 0.7) \times (1 - 0.8) = 0.94.
\]

With all probabilities obtained, the CPT of \textit{fever} is shown in Table 2.7. Thus, noisy-

| cold | flu | \(P(\text{fever}^+ | \text{cold, flu})\) |
|------|-----|----------------------------------|
| \text{f}  | \text{f} | 0                               |
| \text{f}  | \text{t} | 0.8                             |
| \text{t}  | \text{f} | 0.7                             |
| \text{t}  | \text{t} | 0.94                            |

OR models reduce the space complexity of CPTs from \(O(\kappa^n)\) which is exponential on
the number of causes \(n\) down to linear \(O(n \kappa)\), when the number of domain values \(\kappa = 2\).

### 2.4.4 Binary NIN-AND Tree Models

Consider the following example. There are three medicines \(M_1, M_2, M_3\) that
can cure a disease \(D\). Taking \(M_1\) and \(M_2\) at the same time will cancel the effectiveness of each other. Taking \(M_3\) along with taking either \(M_1\) or \(M_2\) will increase the effectiveness. We say that taking \(M_1\) and taking \(M_2\) are \textit{undermining} each other. Taking \(M_1\) (or \(M_2\)) and taking \(M_3\) are \textit{reinforcing} each other. Noisy-OR can only
models undermining causal interactions.

Non-impeding noisy-AND (NIN-AND) tree (NAT) models [18] can model both
reinforcing and undermining causal interactions as well as their mixture over binary variables. A binary \textit{NAT model} consists of a binary NAT topology and a set of
single-causals. In this subsection, knowledge of dual NIN-AND gates, direct NIN-AND gates, NAT topologies and NAT models over binary variables will be reviewed.

[Causal Interactions: Reinforcing and Undermining]

Causal interactions among causes of an effect have two different types which are reinforcing and undermining. Assume we have two causes $c_1$ and $c_2$ and an effect $e$. When

$$P(e^+ \leftarrow c_1^+, c_2^+) \geq \max(P(e^+ \leftarrow c_1^+), P(e^+ \leftarrow c_2^+))$$

holds, we say $c_1$ and $c_2$ reinforce each other since the probability of $e$ caused by joint causes is larger than that caused by each single set. On the other side, when

$$P(e^+ \leftarrow c_1^+, c_2^+) < \min(P(e^+ \leftarrow c_1^+), P(e^+ \leftarrow c_2^+))$$

holds, we say $c_1$ and $c_2$ undermine each other because the joint causes generates a smaller probability to cause $e$. A NAT model can structure both reinforcing and undermining causal interactions and their combinations. Reinforcing is modeled by a dual NIN-AND gate, and undermining is modeled by a direct NIN-AND gate.

[Binary Dual NIN-AND Gates]

Binary dual NIN-AND gates encode reinforcing causal interactions in binary NAT models. A binary dual gate is shown in Figure 2.6. We can find that the input and the output, represented by black circles, of a binary dual gate are failure events. A binary dual gate has two assumptions:

1. Failure conjunction
Figure 2.6: A binary dual NIN-AND gate

Failure conjunction means that the failure event equals the conjunction of related single failure events, which is denoted as

$$(e^− \leftarrow c_1^+, ..., c_n^+) = (e^− \leftarrow c_1^+) \land \cdots \land (e^− \leftarrow c_n^+).$$

The conjunction cause fails to produce the effect when each cause has failed to produce the effect.

2. Failure independence

Failure independence indicates that each failure event \(P(e^− \leftarrow c_i^+)\) is independent of other failure events. The probability of the joint event is the product of probability of single events. Then we can have the equation

\[
P((e^− \leftarrow c_1^+) \land \cdots \land (e^− \leftarrow c_n^+)) = P(e^− \leftarrow c_1^+) \ast \cdots \ast P(e^− \leftarrow c_n^+).
\]

With the above two assumptions, we can have the equation

\[
P(e^− \leftarrow c_1^+, ..., c_n^+) = P(e^− \leftarrow c_1^+) \ast \cdots \ast P(e^− \leftarrow c_n^+).
\]
We can derive another equation with the negation rule that

\[ P(e^+ \leftarrow c_1^+, \ldots, c_n^+) = 1 - P(e^- \leftarrow c_1^+, \ldots, c_n^+) \]

\[ = 1 - P(e^- \leftarrow c_1^+) \ast \cdots \ast P(e^- \leftarrow c_n^+) \]

\[ = 1 - (1 - P(e^+ \leftarrow c_1^+)) \ast \cdots \ast (1 - P(e^+ \leftarrow c_n^+)). \]

[Binary Direct NIN-AND Gates]

Binary direct NIN-AND gates encode undermining causal interactions in binary NAT models. A binary direct gate is shown in Figure 2.7. We can see that the input

![Figure 2.7: A binary direct NIN-AND gate](image)

and the output of a binary direct gate are success events. A binary direct gate also has two assumptions:

1. Success conjunction

Success conjunction means that the success event equals the conjunction of re-
lated single success events, which is denoted as

$$(e^+ \leftarrow c_1^+, \ldots, c_n^+) = (e^+ \leftarrow c_1^+) \land \cdots \land (e^+ \leftarrow c_n^+).$$

The conjunction cause produces the effect when each cause succeeds to produce the effect.

2. Success independence

Success independence indicates that each success event $P(e^+ \leftarrow c_i^+)$ is independent of other success events. The probability of the joint event is the product of probability of single events. Then we have the equation

$$P((e^+ \leftarrow c_1^+) \land \cdots \land (e^+ \leftarrow c_n^+)) = P(e^+ \leftarrow c_1^+) \ast \cdots \ast P(e^+ \leftarrow c_n^+).$$

With the above two assumptions, we can have the equation

$$P(e^+ \leftarrow c_1^+, \ldots, c_n^+) = P(e^+ \leftarrow c_1^+) \ast \cdots \ast P(e^+ \leftarrow c_n^+).$$

[Binary NIN-AND Trees]

Binary NAT models can structure both binary dual gates and binary direct gates into tree topologies. Thus, NAT models have the ability to model both reinforcing and undermining causal interactions as well as their mixtures over binary variables.

We use the example that we introduced at the beginning of this section. Define taking $M_1$ as binary cause $c_1$, taking $M_2$ as $c_2$, taking $M_3$ as $c_3$ and recovering from disease $D$ as binary effect $e$. Each variable has two domains \{false, true\}, simplified as \{f, t\}. For instance, $c_1^+$ means taking $M_1$ (true) and $c_1^-$ means not taking $M_1$.
(false). Figure 2.8 gives an example of a binary NAT. In this example, $c_1$ and $c_2$ are undermining each other and both of them reinforce $c_3$. Therefore, a binary direct gate connects $c_1$ and $c_2$, while $c_3$ is connected to the output of the binary directed gate by a binary dual gate.

![Figure 2.8: A binary NAT](image)

In a binary NAT, the type of two successive levels of binary NIN-AND gates are different. As long as the type of leaf gate of a NAT is given, the types of other gates are determined [26].

All causes following noisy-OR assumption are reinforcing each other. The causal interactions among causes following NAT models can be either reinforcing or undermining. Therefore, the structure of binary NAT models is a more generalized representation of causal interactions.

[Pairwise Causal Interaction]

Pairwise causal interaction (PCI) represents the causal interaction between a
pair of causes of an effect. In a binary NAT topology, each pair of causes \((c_i, c_j)\) corresponds to a PCI, denoted by \(pci(c_i, c_j)\). To define \(pci(c_i, c_j)\), we need to find the type of the first gate whose output causal event contains both \(c_i\) and \(c_j\). Particularly, if we find the first gate to be a binary dual gate, we define \(pci(c_i, c_j) = r\), which means \(c_i\) and \(c_j\) reinforce each other. On the other hand, if the gate is found to be a binary direct gate, we define \(pci(c_i, c_j) = u\), which means \(c_i\) and \(c_j\) undermine each other.

For instance, in Figure 2.8, we find the first gate whose output contains \(c_1\) and \(c_2\) to be a binary direct gate, we define \(pci(c_1, c_2) = u\). By using the same method, we find \(pci(c_1, c_3) = r\) and \(pci(c_2, c_3) = r\).

Consider a set \(C\) of \(n\) causes. There are \(\binom{n}{2} = \frac{n(n-1)}{2}\) pairs of causes. For example, if \(e\) has three causes, there are \(\binom{3}{2} = 3\) pairs of causes. A PCI pattern is a bit string that contains all PCI bits. For instance, with the order \(((c_1, c_2), (c_1, c_3), (c_2, c_3))\), the PCI pattern of Figure 2.8 is \((u, r, r)\). The PCI pattern for each NAT is unique [22].

[CPT Specification with binary NAT Models]

Binary NAT models reduce the space complexity of being exponential on the number of causes \(n\) to fully linear. We show how to obtain a CPT given a binary NAT model here. As mentioned that a binary NAT model consists of a binary NAT topology and all single causes. We use Figure 2.8 to be the NAT topology, and assume the single-causals as \(e^+ ← c_1^+ = 0.8\), \(e^+ ← c_2^+ = 0.7\) and \(e^+ ← c_3^+ = 0.6\). With the NAT model, we have Table 2.8. There are 5 probabilities that need to be derived.
Table 2.8: CPT with only single-causals specified

| $c_1$ | $c_2$ | $c_3$ | $P(e^+|c_1, c_2, c_3)$ |
|-------|-------|-------|------------------------|
| f     | f     | f     | -                      |
| f     | f     | t     | 0.6                    |
| f     | t     | f     | 0.7                    |
| f     | t     | t     | -                      |
| t     | f     | f     | 0.8                    |
| t     | f     | t     | -                      |
| t     | t     | f     | -                      |
| t     | t     | t     | -                      |

At line 1 all causes are inactive, which means $P(e^+|c_1^-, c_2^-, c_3^-) = 0$.

At line 4, $c_1$ is inactive while $c_2$ and $c_3$ are active. We find that $c_2$ and $c_3$ are connected by a binary dual gate. Thus, we can calculate the probability

$$P(e^+|c_1^-, c_2^+, c_3^+) = P(e^+ ← c_2^+, c_3^+)$$

$$= 1 - (1 - P(e^+ ← c_2^+))(1 - P(e^+ ← c_3^+))$$

$$= 1 - (1 - 0.7) * (1 - 0.6) = 0.88.$$ 

At line 6, $c_2$ is inactive while $c_1$ and $c_3$ are active. We see that $c_1$ and $c_3$ are also connected by a binary dual gate. Thus, we have the probability

$$P(e^+|c_1^+, c_2^-, c_3^+) = P(e^+ ← c_1^+, c_3^+)$$

$$= 1 - (1 - P(e^+ ← c_1^+))(1 - P(e^+ ← c_3^+))$$

$$= 1 - (1 - 0.8) * (1 - 0.6) = 0.92.$$ 

At line 7, $c_3$ is inactive while $c_1$ and $c_2$ are active. We notice that $c_1$ and $c_2$ are connected by a binary direct gate. Thus, we have the probability

$$P(e^+|c_1^+, c_2^+, c_3^-) = P(e^+ ← c_1^+, c_2^+)$$

32
At line 8, all causes are active. Since $c_1$ and $c_2$ are undermining each other and both are reinforcing $c_3$, with the probability of $P(e^+ \leftarrow c_1^+, c_2^+)$ above, we can calculated the probability

$$P(e^+|c_1^+, c_2^+, c_3^+) = P(e^+ \leftarrow c_1^+, c_2^+, c_3^+)$$

$$= 1 - (1 - P(e^+ \leftarrow c_1^+, c_2^+))(1 - P(e^+ \leftarrow c_3^+))$$

$$= 1 - (1 - 0.56) \times (1 - 0.6) = 0.824.$$ 

Eventually, the CPT derived from the NAT model is shown in Table 2.9. More efficient algorithms to improve computing conditional probabilities can be found in [15].

Table 2.9: Full CPT derived from the NAT model

| $c_1$ | $c_2$ | $c_3$ | $P(e^+|c_1, c_2, c_3)$ |
|-------|-------|-------|------------------------|
| $f$   | $f$   | $f$   | 0                      |
| $f$   | $f$   | $t$   | 0.6                    |
| $f$   | $t$   | $f$   | 0.7                    |
| $f$   | $t$   | $t$   | 0.88                   |
| $t$   | $f$   | $f$   | 0.8                    |
| $t$   | $f$   | $t$   | 0.92                   |
| $t$   | $t$   | $f$   | 0.56                   |
| $t$   | $t$   | $t$   | 0.824                  |

2.4.5 Noisy-MAX Models

When dealing with complex problems in real world, binary variables have limited power to model the problems precisely. For instance, when describing fever
in real world, it can be normal, low fever or high fever. Each of them represents a
different level of the physical condition of a patient. Therefore, extending the causal
independence model to multi-valued variables is necessary.

Noisy-MAX [2, 5] extends noisy-OR from over binary variables to multi-valued
variables. Noisy-MAX models have been shown to be equivalent to multi-valued dual
NIN-AND gates [21] (will be presented later). Thus, for simplicity and consistency,
we present noisy-MAX models using the same terminologies as those for NAT models.
The assumptions of noisy-MAX are listed as follows.

1. If all causes of the effect are inactive, the effect is inactive. It can be denoted
by $P(e^0 | c^0_1, ..., c^0_n) = 1$ and $P(e^+ | c^0_1, ..., c^0_n) = 0$, where $e^+$ represents all active values
of $e$.

2. Congregate Failure Conjunction

The conjunction cause fails to produce the effect at or higher than level $k$ when
each cause has failed to produce the effect at or higher than level $k$, denoted by

$$(e < e^k \leftarrow c^1_1, ..., c^1_n) = (e < e^k \leftarrow c^1_1) \land \cdots \land (e < e^k \leftarrow c^1_n).$$

3. Congregate Failure Independence

Relative to a graded cause, failures of causes to produce the effect at or higher
than level $k$ are independent of each other. The probability of the joint event is the
product of probability of single events. It is denoted by

$$P((e < e^k \leftarrow c^1_1) \land \cdots \land (e < e^k \leftarrow c^1_n)) = P(e < e^k \leftarrow c^1_1) \ast \cdots \ast P(e < e^k \leftarrow c^1_n).$$

We can derive simple probabilities by using congregate probabilities such as
\( P(e^{k-1} \leftarrow c_{j1}^{i}, ..., c_{jn}^{i}) = P(e < e^k \leftarrow c_{j1}^{i}, ..., c_{jn}^{i}) - P(e < e^{k-1} \leftarrow c_{j1}^{i}, ..., c_{jn}^{i}) \) and vice versa. The CPTs can be obtained in a similar way of noisy-OR models. With the assumption of noisy-MAX, probabilities \( P(e^k \leftarrow c_i^j) \) relative to each active value of \( e \) need to be derived. The space complexity of noisy-MAX is \( O(n \eta) \), where \( \eta \) indicates the largest domain size among the effect and all the causes. Therefore, the space complexity of noisy-MAX is still linear on the number of causes.

For example, in Figure 2.5, assume that the domain of multi-valued variable \( \text{fever} \), denoted by \( FE \), is \{normal, low, high\} (simplified as \{n, l, h\}). The domain of \( \text{flu} \), denoted by \( FL \), is \{true, false\} (simplified as \{t, f\}). The domain of multi-valued \( \text{cold} \), denoted by \( CO \) is \{normal, mild, bad\} (simplified as \{n, m, b\}). Assume these variables follow the noisy-MAX assumption. The simple single-causal probabilities below are given.

\[
\begin{align*}
P(FE = l | CO = m) &= 0.70, \quad P(FE = h | CO = m) = 0.20, \\
P(FE = l | CO = b) &= 0.30, \quad P(FE = h | CO = b) = 0.65, \\
P(FE = l | FL = t) &= 0.35, \quad P(FE = h | FL = t) = 0.60,
\end{align*}
\]

The steps of obtaining \( P(FE = l | CO = b, FL = t) \) is shown as follows. First
we calculate $P(\text{FE} < l|\text{CO} = b, \text{FL} = t)$.

\[
P(\text{FE} < l|\text{CO} = b, \text{FL} = t) \\
= P(\text{FE} < l|\text{CO} = b) \cdot P(\text{FE} < l|\text{FL} = t) \\
= (1 - P(\text{FE} = l|\text{CO} = b) - P(\text{FE} = h|\text{CO} = b)) \\
\times (1 - P(\text{FE} = l|\text{FL} = t) - P(\text{FE} = h|\text{FL} = t)) \\
= (1 - 0.30 - 0.65) \times (1 - 0.35 - 0.60) = 0.0025
\]

Then, we calculate $P(\text{FE} < h|\text{CO} = b, \text{FL} = t)$ in a similar way.

\[
P(\text{FE} < h|\text{CO} = b, \text{FL} = t) \\
= P(\text{FE} < h|\text{CO} = b) \cdot P(\text{FE} < h|\text{FL} = t) \\
= (1 - P(\text{FE} = h|\text{CO} = b)) \times (1 - P(\text{FE} = h|\text{FL} = t)) \\
= (1 - .065) \times (1 - 0.60) = 0.14
\]

Finally, $P(\text{FE} = l|\text{CO} = b, \text{FL} = t)$ can be derived as follows.

\[
P(\text{FE} = l|\text{CO} = b, \text{FL} = t) \\
= P(\text{FE} < h|\text{CO} = b, \text{FL} = t) - P(\text{FE} < l|\text{CO} = b, \text{FL} = t) \\
= 0.14 - 0.0025 = 0.1375
\]

The resultant probability is underlined in Table 2.10. Other probabilities can be derived in the similar way. The resultant CPT is shown in Table 2.10.

### 2.4.6 Multi-valued NIN-AND Tree Models

Noisy-MAX extended noisy-OR to over multi-valued variables, but it is still limited to model only reinforcing causal interactions. Binary NAT models has been
Table 2.10: CPT of multi-valued flu fever and cold

| CO | FL | P(FE = n|CO, FL) | P(FE = l|CO, FL) | P(FE = h|CO, FL) |
|----|----|-----------------|-----------------|-----------------|
| n  | f  | 1.0             | 0.0             | 0.0             |
| n  | t  | 0.05            | 0.35            | 0.6             |
| m  | f  | 0.1             | 0.7             | 0.2             |
| m  | t  | 0.005           | 0.315           | 0.68            |
| b  | f  | 0.05            | 0.3             | 0.65            |
| b  | t  | 0.0025          | 0.1375          | 0.86            |

Extended to multi-valued variables [17]. In this subsection, multi-valued NAT models are reviewed.

[Causal Interactions over Multi-valued Variables]

The definition of causal interactions over multi-valued variables are slightly different from binary ones. Assume we have two multi-valued causes $c_1$ and $c_2$ and a multi-valued effect $e$. When

$$P(e \geq e^k \leftarrow c_1^+, c_2^+) < \min(P(e \geq e^k \leftarrow c_1^+), P(e \geq e^k \leftarrow c_2^+)),$$

where $e^k$ is an active value for $e$, holds for all active values of $c_1$ and $c_2$, we say $c_1$ and $c_2$ reinforce each other relative to $e^k$. On the other side, when

$$P(e \geq e^k \leftarrow c_1^+, c_2^+) \geq \max(P(e \geq e^k \leftarrow c_1^+), P(e \geq e^k \leftarrow c_2^+))$$

holds for all active values of $c_1$ and $c_2$, we say $c_1$ and $c_2$ undermine each other relative to $e^k$. Therefore, PCI over multi-valued variables is relative to $e^k$, such as $pci(e^k, c_1, c_2)$. Therefore, a PCI pattern for a multi-valued NAT is relative to $e^k$ as well.

[Multi-valued Dual NIN-AND Gates]
The input and output of a multi-valued dual NIN-AND gate are congregate causal failure, which is different from a binary dual gate. A congregate causal failure is denoted $e < e^k \leftarrow c_i^j$ ($k, j > 0$). It explains that if cause $c_i$ is active, with other causes being inactive, it fails to produce the effect at or higher than level $k$. A multi-valued dual gate is illustrated in Figure 2.9.

A multi-valued dual gate has the following two assumptions.

1. Congregate Failure Conjunction

The conjunction cause fails to produce the effect at or higher than level $k$ when each cause has failed to produce the effect at or higher than level $k$, denoted by

$$(e < e^k \leftarrow c_1^{j_1}, ..., c_n^{j_n}) = (e < e^k \leftarrow c_1^{j_1}) \land \cdots \land (e < e^k \leftarrow c_n^{j_n}).$$

2. Congregate Failure Independence

Relative to a graded cause, each failure of causes to produce the effect at or higher than level $k$ is independent of others. The probability of the joint event is the
product of probability of single events. It is denoted by

\[ P((e < e^k \leftarrow c_{j_1}^{i_1}) \land \cdots \land (e < e^k \leftarrow c_{j_n}^{i_n})) = P(e < e^k \leftarrow c_{j_1}^{i_1}) \cdots P(e < e^k \leftarrow c_{j_n}^{i_n}). \]

Therefore, with the above assumptions, we derive that the probability of multi-causal failure is the product of probabilities of single-causal failure events, denoted by

\[ P(e < e^k \leftarrow c_{j_1}^{i_1}, ..., c_{j_n}^{i_n}) = P(e < e^k \leftarrow c_{j_1}^{i_1}) \cdots P(e < e^k \leftarrow c_{j_n}^{i_n}). \]

[Multi-valued Direct NIN-AND Gates]

A multi-valued direct NIN-AND gate has the input and output as congregate causal success as well, which is different from a binary direct gate. A congregate causal success is denoted \( e \geq e^k \leftarrow c_i^{j_i} (k, j > 0) \). It explains that if cause \( c_i \) is active, with other causes being inactive, it produces the effect at or higher than level \( k \). A multi-valued direct gate is illustrated in Figure 2.10.

![Figure 2.10: A multi-valued NIN-AND direct gate](image)
A multi-valued direct gate also has two assumptions as follows.

1. Congregate Success Conjunction

The conjunction cause produces the effect at or higher than level $k$ when each cause has succeeded to produce the effect at or higher than level $k$, denoted by

$$(e \geq e^k \leftarrow c^j_1, ..., c^j_n) = (e \geq e^k \leftarrow c^j_1) \wedge \cdots \wedge (e \geq e^k \leftarrow c^j_n).$$

2. Congregate Success Independence

Relative to a graded cause, each success of causes to produce the effect at or higher than level $k$ is independent of others. The probability of the joint event is the product of probability of single events. It is denoted by

$$P((e \geq e^k \leftarrow c^j_1) \wedge \cdots \wedge (e \geq e^k \leftarrow c^j_n)) = P(e \geq e^k \leftarrow c^j_1) \ast \cdots \ast P(e \geq e^k \leftarrow c^j_n).$$

Therefore, with the above assumptions, we derive that the probability of multi-causal success is the product of probabilities of single-causal success events, denoted by

$$P(e \geq e^k \leftarrow c^j_1, ..., c^j_n) = P(e \geq e^k \leftarrow c^j_1) \ast \cdots \ast P(e \geq e^k \leftarrow c^j_n).$$

[Multi-valued NIN-AND Trees]

Multi-valued NAT models structure both multi-valued dual gates and direct gates. There are two main differences of multi-valued NAT from binary NAT. First, the input and output of the multi-valued gates are congregate events, but they are simple events in binary gates. Second, each congregate event is relative to a specific value of variables, and those values must keep the same.
Figure 2.11: A multi-valued NAT

Figure 2.11 is a multi-valued NAT with three causes. We extend the example that we used for binary NAT. Define taking $M_1$ as multi-valued cause $c_1$, taking $M_2$ as $c_2$, taking $M_3$ as $c_3$ and recovering from disease $D$ as multi-valued effect $e$. Each cause has the domain \{no, low, high\}. For example, $c_1^0$ means not taking $M_1$; $c_1^1$ means taking low dosage of $M_1$ and $c_1^2$ means taking high dosage of $M_1$. The effect has three domains \{no, partially recovered, fully recover\}. It explains that $e^0$ means not recovering from $D$ at all; $e^1$ means partially recovered and $e^3$ means fully recovered. We have that $c_1$ and $c_2$ are undermining each other and both are reinforcing $c_3$. Thus, $c_1$ and $c_2$ are connected with a direct gate, and $c_3$ is connected to the output of the direct gate with a dual gate. We can see that the input and output of the direct gate are congregate success events. After reversing the congregate success of $e \geq e^1 \leftarrow c_1^1$, $c_2^2$ into a congregate failure (the white circle representing $e < e^1 \leftarrow c_1^1$, $c_2^2$), the input and output of the dual gate are congregate failure events. In addition, it can be easily noticed that the value of each related variable keeps the same.
The rules of deriving CPTs from multi-valued NAT models are similar to that from binary NAT models. The only difference is that congregate probabilities are used. For example, we use the NAT as shown in Figure 2.11, where the domain sizes of variables keep the same. Simple single-causal probabilities are given as follows.

\[
P(e^1 \leftarrow c^1_1) = 0.20, \quad P(e^2 \leftarrow c^1_1) = 0.40; \quad P(e^1 \leftarrow c^2_1) = 0.30, \quad P(e^2 \leftarrow c^2_1) = 0.50;
\]

\[
P(e^1 \leftarrow c^1_2) = 0.25, \quad P(e^2 \leftarrow c^1_2) = 0.35; \quad P(e^1 \leftarrow c^2_2) = 0.45, \quad P(e^2 \leftarrow c^2_2) = 0.35;
\]

\[
P(e^1 \leftarrow c^1_3) = 0.35, \quad P(e^2 \leftarrow c^1_3) = 0.60; \quad P(e^1 \leftarrow c^2_3) = 0.15, \quad P(e^2 \leftarrow c^2_3) = 0.55.
\]

The steps of obtaining the probability \(P(e^1 = c^1_1, c^2_2, c^1_3)\) is shown as follows.

First, we obtain the input congregate single-causal probabilities.

\[
P(e \geq e^1 \leftarrow c^1_1) = P(e^1 \leftarrow c^1_1) + P(e^2 \leftarrow c^1_1) = 0.2 + 0.4 = 0.6
\]

\[
P(e \geq e^1 \leftarrow c^2_2) = P(e^1 \leftarrow c^2_2) + P(e^2 \leftarrow c^2_2) = 0.45 + 0.35 = 0.8
\]

\[
P(e \geq e^1 \leftarrow c^1_3) = P(e^1 \leftarrow c^1_3) + P(e^2 \leftarrow c^1_3) = 0.35 + 0.6 = 0.95
\]

Since \(c_1\) and \(c_2\) are undermining each other, we then calculate:

\[
P(e \geq e^1 \leftarrow c^1_1, c^2_2) = P(e \geq e^1 \leftarrow c^1_1) \cdot P(e \geq e^1 \leftarrow c^2_2) = 0.48
\]

Then, since \(c_1\) and \(c_2\) both reinforce \(c_3\), we calculate:

\[
P(e \geq e^1 \leftarrow c^1_1, c^2_2, c^1_3) = 1 - P(e < e^1 \leftarrow c^1_1, c^2_2, c^1_3)
\]

\[
= 1 - P(e < e^1 \leftarrow c^1_1, c^2_2) \cdot P(e < e^1 \leftarrow c^1_3)
\]

\[
= 1 - (1 - P(e \geq e^1 \leftarrow c^1_1, c^2_2)) \cdot (1 - P(e \geq e^1 \leftarrow c^1_3))
\]

\[
= 1 - (1 - 0.48) \cdot (1 - 0.95) = 0.974
\]
We can use the similar way to calculate that \( P(e \geq e^2 \leftarrow c_1^2, c_2^2, c_3^1) = 0.656 \). Finally, we get the result that

\[
P(e^1 = c_1^1, c_2^2, c_3^1) = P(e \geq e^1 \leftarrow c_1^1, c_2^2, c_3^1) - P(e \geq e^2 \leftarrow c_1^1, c_2^2, c_3^1)
\]

\[
= 0.974 - 0.656 = 0.318
\]

The remaining probabilities can be derived in a similar way. The resultant CPT is shown in Table 2.11, where the probability calculated above is underlined.

| \( c_1 \) | \( c_2 \) | \( c_3 \) | \( P(e^0|c_1, c_2, c_3) \) | \( P(e^1|c_1, c_2, c_3) \) | \( P(e^2|c_1, c_2, c_3) \) |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 1.0 | 0.0 | 0.0 |
| 0 | 0 | 1 | 0.05 | 0.35 | 0.6 |
| 0 | 0 | 2 | 0.3 | 0.15 | 0.55 |
| 0 | 1 | 0 | 0.4 | 0.25 | 0.35 |
| 0 | 1 | 1 | 0.02 | 0.24 | 0.74 |
| 0 | 1 | 2 | 0.12 | 0.1725 | 0.7075 |
| 0 | 2 | 0 | 0.2 | 0.45 | 0.35 |
| 0 | 2 | 1 | 0.01 | 0.25 | 0.74 |
| 0 | 2 | 2 | 0.06 | 0.2325 | 0.7075 |
| 1 | 0 | 0 | 0.4 | 0.2 | 0.4 |
| 1 | 0 | 1 | 0.02 | 0.22 | 0.76 |
| 1 | 0 | 2 | 0.12 | 0.15 | 0.73 |
| 1 | 1 | 0 | 0.64 | 0.22 | 0.14 |
| 1 | 1 | 1 | 0.032 | 0.312 | 0.656 |
| 1 | 1 | 2 | 0.192 | 0.195 | 0.613 |
| 1 | 2 | 0 | 0.52 | 0.34 | 0.14 |
| 1 | 2 | 1 | 0.026 | 0.318 | 0.656 |
| 1 | 2 | 2 | 0.156 | 0.231 | 0.613 |
| 2 | 0 | 0 | 0.2 | 0.3 | 0.5 |
| 2 | 0 | 1 | 0.01 | 0.19 | 0.8 |
| 2 | 0 | 2 | 0.06 | 0.165 | 0.775 |
| 2 | 1 | 0 | 0.52 | 0.305 | 0.175 |
| 2 | 1 | 1 | 0.026 | 0.304 | 0.67 |
| 2 | 1 | 2 | 0.156 | 0.21525 | 0.62875 |
| 2 | 2 | 0 | 0.36 | 0.465 | 0.175 |
| 2 | 2 | 1 | 0.018 | 0.312 | 0.67 |
| 2 | 2 | 2 | 0.108 | 0.26325 | 0.62875 |
2.4.7 Other Existing Models

Researchers have developed other space-efficient models to reduce the space complexity. In this subsection, we review context-specific independence and tensor-decomposition models.

[Context-Specific Independence]

Context-specific independence (CSI) [1] is another technique that reduces the space complexity of CPTs in BNs. The basic idea of CSI is to capture the independence produced by particular variable assignments. After the local independence is found, CSI adds the assignment to regularity in distributions. Assume a JPD is over the variables set $U$. Let $X$, $Y$, $Z$, $C$ be pairwise disjoint sets of variables and each of them is a subset of $U$. It is defined that $X$ and $Y$ are contextually independent given $Z$ and the context $c \in \text{val}(C)$, where $\text{val}(C)$ means a value in $C$, if $P(X|Z,c,Y) = P(X|Z,c)$ holds.

CSI is focused on the local independence which we illustrate with an example. Consider a variable $X$ has 5 causes $A$, $B_1, \ldots, B_4$, as shown in Figure 2.12. For simplicity, assume all 6 variables are binary. Intuitively, $X$ is viewed as the outcome

![Figure 2.12: A simple BN of five causes](image)
of two conditional variables: $X_{A=t}$ and $X_{A=f}$. Then, we define a new variable $X_{A=t}$ such that $P(X_{A=t}|B_1, ... B_4) = P(X|A = t, B_1, ... B_4)$. We can define a new variable $X_{A=f}$ in a similar way. After a series of experiments using CSI, the simulation shows that $X$ only depends on $B_1$ and $B_2$ when $A = t$, and only depends on $B_3$ and $B_4$ when $A = f$. Based on the result, the BN is changed into Figure 2.13. The size of

![Figure 2.13: The BN representation utilizing CSI](image)

the CPT of $X$ before CSI is $2^5 = 32$. After CSI, the size of CPT of $X$ is $2^3 = 8$, and the size of CPTs of $X_{A=t}$ or $X_{A=f}$ is $2^2 = 4$. Therefore, the size of CPTs after CSI is $8 + 4 + 4 = 16$. Obviously, the space complexity is reduced. In addition, CSI can be applied over multi-valued variables.

CSI has its limitations in reducing the space complexity. First, it changes the structure of the original BN by adding more nodes. Second, it cannot promise to significantly reduce the space complexity of BNs since the variables are randomly chosen. Third, CSI explores different CPT representations such as decision graphs, whose structure may affect the result.

[Intercausal Cancellation Models]
Intercausal cancellation models [13] improves the elicitation of probabilities by adding a regulatory nodes. The main advantage of Intercausal cancellation model is not reducing the space complexity, but to speed up defining the interactions of causes. Intercausal cancellation can be modeled as shown in Figure 2.14.

![Figure 2.14: An intercausal cancellation model](image)

Consider two causes $c_1$ and $c_2$ and the effect $E$. The two variables $X_{c_1}$ and $X_{c_2}$ represent the two causes through which the process retrieves the common effect $E$. The inhibitor variables $I_1$ and $I_2$ catch the exception mechanisms that might inhibit the effect. The variables $A$ describes the cancellation effect among two causes, which is called the annihilator variable. $A$ is active only when $c_1$ and $c_2$ are both active. $E$ is active when at least one of $X_{c_1}$ or $X_{c_2}$ is active. With the probabilities $P(X_{c_1}|c_1, A)$, we can tell the degree to which $c_1$ is canceled with $c_2$ being simultaneous active. If $P(X_{c_1}|c_1, A) = P(X_{c_1}|c_1)$, we say $c_1$ is not canceled by $c_2$ being simultaneous. If $P(X_{c_1}|c_1, A) = 0$, we say $c_1$ is full canceled. With the degree of cancellation, we can easily obtain the interaction between $c_1$ and $c_2$. 
2.5 Approximating Binary CPTs with NAT Models

As we mentioned, NAT models can reduce the space complexity of CPTs from $O(\kappa^n)$ to $O(\kappa n)$, where $n$ denotes the number of causes and $\kappa$ denotes the largest domain size within all variables. The upcoming problem is how to approximate a target CPT into a NAT model. Xiang and Liu [23] proposed a method to approximate CPTs with NAT models over binary variables.

Consider compressing a target CPT $P_T$ into a NAT model $M$ which means to approximate $P_T$ by $P_M$, where $P_M$ is the CPT derived from $M$. The NAT topology of $M$ is obtained by a PCI pattern (introduced in Subsection 2.4.4). We use $e$ to denote the effect and $C = \{c_1, \ldots, c_n\}$ to denote the set of causes. The first important key to the approximation is CPT approximation accuracy. *Euclidean distance*, denoted by ED, is used to measure the accuracy. The function to calculate ED is

$$ED(P_T, P_M) = \sqrt{\frac{1}{K} \sum_{i=1}^{\kappa} (P_T(e^+|c_i^+) - P_M(e^+|c_i^+))^2},$$

where $K$ denotes the size of $P_T$ and $c_i^+$ represents all active causes. The range of ED is [0,1] which matches the range of probability. When $ED(P_T, P_M) = 0$, it is a perfect approximation. However, it is not possible for every $P_T$. CPT approximation by NAT models over binary variables is reviewed in this section.

2.5.1 Partial PCI Pattern Identification

To define a PCI pattern, we start with defining PCI bits. A bit $pci(c_i, c_j) = u$ means that the causal interaction between $c_i$ and $c_j$ is undermining. It is well-defined
if $P(e^+ \leftarrow c_i^+, c_j^+) < \min(P(e^+ \leftarrow c_i^+), P(e^+ \leftarrow c_j^+))$. A PCI bit $pci(c_i, c_j) = r$ is well-defined if $P(e^+ \leftarrow c_i^+, c_j^+) \geq \max(P(e^+ \leftarrow c_i^+), P(e^+ \leftarrow c_j^+))$ and it means $c_i$ and $c_j$ reinforce each other and . A PCI bit $pci(c_i, c_j) = null$ means that it is not well-defined if $P(e^+ \leftarrow c_i^+) < P(e^+ \leftarrow c_i^+, c_j^+) < P(e^+ \leftarrow c_j^+)$. Thus, $pci(c_i, c_j) \in \{u, r, null\}$. Some PCI bits may not be well-defined, unless the target CPT is already a NAT. Therefore, it does not always yield a fully-defined PCI pattern with all PCI bits defined.

Soft PCI identification has been designed [25]. If $c_i$ and $c_j$ are closer to undermining than reinforcing, treat them as undermining. This means if

$$|P(e^+ \leftarrow c_i^+, c_j^+) - \min(P(e^+ \leftarrow c_i^+), P(e^+ \leftarrow c_j^+))|$$

$$< |P(e^+ \leftarrow c_i^+, c_j^+) - \max(P(e^+ \leftarrow c_i^+), P(e^+ \leftarrow c_j^+))|$$

holds, we define $pci(c_i, c_j) = u$. Otherwise, $pci(c_i, c_j) = r$. If we use soft PCI identification, a fully-defined PCI pattern can be obtained. This means that there would be only one candidate NAT. However, due to the arbitrary CPT, the candidate NAT obtained by the fully-defined PCI pattern may not be the best NAT. A fully-defined PCI pattern may not always yield an accurate $P_M$. Therefore, a well-defined PCI pattern with some undefined bits improves the accuracy of the CPT approximation. Such a PCI pattern is called a partial PCI pattern.

### 2.5.2 PCI Pattern Based Search Trees and Candidate NAT Search

If we use the given PCI pattern to search for candidate NATs through linear search, it is not efficient since the number of NATs increases super-exponentially on
the number of cause \( n \) [24]. The number of NATs with causes between 3 and 9 is shown in Table 2.12. With the PCI pattern, a structure named *PCI pattern based search tree* (PST) is developed to retrieve candidate NATs. PSTs and NAT structures are enumerated and stored ahead of time since all of them can be reused.

A PCI pattern defined by a NAT with \( n \) causes has \( N = \binom{n}{2} \) bits. A PST has \( N + 1 \) levels indexed 0 to \( N \). Level 0 is the root \( t \). Each level \( l > 0 \) is related to a PCI bit, and each node at each level \( l > 0 \) is labeled with \( u \) or \( r \), representing undermining and reinforcing. All leaf nodes are at level \( N \). A leaf \( z \) exist if and only if there exists a path from \( t \) to \( z \) which represents a PCI pattern of a NAT. Each NAT with \( n \) causes is assigned to each leaf node in a PST for \( n \).

A PST for \( n = 3 \) is shown in Figure 2.15. It has 4 level \( (N = \binom{3}{2} = 3) \). Level 0 is the root and levels 1 to 3 are labeled by PCI bit values. The leaf gate of a NAT has two types. When the leaf gate of the same NAT topology is dual gate or direct gate, it can represent two different NATs. Thus, we represent a NAT by keeping input causes, the tree topology and the type of leaf gate. We omit all causal events and types of other gates [26]. The leaf gate of each assigned NAT is a direct gate. Every other leaf node is connected to one of the same group of NAT topologies, but the difference is that the leaf gate is a dual gate. From a PCI pattern \((u, r, u)\), with the order \(((c_1, c_2), (c_1, c_3), (c_2, c_3))\), we can retrieve the NAT topology indexed 2 with a direct leaf gate. With this method, a partial PCI pattern leads to a set of

<table>
<thead>
<tr>
<th>No. of causes ( n )</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>No. of NATs</td>
<td>8</td>
<td>52</td>
<td>472</td>
<td>5504</td>
<td>78416</td>
<td>1320064</td>
<td>25637824</td>
</tr>
</tbody>
</table>
candidate NATs. For example, PCI pattern \((u, r, \_\_)\), where \(\_\) represents the PCI bit that is undefined \((null)\), leads to NAT topologies indexed 2 and 3. With one more PCI bits undefined, the number of candidate NATs will be doubled. In this example, if 3 PCI bits are all undefined, there will be \(2^3 = 8\) candidate NATs.

### 2.5.3 Parameter Search by Gradient Descent

After retrieving candidate NATs, we calculate single-causals for each candidate. There are single-causals existing in \(P_T\), but they are unlikely to yield an accurate \(P_M\). The method used to calculate single-causals is gradient descent, where a point moves along a multi-dimensional function in the direction of deepest descent, until
the gradient is less than the threshold. A 3-dimensional gradient descent is shown in Figure 2.16.

Figure 2.16: A 3-dimensional gradient descent

Descent is guided by the function $ED(P_T, P_M)$ between the target CPT $P_T$ and the CPT $P_M$ generated from the NAT model $M$. The search space has $n$ dimensions, and the point is a vector of $n$ single-causals. Each single-causal is bounded in $(0, 1)$.

After the single-causal search for all candidate NATs are obtained, with the function $ED(P_T, P_M)$, the NAT model with the smallest ED is chosen as the resultant NAT model. Eventually, the CPT approximation is finished.

2.5.4 Experimental Result of Binary CPT Approximation

The experimental data shows that binary CPT approximation by NAT models is better than noisy-OR. First, CPT approximation by NAT models can be applied on the CPTs of noisy-OR models. The runtime and ED are identical. Second, given the
same set of target CPTs, the resultant ED of CPT approximation by NAT models (0.179) is lower than that of noisy-OR (0.255). CPT Approximation by noisy-OR models needs less runtime than that by NAT models since Noisy-OR searches only one tree but NAT searches a set of candidates. Therefore, CPT approximation by noisy-OR models is faster, but CPT approximation by NAT models is more accurate. In addition, the result shows that if the number of undefined bits of PCI patterns increases, the accuracy of binary NAT approximation is improved with larger runtime.

2.6 Bayesian Network Inference with NAT Models

To speed up BN inference with NAT models, Multiplicative Factorization (MF) is applied to binary NAT models [16], and extended to multi-valued NAT models [21]. Assume that the effect \( e \) with the domain \( \{ e^0, e^1, ...e^n \} \) has \( n \) causes each of which has the domain \( \{ c^0_i, ..., c^n_i \} \) (\( i = 1, ..., n \)). Since a NAT may have two different types of NIN-AND gates which are dual gates and direct gates, a gate MF can be defined by two different ways based on the gate type.

![Figure 2.17: Hybrid graph for MF of a gate](image)

Figure 2.17: Hybrid graph for MF of a gate
Figure 2.17 illustrates how MF is organized for a gate (direct or dual). Auxiliary variables \(d_j (j = 1, ..., \eta)\) are created for each active value of \(e\). The link between each pair of \(c_i\) and \(d_j\) is undirected and it is called \(clink\) since \(c_i\) is a cause. The link between each \(d_j\) and \(e\) is directed. A potential \(f(d_j, c_i)\) is assigned to each clink. A general potential \(f(d_1, ..., d_\eta, e)\) is assigned to \(e\) over its \(family\ potential\). A product of potentials is obtained to marginalize out some variables such as

\[
f(e, c_1, ..., c_n) = \sum_{d_1, ..., d_n} f(d_1, ..., d_\eta, e) \prod_{1 \leq j \leq \eta, 1 \leq i \leq n} f(d_j, c_i).
\]

A NAT model consists of a NAT topology and a set of single-causal probabilities. An example of a NAT topology is shown in Figure 2.18 (a), where causal event labels are simplified by causes and the white circle into gates are omitted. MF of a NAT model contains a hybrid graph and a set of potentials. The potentials are derived over each undirected links and each family in the hybrid graph. The MF graph is made up by gate MFs based on the NAT topology. The corresponding MF graph of the example NAT topology in Figure 2.18 (a) is shown in Figure 2.18 (b). For example, \(g_4\) in (a) is replaced by a subgraph \(\{c_1, c_2, a_1, a_2, b_1\}\). We can notice that two auxiliary variables \(a_1\) and \(a_2\) are added, and the \(internal\ variable\) is labeled by \(b_1\). Since the internal variable \(b_1\) is not a cause variable, the link \(\langle b_1, d_1 \rangle\), called \(ilink\), is different from clink \(\langle c_3, d_1 \rangle\). The potential assigned to an ilink is different from the one assigned to a clink. There are four types of gate MFs in total that have been developed. One of them is assigned to a particular gate depending on the gate type and the level. More details can be found in [21].

The experimental result shows that comparing to inference by CPTs, inference
by MF of NAT models saves runtime and reduces the space. As $n$ grows from 5 to 11, inference with CPTs grows by 52 times of space and 193 times of runtime, while inference by MF of NAT models grows 1.1 times in space and takes about the same time. In particular, when $n = 11$, inference by MF of NAT uses less than 2% of space and is 396 times faster than inference by CPTs. The result shows that inference with NAT models significantly improves the space complexity and is time efficient.
Chapter 3

Approximating General CPTs with NAT Modeling

As we introduced in the previous chapter, CPTs can be derived by NAT models with single-causal probabilities and causal interactions among causes. The number of single-causals is linear on the number of causes $n$, and causal interactions are modeled by a NAT. Since the sizes of CPTs are exponential on $n$, NAT models can reduce the space complexity significantly. Additionally, NAT models can increase the speed of inference [16, 21]. With these two reasons, modeling CPTs by NAT models can significantly reduce the space of memory. The approximation of binary CPTs with NAT models has been proposed [23]. The idea is to approximate a target CPT over binary variables by a NAT model. However, binary variables are limited to model complex problems in real world. Due to this reason, we develop techniques to extend the CPT approximation to multi-valued variables.

Section 3.1 analyzes the challenges, particularly the difference between the approximation over binary variables and that over multi-valued variables. In Section 3.2, the distance measure of accuracy is discussed. Section 3.3 studies the method to extend the CPT approximation to multi-valued variables. In Section 3.4, the approximation of CPTs with persistent leaky causes is investigated.
3.1 Challenges

In this section, we analyze the challenges of extending the approximation to multi-valued variables. As we mentioned in Section 2.5, approximating binary CPTs has four main steps listed as follows.

1. Identify the PCI pattern from target CPT $P_T$.
2. Use the PCI pattern to retrieve a set of candidate NATs from the PST.
3. Search for single-causals for each candidate NAT.
4. Select the NAT model with the minimum error from $P_T$.

3.1.1 Challenges of Multi-valued CPT Approximation

We analyze the challenge of using NAT models to approximate multi-valued CPTs, comparing to binary CPT approximation step by step. In step 1, the goal is to extract the PCI pattern out of $P_T$. The definition of causal interactions over multi-valued variables can be found in Subsection 2.4.6. A pair of causes $c_i$ and $c_j$ with active values $c^{v_i}_i$ and $c^{v_j}_j$ may be undermining or reinforcing, relative to an active value $e^k$ of the effect. The CPT of a NAT model is referred to as a NAT CPT. In a NAT CPT, the causal interaction between $c_i$ and $c_j$ is independent of the indexes $v_i$, $v_j$ and $k$. In an arbitrary CPT, this may not be the case. Therefore, we need to determine the causal interaction between $c_i$ and $c_j$ by integrating the above information. In binary case, each variable only has one active value, so it does not have the above limitation. For multi-valued variables, each possible combination of active values of a pair of causes and the effect should be considered.
In Section 2.4.6, we defined undermining and reinforcing causal interactions over multi-valued variables. However, causes in an arbitrary target CPT may not satisfy the definition. We can relax the definition into a soft version as in the binary case (see Section 2.5). In doing so, we find that the soft definition in binary case has limitations. For instance, suppose that we have probabilities

\[ P(e \geq e^k \leftarrow c_i^+, c_j^+) \]  

\[ P(e \geq e^k \leftarrow c_i^+) \]  

\[ P(e \geq e^k \leftarrow c_j^+) \]  

If the double-causal \( P(e \geq e^k \leftarrow c_i^+, c_j^+) \) is between single-causals \( P(e \geq e^k \leftarrow c_i^+) \) and \( P(e \geq e^k \leftarrow c_j^+) \), following the similar idea of soft PCI definition in Section 2.5, we softly define \( pci(e^k; c_i, c_j) \) as undermining if \( P(e \geq e^k \leftarrow c_i^+, c_j^+) \) is close to \( \min\{P(e \geq e^k \leftarrow c_i^+), P(e \geq e^k \leftarrow c_j^+)\} \). The definition is similar for reinforcing. If the gap between these two probabilities is very small or the double-causal differs equally from the two single-causals, there is no good reason to assign the causal interaction as reinforcing or undermining. These issues will be resolved in Subsection 3.3.1.

After the PCI pattern, which is independent of the effect, is extracted, we use the PST to retrieve candidate NATs in the same way as step 2 for binary CPT approximation. Then, we search single-causals for the retrieved candidate NATs. In step 3 of binary CPT approximation, we find that all single-causals can be independently adjusted. However, to deal with multi-valued variables, the parameter search is more complicated. Assume that each variable has \( \eta > 1 \) active values. Each single-causal \( P(e \geq e^k \leftarrow c_i^+) \) involves one active value of a cause and one active value of the effect. For \( n \) causes, there are \( n\eta^2 \) single-causals in total. The \( \eta \) single-causals that
involve the same active value of the same cause are not independent. For example, with $\eta = 2$, $P(e \geq e^1 \leftarrow c_1^1)$ and $P(e \geq e^2 \leftarrow c_1^1)$ must sum to less than one. Therefore, the parameter search over multi-valued CPTs must be constrained. This will be presented in Subsection 3.3.2. After the parameter search for all candidate NATs is done, we select the NAT model with the smallest error from the target CPT as the result. This step is similar to step 4 for the binary case.

3.1.2 Challenges of Approximating CPTs with Persistent Leaky Causes

In Subsection 2.4.2, we discussed that leaky causes exist in CPTs from real world BNs. A target CPT may or may not have a leaky cause. A leaky cause may be persistent or non-persistent. We denote the leaky cause by $c_0$. A non-persistent leaky cause (NPLC) can be modeled in the same way as normal causes (see Subsection 2.4.2). A target CPT with a NPLC has $P(e|c_0, c_1, ..., c_n)$ fully specified where $P(e_0|c_0^0, c_1^0, ..., c_n^0) = 1$ and $P(e_k|c_0^0, c_1^0, ..., c_n^0) = 0$ ($k > 0$). Therefore, the approximation of a CPT with a NPLC can be performed in the same way as approximating CPTs without leaky causes.

A persistent leaky cause (PLC) is always active. We model $c_0 \in \{c_0^0, c_0^1\}$, where $c_0 = c_0^1$ always holds, which means that $c_0$ is always active. The target CPT with a PLC is in the form of $P(e|c_0^0, c_1, ..., c_n)$. In such a CPT, probabilities $P(e|c_0^0, c_1, ..., c_n)$ are not available and $P(e|c_0^1, c_1^0, ..., c_n^0) > 0$ since a PLC is always active.

PLC leads to some issues in CPT approximation. Probabilities $P(e|c_0^0, c_1, ..., c_n)$
are undefined so the target CPT with a PLC is in the form

\[ P'(e|c_1, \ldots, c_n) \ (n \text{ causes}) = P(e|c_0^1, c_1, \ldots, c_n) \ (n+1 \text{ causes}) \].

In the NAT CPT, if we do not explicitly model \( c_0 \), the interactions between \( c_0 \) and other causes cannot be expressed. In addition, the NAT CPT has \( P_M(e^k|c_1, \ldots, c_n^0) = 0 \ (k > 0) \). This leads to a systematic error. Therefore, we model the NAT CPT in the form \( P_M(e|c_0^1, c_1, \ldots, c_n) \). However, the new form leads to another challenge. The probabilities

\[ P(e \geq e^k \leftarrow c_i^+, c_j^+) \], \( P(e \geq e^k \leftarrow c_i^+) \) and \( P(e \geq e^k \leftarrow c_j^+) \),

\((i, j > 0)\) used to extract PCI patterns cannot be found in target CPTs with PLCs. The reason is that these probabilities requires \( c_0 = c_0^0 \), which is impossible given the existence of PLC. Therefore, we cannot use the similar way to extract the causal interactions as approximating CPTs without leaky causes. The main challenge of approximating CPTs with PLCs is how to extract PCI patterns out of the target CPTs. This will be studied in Section 3.4.

After the PCI pattern is extracted, we retrieve candidate NATs with the corresponding PST as approximating a CPT with a PLC. The difference is that the extracted PCI pattern encodes causal interactions among \( n + 1 \) causes (\( n \) normal causes and the PLC). The PST we use is changed accordingly. After candidate NATs are found, the parameter search for single-causals when leaky causes are absent can be applied. In the end, we select the NAT model with the smallest error from the target CPT as the approximation.
3.2 Distance Measure of Accuracy

In this section, we propose the functions to measure the distance between target CPT $P_T$ and $P_M$ which is the CPT generated from the approximated NAT model. There exists several ways to measure the distance of two probability distributions. Among all measurements, Euclidean distance (ED) and Kullback-Leibler divergence (KL) are commonly used [27]. ED is based on the absolute difference, so it is not sensitive to large relative difference in small probabilities. KL has no such limit. Therefore, we pick KL as our main function to measure the accuracy. Denote the target CPT as $P_T$ and the CPT obtained by NAT model $M$ as $P_M$. The KL function between the two CPTs is defined as

$$KL(P_T, P_M) = \sum_i P_T(i) \log \frac{P_T(i)}{P_M(i)},$$

where $i$ is the index of probabilities in $P_T$ and $P_M$. Gradient descent estimates single-causals of $M$ with the minimal $KL(P_T, P_M)$. In the experiment, we also compute ED, which is defined as

$$ED(P_T, P_M) = \sqrt{\frac{1}{N} \sum_{i=1}^{N} (P_T(i) - P_M(i))^2},$$

where $N$ is the number of parameters in $P_T$. We want to mention a special case in approximating CPTs with PLCs. Since we model the PLC as a normal binary cause, the CPT generated by the NAT model has $P_M(e|c_0, c_1, ..., c_n)$. However in target CPT, we only has $P_T(e|c_0, c_1, ..., c_n)$. Therefore, only $P_M(e|c_0^1, c_1, ..., c_n)$ will be used in calculating ED and KL.
3.3 Approximating Multi-valued CPTs

We first focus on the approximation of multi-valued CPTs without leaky causes. The work reported in this section has been published [20]. As we analyzed in Section 3.1, there are two main challenges to extend the approximation of multi-valued CPTs without leaky causes. The goal is to approximate a target CPT $P_T$ with the effect $e$ and a set $C$ of $n$ causes by a CPT $P_M$ generated by a NAT model $M$. In this section, we study how to extract PCI pattern from a target CPT in 3.3.1. Then the method to solve parameter search will be introduced in 3.3.2.

3.3.1 Extracting PCI Patterns from Multi-valued CPTs

This subsection solves the challenges of PCI pattern extraction. We start with extracting each value-pair interaction $vpi(e^k, c_i^+, c_j^+)$ ($i, j, k > 0$) from a target CPT. Then we determine the PCI bits $pci(e^k, c_i, c_j)$ based on value-pair interactions. Assume each variable has $\eta$ active values. We have $\eta$ candidate PCI patterns denoted by $pci(e^k)$ since a PCI pattern over multi-valued variables is defined relative to an active value $e^k$. Finally, we pick a best PCI pattern out of all $\eta$ candidates. With the PST structure, we can identify candidate NAT topologies based on the chosen partial PCI pattern.

The definition of a well-defined PCI bit between $c_i$ and $c_j$ relative to $e^k$ has been shown in Subsection 2.4.6. To simplify the function, we denote $P(e \geq e^k \leftarrow c_i^+)$ as $p$, $P(e \geq e^k \leftarrow c_j^+)$ as $q$ and $P(e \geq e^k \leftarrow c_i^+, c_j^+)$ as $o$. The function of a well-defined
value-pair interaction between $c_i^+$ and $c_j^+$ relative to $e^k$ is shown as follows.

$$vpi(e^k, c_i^+, c_j^+) = \begin{cases} 
  u : o < \min(p, q), \\
  r : o \geq \max(p, q). 
\end{cases}$$

The PCI bit $pci(e^k, c_i, c_j)$ is well defined if all value-pair interactions for all active values of $c_i$ and $c_j$ are defined as the same value ($u$ or $r$). However, the experimental result in Section 4.2 shows that in an arbitrary CPT, each condition hardly holds for all active values of $c_i$ and $c_j$ relative to any $e^k$. Very few bits are well defined which leads to a partial PCI pattern with nearly all bits undefined. As we said in Section 2.5, one missing PCI bit will double the number of candidate NATs. A partial PCI pattern with nearly all bits undefined corresponds to a large set of candidate NATs. This leads to costly subsequent search. Therefore, the best PCI pattern has the most defined PCI bits. We develop rules for flexible PCI extraction to define more PCI bits. The goal is to extract a partial PCI pattern which approximates causal interactions from a target CPT $P_T$. A PCI bit may be undermining ($u$), reinforcing ($r$) or undefined ($null$).

The function of a well-defined value-pair interaction has been given above. If $o \notin [\min(p, q), \max(p, q)]$. This function shows the situation that $vpi(e^k, c_i^+, c_j^+)$ is always well defined. When $o \in [\min(p, q), \max(p, q)]$, the rules to define a value-pair interaction are shown as follows.

The limit of soft PCI identification has been analyzed in Section 3.1. When $o \in [\min(p, q), \max(p, q)]$, we say that $vpi(e^k, c_i^+, c_j^+)$ is undefined if the distance $|p - q|$ is too small. We set up a possible value of threshold $\tau_0 \in (0, 1)$, say, $\tau_0 = 0.2$. 


This means when
\[ o \in [\min(p, q), \max(p, q)] \text{ and } |p - q| \leq \tau_0, \; vpi(e^k, c_i^+, c_j^+) = \text{null}. \]

This rule is called *tight enclosure*. We consider that under tight enclosure, both \(u\) and \(r\) may approximate the value-pair interaction between \(c_i^+\) and \(c_j^+\). Hence, both reinforcing and undermining are included as the candidate, which means that the interaction is undefined.

On the other hand, when \(o \in [\min(p, q), \max(p, q)]\) and \(|p - q| > \tau_0\), we call it *loose enclosure*. We first compute the ratio \(R = \frac{o - 0.5(p + q)}{|p - q|}\) to determine how close \(o\) is from \(p\) or \(q\). \(R \in [-0.5, 0.5]\) and it reaches the bound when \(o\) equals \(p\) or \(q\). When \(R < 0\), it is closer to \(\min(p, q)\). When \(R > 0\), it is closer to \(\max(p, q)\). When \(R = 0\), \(o\) has an equal distance from \(p\) and \(q\), which means that \(o\) is in the middle of \(p\) and \(q\). If \(o\) is in the middle, or in a small range of middle area, the interaction cannot be defined. Therefore, we set up a possible value of threshold \(\tau_1 \in (0, 0.5)\), say, \(\tau_1 = 0.4\), and obtain the softly-defined interaction as follows called *sided loose enclosure*. When \(o \in [\min(p, q), \max(p, q)]\) and \(|p - q| > \tau_0\),

\[
vpi(e^k, c_i^+, c_j^+) = \begin{cases} 
  u : & R < -\tau_1, \\
  \text{null} : & -\tau_1 \leq R \leq \tau_1, \\
  r : & R > \tau_1.
\end{cases}
\]

To intuitively summarize the value-pair interaction identification, we explain *tight enclosure* rule as shown in Figure 3.1 (a), and Figure 3.1 (b) illustrates *sided loose enclosure* rule. We see that in Figure 3.1 (b), the interaction is defined as \(u\) no matter it is well-defined or softly-defined, and so is for \(r\).
Several examples of defining \( vpi(e, c_i^+, c_j^+) \) with different rules are given as follows. We are given \( \tau_0 = 0.2, \tau_1 = 0.4 \). In the first example, we have

\[
P(e \geq e^1 \leftarrow c_1^1, c_3^1) = 0.7748, \quad P(e \geq e^1 \leftarrow c_1^2) = 0.7834, \quad P(e \geq e^1 \leftarrow c_1^3) = 0.7823.
\]

We have \( p = 0.7834, q = 0.7823, o = 0.7784 \). Since \( o < \min(p, q) \), \( vpi(e^1, c_1^1, c_3^1) = u \) is well defined.

In the second example, we have

\[
P(e \geq e^1 \leftarrow c_1^2, c_3^1) = 0.6702, \quad P(e \geq e^1 \leftarrow c_1^3) = 0.6527, \quad P(e \geq e^1 \leftarrow c_3^1) = 0.7823.
\]

We have \( p = 0.6527, q = 0.7823, o = 0.6702 \). Since \( q < o < p \), we first calculate \( |p - q| = 0.1296 \leq \tau_0 \). Thus, the tight closure rule is applied and we obtain \( vpi(e^1, c_1^2, c_3^1) = \text{null} \).
In the third example, we have

\[ P(e \geq e^1 \leftarrow c_1^2, c_3^2) = 0.6536, \quad P(e \geq e^1 \leftarrow c_1^3) = 0.6527, \quad P(e \geq e^1 \leftarrow c_3^3) = 0.8675. \]

We have \( p = 0.6527, q = 0.8675, o = 0.6536 \). Since \( q < o < p \), we first calculate \( |p - q| = 0.2148 > \tau_0 \). Then, we calculate the ratio \( R = \frac{o - 0.5(p + q)}{|p - q|} = -0.4958. \)

We find that \( R < -\tau_1 \), so \( vpi(e^1, c_1^2, c_3^3) = u \) is softly defined.

Given \( e^k \), above rules determine the value-pair interaction \( vpi(e^k, c_i^+, c_j^+) \) for a pair of values \( c_i^+ \) and \( c_j^+ \). If \( c_i \) has \( m_i \) active values and \( c_j \) has \( m_j \) active values, there are \( m_i m_j \) pairs of active values for \( c_i \) and \( c_j \). The value-pair interaction of each pair may be different. We determine the PCI bit \( pci(e^k, c_i, c_j) \) based on the majority of \( m_i m_j \) value-pair interactions. We use \( M_{vpi} \) to denote the number of value-pair interactions \( (m_i m_j) \) between \( c_i \) and \( c_j \), \( M_u \) to denote the number of undermining interactions where \( vpi(e^k, c_i^+, c_j^+) = u \), and \( M_r \) to denote the number of reinforcing interactions where \( vpi(e^k, c_i^+, c_j^+) = r \). We set up a possible value of threshold \( \tau_2 \in [0.5, 1) \), say, \( \tau_2 = 0.5 \), and have

\[
pci(e^k, c_i, c_j) = \begin{cases} 
  u & : M_u > \tau_2 M_{vpi}, \\
  r & : M_r > \tau_2 M_{vpi}, \\
  \text{null} & : \text{otherwise.}
\end{cases}
\]

When \( M_u \) is above the threshold, undermining is more reasonable to define the PCI bit, so we define \( pci(e^k, c_i, c_j) = u \). The same holds for reinforcing. There are two situations that lead to \( pci(e^k, c_i, c_j) = \text{null} \). The first one is that \( M_u \) and \( M_r \) both are less than the threshold. The other is \( M_u = M_r = 0.5 M_{vpi} \) which may appear when \( M_{vpi} \) is an even number. In any of this situation, there is no clear
evidence for \( u \) or \( r \) to dominate. Therefore, no decision can be made and the PCI bit is undefined. We leave it to the subsequent search to numerically decide the interaction. For example, the domain size of \( c_1 \) is \( D_1 = 3 \) and the domain size of \( c_3 \) is \( D_3 = 4 \), and set \( \tau_2 = 0.5 \). This means we have \( M_{vpi} = 2 \times 3 = 6 \). We get the set \( \{ vpi(e^1, c_1^+, c_3^+) \} = \{ u, u, n, u, r \} \). Since \( M_u = 4 > \tau_2 M_{vpi} \), we define \( pci(e^1, c_1, c_3) = u \).

After extracting the PCI bit \( pci(e^k; c_i, c_j) \) for each cause pair \( (c_i, c_j) \), a candidate PCI pattern \( pci(e^k) \) is defined. If \( e \) has \( \eta \) active values, we can have \( \eta \) candidates. Assume that \( e \) has \( n \) causes. The candidate PCI pattern relative to \( e^k \) is \( pci(e^k) = \{pci(e^k, c_i, c_j)\} \) for all pairs of causes \( (c_i, c_j) \). We calculate the number \( N_k \) of defined PCI bits in \( pci(e^k) \) such that \( pci(e^k, c_i, c_j) \neq null \). Then, we select \( pci(e^k) \) with the largest \( N_k \). In particular, we denote the largest \( N_k \) as \( N_x \), and \( pci(e^x) \) with the largest \( N_k \) as \( pci(e^x) \). We set up a possible value of threshold \( \tau_3 \in (0.5, 1) \), say, \( \tau_3 = 0.8 \). Finally, we select \( N_x \) as the partial PCI pattern if

\[
N_x > \tau_3 \left( \frac{n}{2} \right).
\]

We set up \( \tau_3 \) to make sure that enough PCI bits are extracted, which will save the processing time of subsequent search. If \( N_x \leq \tau_3 \left( \frac{n}{2} \right) \), this partial PCI pattern is denied. Then, we redo the above procedures by relaxing thresholds \( \tau_0 \) to \( \tau_3 \) until at least one PCI pattern is selected. For example, the domain size of the effect \( e \) is \( D_e = 4 \) and the number of causes \( n = 4 \). We get 3 candidate PCI patterns based on 3 active value of \( e \), each of which has \( \left( \frac{n}{2} \right) = 6 \) PCI bits. For simplicity, we use \( - \) to represent that the PCI bits is undefined. We set \( \tau_3 \) as 0.8, so \( \tau_3 \times 6 = 4.8 \). This
means that there must be at least 5 defined PCI bits if the PCI pattern is selected for subsequent search. Given the candidate PCI patterns \( pci(e^1) = \{u, u, \_\_\_r, \_\_\_\_\_\} \), \( pci(e^2) = \{u, u, \_\_\_, u, \_\_\_\} \) and \( pci(e^3) = \{u, u, \_\_\_, r, u, u\} \). We have \( N_1 = 3 \), \( N_2 = 3 \) and \( N_3 = 5 \). Therefore, only \( pci(e^3) \) can be selected.

3.3.2 Parameter Search with Constrained Gradient Descent

After we extract a partial PCI pattern, combined with PST in old steps, the corresponding candidate NATs can be determined [24]. Since the parameter search in binary case has limitations, for each obtained candidate NAT, we search for single-causals by adding constraints to gradient descent.

A NAT model \( M \) contains a NAT topology and a set of single-causal probabilities. We measure the accuracy of approximation by KL divergence. The functions are already been listed in Section 3.2. During descent, the point moving on the \( n \)-dimension surface is a vector of \( n \) single-causals. In binary CPT approximation, \( n \) single-causals can be independently obtained. For multi-valued CPTs, assume that the effect \( e \) has the domain size \( |D_e| = \eta + 1 \) (\( \eta \) active values and 1 inactive value) and each cause \( c_i \) of \( n \) causes has the domain size \( |D_i| = m + 1 \). The point of descent is a vector with \( \eta nm \) parameters. Each parameter corresponds to a single-causal \( P(e^+ \leftarrow c_i^+) \). We divide all single-causals into \( nm \) groups for each value of causes. Each group has \( \eta \) single-causals. Unlike binary CPT approximation, \( \eta \) parameters in the same group are not independent. For example, consider a CPT with \( |D_e| = 3 \),
each $|D_i| = 3$ and $n = 3$. According to the sum rule, we have

$$P(e^1 \leftarrow c_1^+) + P(e^2 \leftarrow c_1^+) = 1 - P(e^0 \leftarrow c_1^+).$$

Since single-causals are related to an active value of $e$ and an active value of $c_i$, during descent, single-causals that we search are $P(e^+ \leftarrow c_i^+)$. In this example, we calculate $P(e^1 \leftarrow c_1^+)$ and $P(e^2 \leftarrow c_1^+)$. The sum of these two single-causals is less than 1. Hence, these two single-causals are dependent. Therefore, the following constraints must be satisfied during descent.

For each single-causal, $P(e^+ \leftarrow c_i^+) > 0$ holds because of the range of probability. However, it cannot reach 0, otherwise $c_i^+$ is no longer a cause of $e^+$. Second, each parameter $P(e^+ \leftarrow c_i^+) < 1$. According to the range of probability, each parameter has the upper bound 1, but cannot reach the bound. If it reaches 1, it is a certain cause, which violates our assumption. We add another constraint that for each $c_i^+$,

$$\sum_{v=1}^{\eta} P(e^v \leftarrow c_i^+) < 1$$

must hold. If violated, the obtained parameters $P(e^1 \leftarrow c_1^+), ..., P(e^\eta \leftarrow c_1^+)$ will violate the sum rule, so they will be invalid. There are $nm$ such constraints (one for each group), and each of them controls $\eta$ parameters. Therefore, we extend gradient descent to ensure that all constraints hold.

To ensure the result of gradient descent is reasonable, for each candidate NAT, we run the descent for multiple rounds, say, 10 rounds. At the beginning of each round of descent, we initialize each group of $\eta$ single-causal under the same constraint as follows. Given a small positive real number $\delta$, say, $\delta = 0.01$, we generate $\eta + 1$ random
numbers for $P(e^0 \leftarrow c^+_1), ..., P(e^n \leftarrow c^+_1)$ in the range $[\delta, 1-\delta]$ to avoid reaching 0 or 1. Let $S$ be the sum of these $\eta+1$ numbers. Then we arbitrarily drop one number which we treat as $P(e^0 \leftarrow c^+_1)$. Given a positive real $\gamma$ close to 1, say, $\gamma = 0.99$, we multiply the remaining $\eta$ numbers by $\gamma S$ and assign resultant numbers as initial single-causals. Therefore, we have the following constraint

$$ \sum_{v=1}^{\eta} P(e^v \leftarrow c^+_1) \leq \gamma. $$

For example, when $\eta = 3$, $P(e^1 \leftarrow c^+_1)$, $P(e^2 \leftarrow c^+_1)$ and $P(e^3 \leftarrow c^+_1)$ are to be initialized. We randomly generate $\eta+1 = 4$ numbers which are in the range $[\delta, 1-\delta]$. Given $\delta = 0.01$, we randomly generate 0.3, 0.4, 0.6 and 0.7 in the range $[0.01, 0.99]$, and their sum $S = 2.0$. Then we arbitrarily drop one number 0.4, which we consider as $P(e^0 \leftarrow c^+_1)$. Given $\gamma = 0.99$, we multiply the remaining numbers 0.3, 0.6 and 0.7 by $\frac{\gamma}{S} = 0.495$, and obtain 0.1485, 0.2970 and 0.3465. Finally, we assign them as initial single-causals, say,

$$ P(e^1 \leftarrow c^+_1) = 0.1485, \ P(e^2 \leftarrow c^+_1) = 0.2970, \ P(e^3 \leftarrow c^+_1) = 0.3465. $$

The multiplication guarantees that the sum of initialized single-causals is less than 1.

It takes steps to finish one round of gradient descent. Each step of gradient descent updates all $\eta mn$ parameters one at a time. To ensure that all constraints still hold, we constrain the process of descent as follows. We use $P(e^k \leftarrow c^+_i)$ to represent the probability before the update, $P'(e^k \leftarrow c^+_i)$ to represent the probability after the update, and $S'$ to represent the sum of $P'(e^k \leftarrow c^+_i)$. After each $P'(e^k \leftarrow c^+_i)$ is updated, check if $P'(e^k \leftarrow c^+_i) \geq \delta$ holds. If not, it means that $P'(e^k \leftarrow c^+_i)$ is
below the lower bound. We set \( P'(e^k \leftarrow c_i^+) = \delta \), and stop \( P'(e^k \leftarrow c_i^+) \) for further descent. Otherwise, check whether \( S' = \sum_{v=1}^{\eta} P'(e^v \leftarrow c_i^+) \leq \gamma \) still holds. If not, we set \( P'(e^k \leftarrow c_i^+) = P'(e^k \leftarrow c_i^+) + \gamma - S' \) and stop \( P'(e^k \leftarrow c_i^+) \) for further descent. If it satisfies both conditions, assign \( P'(e^k \leftarrow c_i^+) \) by the updated value and continue descent. These tests end until the descent ends. For example, in one step of descent, \( P'(e^1 \leftarrow c_2^1) = 0.0975 \) is updated. The test for the lower bound constraint shows \( P'(e^1 \leftarrow c_2^1) = 0.0975 < \delta \). Thus, we set \( P'(e^1 \leftarrow c_2^1) = 0.01 \) and stop \( P'(e^1 \leftarrow c_2^1) \) for further descent. As another example, we have

\[
P'(e^1 \leftarrow c_2^1) = 0.1675, \quad P'(e^2 \leftarrow c_2^1) = 0.3320,
\]

\[
P'(e^3 \leftarrow c_2^1) = 0.4831 \quad \text{and} \quad P'(e^3 \leftarrow c_2^1) = 0.5384,
\]

where \( P'(e^3 \leftarrow c_2^1) \) is updated. We have \( S = 0.1675 + 0.3320 + 0.4831 = 0.9826 < \gamma \), which satisfies the sum constraint before the update. After the update, the test shows that \( S' = 0.1675 + 0.3320 + 0.5384 = 1.0379 > \gamma \), which violates the constraint. Therefore, we set \( P'(e^3 \leftarrow c_2^1) = 0.5384 + 0.99 - 1.0379 = 0.4905 \) and stop \( P'(e^3 \leftarrow c_2^1) \) for further descent.

In summary, the initiation step ensures that descent starts with valid parameters. The tests for each step of descent ensure that each parameter remains valid after it is updated. Therefore, the constrained gradient descent ensures valid single-causals.

After gradient descent finishes for all candidate NATs, corresponding candidate NAT models can be determined. The final step of multi-valued CPT approximation is to select the resultant NAT model \( M \) with the smallest error from the target CPT. This step is the same as binary CPT approximation (see Section 2.5).
3.4 Approximation of CPTs with Persistent Leaky Causes

In real world CPTs, leaky causes may exist which can be persistent or non-persistent. Section 3.3 extends the approximation of multi-valued CPTs without leaky causes. A NPLC can be modeled the same way as a normal cause. Therefore, the method presented in Section 3.3 can be applied to approximate CPTs with NPLCs (see Section 3.1). The main challenge to approximate a target CPT with a PLC into a NAT model is to extract the PCI pattern. We create a novel idea to extract PCI patterns by using probabilities

\[ P(e \geq e^k \leftarrow c_{0}^{1}), \quad P(e \geq e^k \leftarrow c_{0}^{1}, c_{1}^{+}), \quad P(e \geq e^k \leftarrow c_{0}^{1}, c_{1}^{+}, c_{j}^{+}) \]

where the PLC is modeled by \( c_{0} \) and \( c_{0} = c_{0}^{1} \) always holds. After the PCI pattern is extracted, PSTs can be used to retrieve candidate NATs. The difference is that if the target CPT has \( n \) causes, we search the PST for \( n + 1 \) since we explicitly model the PLC. As we analyzed in Section 3.1, the subsequent parameter search follows the same idea as approximating CPTs without leaky causes. Therefore, we focus on how to extract the PCI pattern in this section. The work reported in this section has been published [19].

3.4.1 Extraction of Causal Interaction Bits by NAT Differentiation

The main difficulty to extract the PCI pattern is that probabilities which are used to define causal interactions over multi-valued variables (see Subsection 2.4.6)

\[ P(e \geq e^k \leftarrow c_{i}^{+}, c_{j}^{+}), \quad P(e \geq e^k \leftarrow c_{i}^{+}), \quad P(e \geq e^k \leftarrow c_{j}^{+}) \]
(i, j > 0), are not available in the target CPT. Instead of extracting the causal interaction \( pci(e^k, c_i, c_j) \) relative to \( e^k \) between \( c_i \) and \( c_j \), we develop a way to extract the causal interactions among \( c_0, c_i \) and \( c_j \). The available probabilities in the target CPT that involve \( c_0, c_i \) and \( c_j \) are

\[
P(e \geq e^k \leftarrow c_0^1), P(e \geq e^k \leftarrow c_0^1, c_i^+), P(e \geq e^k \leftarrow c_0^1, c_j^+) \text{ and } P(e \geq e^k \leftarrow c_0^1, c_i^+, c_j^+).
\]

To determine causal interactions among \( c_0, c_i \) and \( c_j \), first we extract value-pair interactions among \( c_0^1, c_i^+ \) and \( c_j^+ \).

We start with illustrating possible NAT topologies with the value tuple \( (e^k, c_0^1, c_i^+, c_j^+) \).

Figure 3.2: (Sub)NATs over \( c_0, c_i \) and \( c_j \)

Figure 3.2 enumerates all possible NATs with the three cause values. Each NAT corresponds to a set of value-pair interactions. Value-pair interactions are summarized
in Table 3.1. If we can identify which NAT in Figure 3.2 describes the potential value-
pair interactions among $c_1^1$, $c_i^+$ and $c_j^+$ relative to $e^k$, we can obtain the corresponding value-pair interactions from Table 3.1. For example, according to $T_b$ in Figure 3.2, we can observe that, relative to $e^k$, $c_i^+$ and $c_j^+$ are reinforcing each other, and they both undermine $c_1^1$. Therefore, we have $vpi(e^k, c_1^1) = u$, $vpi(e^k, c_i^+) = u$ and $vpi(e^k, c_i^+, c_j^+) = r$,

which matches the $T_b$ column in Table 3.1. Other NATs and corresponding interactions can be identified in the similar way.

We compare the four available probabilities in the target CPT and analyze $inom{4}{2} = 6$ comparisons with the NATs in Figure 3.2. We summarize the comparisons in Table 3.2, where $+$ represents that the result is greater than 0, $-$ represents that the result is less than 0, and $+/-$ represents that the result might be greater or less than 0.

For simplicity, in comparisons, we rewrite the probabilities $P(e \geq e^k \leftarrow c_0^1)$, $P(e \geq e^k \leftarrow c_0^1, c_i^+)$, $P(e \geq e^k \leftarrow c_0^1, c_j^+)$ and $P(e \geq e^k \leftarrow c_0^1, c_i^+, c_j^+)$ as $P(0^+)$, $P(0i^+)$, $P(0j^+)$ and $P(0ij^+)$, respectively. We analyze all 6 comparisons into two parts.
Table 3.2: Pairwise causal probability comparison by NAT models

<table>
<thead>
<tr>
<th></th>
<th>(T_a)</th>
<th>(T_b)</th>
<th>(T_e)</th>
<th>(T_f)</th>
<th>(T_d)</th>
<th>(T_g)</th>
<th>(T_c)</th>
<th>(T_h)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(P(e \geq e^k \leftarrow c_0^i, c_i^+, c_j^+))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-P(e \geq e^k \leftarrow c_0^i, c_i^+))</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>(P(e \geq e^k \leftarrow c_0^i, c_i^+, c_j^+))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-P(e \geq e^k \leftarrow c_0^i, c_j^+))</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>(P(e \geq e^k \leftarrow c_0^i, c_i^+))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-P(e \geq e^k \leftarrow c_0^i))</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>(P(e \geq e^k \leftarrow c_0^i, c_i^+))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-P(e \geq e^k \leftarrow c_0^i))</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
<tr>
<td>(P(e \geq e^k \leftarrow c_0^i, c_j^+))</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>(-P(e \geq e^k \leftarrow c_0^i))</td>
<td>+</td>
<td>-</td>
<td>-</td>
<td>+</td>
<td>+</td>
<td>-</td>
<td>+</td>
<td>+</td>
</tr>
</tbody>
</table>

In the first part, we analyze the first three comparisons. For \(T_a\), any disjoint subsets of \(\{c_0, c_i, c_j\}\) reinforce each other, so we have

\[P(0ij^+) > P(0i^+), P(0ij^+) > P(0j^+) \text{ and } P(0ij^+) > P(0^+).\]

If the above conditions hold, we can define the value-pair interaction as \(T_a\) column in Table 3.1.

For \(T_e\), any disjoint subsets of \(\{c_0, c_i, c_j\}\) undermine each other, so we have

\[P(0ij^+) < P(0i^+), P(0ij^+) < P(0j^+) \text{ and } P(0ij^+) < P(0^+).\]

If the above conditions hold, we can define the value-pair interaction as \(T_e\) column in Table 3.1.

For \(T_b\), \(P(0ij^+)\) results from the interaction between \(c_0\) and the group \(\{c_i, c_j\}\), and \(P(0i^+)\) results from the interaction between \(c_0\) and \(c_i\). We find that \(c_j\) reinforces \(c_i\) in \(T_b\), which means \(P(0ij^+) > P(0i^+)\). Since \(c_i\) and \(c_j\) are symmetric, we derive
\( P(0ij^+) > P(0j^+) \). In addition, \( c_0 \) undermines the group \{\( c_i, c_j \)\}, so \( P(0ij^+) < P(0^+) \) can be obtained. Therefore, with resultant comparisons, \( T_b \) can be identified. \( T_f \) has the same NAT topology as \( T_b \) but the type of leaf gate is different. Thus, for \( T_f \), we derive
\[
P(0ij^+) < P(0i^+), \quad P(0ij^+) < P(0j^+), \quad \text{and} \quad P(0ij^+) > P(0^+).
\]

For \( T_c \), \( P(0ij^+) \) results from the interaction between \( c_j \) and the group \{\( c_0, c_i \)\}. We find that \( c_j \) reinforces \{\( c_0, c_i \)\}, which means \( P(0ij^+) > P(0i^+) \). In addition, \( c_i \) undermines \( c_0 \), and we obtain \( P(0ij^+) < P(0j^+) \). To compare \( P(0ij^+) \) and \( P(0^+) \), we have
\[
P(0ij^+) - P(0^+) = (1 - P(0i^-)) - (1 - P(0^+))
= P(0^-) - P(0i^-) = (1 - P(0^+)) - (1 - P(0^+) \cdot P(j^+)) \cdot (1 - P(i^+))
= 1 - P(0^+) - (1 - P(i^+)) + P(0^+) \cdot P(j^+) \cdot (1 - P(i^+))
= P(i^+) - P(0^+) \cdot (1 - (P(j^+) \cdot (1 - P(i^+)))).
\]

If \( P(i^+) \) is close to 1, the above equation is about \( 1 - P(0^+) > 0 \). If \( P(i^+) \) is close to 0, the above equation is about \( -P(0^+)(1 - (P(j^+)) < 0 \). Hence, the comparison between \( P(0ij^+) \) and \( P(0^+) \) is not deterministic for \( T_c \). Since \( c_i \) and \( c_j \) are symmetric, the same holds for \( T_d \). \( T_g \) has the same NAT topology as \( T_c \), but the type of leaf gate is different. Thus, they have opposite symbols in comparisons. The same holds for \( T_d \) and \( T_h \).

In the second part, we analyze the last three comparisons. For \( T_a \), we can only observe that \( c_i \) and \( c_j \) are reinforcing each other, but we are not sure which probability is greater between \( P(0i^+) \) and \( P(0j^+) \). Thus, the comparison is not deterministic.
Moreover, $c_i$ and $c_j$ both reinforce $c_0$, so we have

$$P(0i^+) > P(0^+) \text{ and } P(0j^+) > P(0^+).$$

For $T_e$, with the similar idea of $T_a$, we obtain that the comparison between $P(0i^+)$ and $P(0j^+)$ is not deterministic, and have

$$P(0i^+) < P(0^+) \text{ and } P(0j^+) < P(0^+).$$

For $T_b$, the comparison of $P(0i^+)$ and $P(0j^+)$ is not deterministic due to the same reason as $T_a$. In $T_b$, $c_i$ and $c_j$ both undermine $c_0$, so we have

$$P(0i^+) < P(0^+) \text{ and } P(0j^+) < P(0^+).$$

For $T_f$, with the similar idea of $T_b$, we obtain that the comparison between $P(0i^+)$ and $P(0j^+)$ is not deterministic, and have

$$P(0i^+) > P(0^+) \text{ and } P(0j^+) > P(0^+).$$

For $T_c$, we find that $c_0$ undermines $c_i$ and reinforces $c_j$, so we have $P(0j^+) > P(0^+) > P(0i^+)$. Therefore, we have the three comparisons

$$P(0i^+) < P(0j^+), \ P(0i^+) < P(0^+) \text{ and } P(0j^+) > P(0^+).$$

For $T_h$, $c_0$ reinforces $c_j$ and undermines $c_i$, and have $P(0i^+) < P(0^+) < P(0j^+)$. Therefore, we have the three comparisons

$$P(0i^+) < P(0j^+), \ P(0i^+) < P(0^+) \text{ and } P(0j^+) > P(0^+).$$
For $T_d$, we find that $c_0$ reinforces $c_i$ and undermines $c_j$, so we have $P(0i^+) > P(0^+) > P(0j^+)$. Therefore, we have the three comparisons

$$P(0i^+) > P(0j^+), \ P(0i^+) > P(0^+) \text{ and } P(0j^+) < P(0^+).$$

For $T_g$, $c_0$ undermines $c_j$ and reinforces $c_i$, and have $P(0j^+) < P(0^+) < P(0i^+)$. Therefore, we have the three comparisons

$$P(0i^+) > P(0j^+), \ P(0i^+) > P(0^+) \text{ and } P(0j^+) < P(0^+).$$

In conclusion, we can uniquely identify $T_a, T_b, T_e$ and $T_f$ in Figure 3.2 based on the first three comparisons. Hence, the value-pair interactions among $\{c_0^1, c_i^+, c_j^+\}$ can be defined from the corresponding columns in Table 3.1. Furthermore, we notice that all comparisons for $T_e$ and $T_h$ are identical. The same holds for $T_d$ and $T_g$. Therefore, $\{T_c, T_h\}$ can be identified as a group, and so can $\{T_d, T_g\}$. However, two NAT members in each group cannot be differentiated by the comparisons. Therefore, we will introduce a new idea to the differentiate the group members of the two NAT groups in the next subsection.

### 3.4.2 NIN-AND Tree Group Member Differentiation

The pairwise comparisons among 4 available probabilities allow us to uniquely identify 50% (4 out of 8) of the NATs with values $c_{0}^1, c_i^+, c_j^+$, where $i, j > 0$. As explained in the last subsection, $T_a, T_b, T_e$ and $T_f$ can be uniquely defined. Group $\{T_c, T_h\}$ and $\{T_d, T_g\}$ can also be defined. In Table 3.1, we compare the columns for $T_e$ and $T_h$, and find $vpi(e^k, c_0^1, c_i^+)$ and $vpi(e^k, c_0^1, c_j^+)$ are the same. This means
that these two value-pair interactions can always be defined. The same holds for $T_d$ and $T_g$. Therefore, we derive that $vpi(e^k, c^1_0, c^1_1)$ and $vpi(e^k, c^1_0, c^1_2)$ can always be defined and only $vpi(e^k, c^+_i, c^+_j)$ has 50% chance to be defined. In total, there are
\[ \binom{n+1}{2} = \frac{n(n+1)}{2} \]
value-pair interactions. For $vpi(e^k, c^+_i, c^+_j)$, there are $\binom{n}{2} = \frac{n(n-1)}{2}$ pairs that have 50% chance to be defined. In general, the number of value-pair interactions that can be defined is $\frac{n(n+1)}{2} - \frac{n(n-1)}{4} = \frac{n(n+3)}{4}$. In other words, the chance of a value-pair interaction being defined is $\frac{n+3}{2(n+1)}$. As $n$ grows from 2 to 6, for example, the chance that a value-pair interaction is defined is from 86% to 64%. The number of candidate NATs grows exponentially on the number of undefined PCI bits (see Section 2.5). If there are many PCI bits undefined, it will increase the number of candidate NATs which will raise the complexity of subsequent search. Therefore, further differentiation between $T_c$, $T_d$, $T_g$ and $T_h$ is desirable.

Consider the probability $P(e \geq e^k \leftarrow c^1_0, c^+_i)$. If $c_0$ and $c_i$ are undermining each other, we have
\[
P(e \geq e^k \leftarrow c^1_0, c^+_i) = P(e \geq e^k \leftarrow c^1_0) \times P(e \geq e^k \leftarrow c^+_i).
\]
The probability $P(e \geq e^k \leftarrow c^+_i)$ is unavailable in the target CPT, but we can estimate it from the available probabilities $P(e \geq e^k \leftarrow c^1_0)$ and $P(e \geq e^k \leftarrow c^+_i)$ by
\[
P(e \geq e^k \leftarrow c^+_i) = \frac{P(e \geq e^k \leftarrow c^1_0, c^+_i)}{P(e \geq e^k \leftarrow c^1_0)}.
\]
If $c_0$ and $c_i$ are reinforcing each other, we have
\[
P(e < e^k \leftarrow c^+_0, c^+_i) = P(e < e^k \leftarrow c^+_0) \times P(e < e^k \leftarrow c^+_i),
\]
and we can estimate
\[ P(e < e^k \leftarrow c_i^+) = \frac{P(e < e^k \leftarrow c_0^1, c_i^+)}{P(e < e^k \leftarrow c_0^1)}. \]

For both members of group \( \{T_c, T_h\} \), \( c_0 \) and \( c_i \) are undermining each other, and \( c_0 \) and \( c_j \) are reinforcing each other. We can estimate single-causals \( P(e \geq e^k \leftarrow c_i^+) \) and \( P(e \geq e^k \leftarrow c_j^+) \) according to the above equations and available probabilities

\[ P(e \geq e^k \leftarrow c_0^1, c_i^+), \ P(e \geq e^k \leftarrow c_0^1, c_j^+) \text{ and } P(e \geq e^k \leftarrow c_0^1). \]

Then, we use the estimated \( P(e \geq e^k \leftarrow c_i^+) \) and \( P(e \geq e^k \leftarrow c_j^+) \) with the available probability \( P(e \geq e^k \leftarrow c_0^1) \) to calculate two multi-causals \( P_c(e \geq e^k \leftarrow c_0^1, c_i^+, c_j^+) \) and \( P_h(e \geq e^k \leftarrow c_0^1, c_i^+, c_j^+) \) according to the causal interactions in \( T_c \) and \( T_h \), respectively. Compare the two multi-causals to \( P(e \geq e^k \leftarrow c_0^1, c_i^+, c_j^+) \) from the target CPT, the NAT will be selected if its multi-causal is closer to the one from the target CPT.

For group \( \{T_d, T_g\} \), \( c_0 \) and \( c_i \) are reinforcing each other in both NATs, and \( c_0 \) and \( c_j \) are undermining each other. The similar approach is applied to differentiate the two group members.

Although this method often resolves the problem of extracting value-pair interaction, it does not always do so. For example, in a target CPT, we observe that

\[ P(e \geq e^k \leftarrow c_0^1, c_i^+, c_j^+) = 0.5438, \ P(e \geq e^k \leftarrow c_0^1, c_i^+) = 0.2620, \]

\[ P(e \geq e^k \leftarrow c_0^1, c_j^+) = 0.6316, \text{ and } P(e \geq e^k \leftarrow c_0^1) = 0.8317. \]

From the observation, we have \( P(0ij^+) > P(0i^+) \) and \( P(0ij^+) < P(0j^+) \), which matches (+, −) of the first two comparisons in Table 3.2. Thus, we pick the group
\{T_c, T_h\}$. However, we notice that $P(0j^+) < P(0^+)$ in this example. It is different from the last comparison of $T_c$ and $T_h$. Therefore, estimating $P(e \geq e^k \leftarrow c_j^+)$ by reinforcing is not applicable.

We recognize that in Table 3.2, the last two comparisons depend on the causal interactions of $(c_0, c_i)$ and $(c_0, c_i)$, respectively. In the last example, the corresponding comparisons are $(-, -)$, which are different from the ones for $T_c$ and $T_h$. Therefore, we conclude that the estimation method works only when the last two comparisons match the corresponding group members. This also reminds us that the last two comparisons can be used for identifying NATs when the comparisons do not match.

With this idea, we use the the last two comparisons $(-, -)$ to obtain a new NAT group $\{T_b, T_e\}$. Since the first two comparisons in the example $(+, -)$ are different from $T_b$ $(+, +)$ or $T_e$ $(-, -)$, we need to differentiate the group member between $T_b$ and $T_e$. To reach this goal, we estimate single-causals $P(e \geq e^k \leftarrow c_i^+)$ and $P(e \geq e^k \leftarrow c_j^+)$ by the estimation method. Since the last two comparisons match, we can use the undermining equation to estimate two single-causals. After we estimate two single-causals, we calculate the multi-causals $P_b(e \geq e^k \leftarrow c_0^1, c_i^+, c_j^+)$ and $P_e(e \geq e^k \leftarrow c_0^1, c_i^+, c_j^+)$ for $T_b$ and $T_e$, respectively. Then, we use the two multi-causals to compare to $P(e \geq e^k \leftarrow c_0^1, c_i^+, c_j^+)$ from the target CPT. The NAT with a smaller error will be selected.

As another example, we observe a target CPT where

\[
P(e \geq e^k \leftarrow c_0^1, c_i^+, c_j^+) = 0.9527, \quad P(e \geq e^k \leftarrow c_0^1, c_i^+) = 0.9580, \]

\[
P(e \geq e^k \leftarrow c_0^1, c_j^+) = 0.9343, \quad \text{and} \quad P(e \geq e^k \leftarrow c_0^1) = 0.7412.
\]
Applying the first two comparisons (−, +) in Table 3.2, the group \{T_d, T_g\} is selected. However, the last two comparisons (+, +) do not match Table 3.2, which makes the estimation of \(P(e \geq e^k \leftarrow c_0^1, c_j^+\) not applicable. Thus, we differentiate the member in the group \{T_a, T_f\} instead.

From the above analysis, we summarize the procedure of obtaining the value-pair interactions among the tuple \((e^k, c_0^1, c_i^+, c_j^+)\) in Figure 3.3. In a target CPT, first we locate the four available probabilities \(P(e \geq e^k \leftarrow c_0^1), P(e \geq e^k \leftarrow c_0^1, c_i^+)\) and \(P(e \geq e^k \leftarrow c_0^1, c_i^+, c_j^+)\). We start with doing the first three comparisons in Table 3.2 to identify the NAT. If it succeeds in uniquely identifying NAT \(T_a, T_b, T_e\) or \(T_f\) in Figure 3.2, we can obtain the corresponding value-pair interactions from Table 3.1. If it identifies a NAT group \{T_d, T_g\} or \{T_c, T_h\}, we check whether the last two comparisons match the ones in the identified group. If the last two comparisons match, we apply the single-causal estimation to estimate the single-causals \(P(e \geq e^k \leftarrow c_i^+)\) and \(P(e \geq e^k \leftarrow c_j^+)\). Then we calculate the multi-causal \(P'(e \geq e^k \leftarrow c_0^1, c_i^+, c_j^+)\) for each identified group member. Finally, we compare the calculated multi-causal with the one from target CPT, and select the one with the smaller error as the resultant NAT. If the last two comparison do not match, we use the comparisons to identify a NAT group of \{T_a, T_f\} or \{T_b, T_e\}. Then we apply the single-causal estimation, calculate the multi-causals and differentiate the group members. After we identify the NAT, we can use Table 3.1 to extract the value-pair interactions.
For example, in a CPT with a PLC, we observe

\[ P(e \geq e^1 \leftarrow c_0^1, c_1^1, c_2^1) = 0.5582, \ P(e \geq e^1 \leftarrow c_0^1, c_1^1) = 0.5012, \]

\[ P(e \geq e^1 \leftarrow c_0^1, c_2^1) = 0.6743, \ \text{and} \ P(e \geq e^1 \leftarrow c_0^1) = 0.4412. \]

We start with applying the first three comparisons and have \((+, -, +)\). According to Table 3.2, we select the group \(\{T_c, T_h\}\). Then we check the last two comparisons and
have (+, +). The group \( \{T_a, T_f\} \) is now selected instead. According to Table 3.1, we have \( pci(e^1 \leftarrow c^1_0, c^1_1) = r \) and \( pci(e^1 \leftarrow c^1_0, c^1_2) = r \). Thus, we apply estimation method and obtain single-causals

\[
P(e \geq e^1 \leftarrow c^1_1) = 0.1074 \text{ and } P(e \geq e^1 \leftarrow c^1_2) = 0.4171.
\]

We calculate multi-causals

\[
P_a(e \geq e^1 \leftarrow c^1_0, c^1_1, c^1_2) = 0.7092 \text{ and } P_f(e \geq e^1 \leftarrow c^1_0, c^1_1, c^1_2) = 0.4663
\]

based on the estimated single-causals. Then we compare the two multi-causals with the observed probability from target CPT, and obtain

\[
|P_a(e \geq e^1 \leftarrow c^1_0, c^1_1, c^1_2) - P(e \geq e^1 \leftarrow c^1_0, c^1_1, c^1_2)| = 0.1511,
\]

and

\[
|P_f(e \geq e^1 \leftarrow c^1_0, c^1_1, c^1_2) - P(e \geq e^1 \leftarrow c^1_0, c^1_1, c^1_2)| = 0.0920.
\]

Therefore, \( T_f \) is selected and from Table 3.1, we define value-pair interactions

\[
vpi(e^1 \leftarrow c^1_0, c^1_1) = r, \ vpi(e^1 \leftarrow c^1_0, c^1_2) = r \text{ and } vpi(e^1 \leftarrow c^1_1, c^1_2) = u.
\]

### 3.4.3 Effect-Dependent PCI Patterns

After we extract value-pair interactions \( vpi(e^k, c^+_i, c^+_j) \) for all active values of \( c_i \) and \( c_j \) (\( i, j \geq 0 \)) relative to \( e^k \), the *effect-dependent PCI bit* \( pci(e^k, c_i, c_j) \) can be defined with a majority vote over \( vpi(e^k, c^+_i, c^+_j) \).

Assume the number of active values of \( c_i \) and \( c_j \) be \( m_i \) and \( m_j \). Given \( e^k \), the total number of different \( pci(e^k, c^+_i, c^+_j) \) is \( m_i \ast m_j \). If \( m_i \ast m_j \) is an odd number, a majority vote can be obtained and \( pci(e^k, c_i, c_j) \) is either \( r \) or \( u \). However, if \( m_i \ast m_j \)
is even, it may occur that the votes for \( r \) and \( u \) are equal. Both \( u \) and \( r \) can define the causal interactions. In this case, we assign \( pci(e^k, c_i, c_j) = \text{null} \). Therefore, the subsequent search will consider candidate NATs with \( pci(e^k, c_i, c_j) \) being either \( r \) and \( u \).

With \( pci(e^k, c_i, c_j) \) assigned for each pair of causes (including the PLC), an effect-dependent PCI pattern \( pci(e^k) \) relative to \( e^k \) is defined. It is usually a partial PCI pattern since some bits are undefined (\text{null}). The number of effect-dependent PCI patterns is \( \eta \) (the number of active values of \( e \)). There are several ways to obtain candidate NATs with these PCI patterns. In the most efficient way, we select the PCI pattern with the least \text{null} \) bits. The selected PCI pattern generates the least candidate NATs. The least efficient way is to use all \( \eta \) PCI patterns to generate all possible candidate NATs. As explained at the beginning of this section, the selected PCI pattern will be used for the subsequent search by the similar way as introduced in Section 3.3.
Chapter 4

Experiments

To evaluate the accuracy and efficiency of general CPT approximation, we design and conduct experiments in this chapter. In Section 4.1, we introduce the experimental design. Particularly, we discuss the objectives and setup of the experiments. In Section 4.2, the experimental results are presented.

4.1 Experimental Design

In this section, we present the experimental design for the approximation of general CPTs, including objectives and setup. The experiments are designed for approximating CPTs without leaky causes or with NPLCs, and CPTs with PLCs. Target CPTs are randomly generated (see Subsection 4.1.1 and 4.1.2) or obtained from the real world BNs (see Subsection 4.1.3). They are organized into six experiments that are presented below.

4.1.1 Evaluating Approximation of General CPTs

Experiment 1 shows the necessity of flexible PCI extraction. In Section 3.3, we set several rules to softly define PCI patterns. The performance of well-defined
PCI pattern extraction is tested to show whether flexible PCI pattern extraction is necessary. Two batches of CPTs without leaky causes are randomly generated. Both of them are over \( n = 5 \) causes and the domain sizes of all variables are 4. Batch 1.1 contains 100 arbitrary CPTs and Batch 1.2 contains 100 NAT CPTs. Each NAT CPT is the CPT of a randomly selected NAT among 472 NATs when \( n = 5 \) (see Table 2.12 in Section 2.5). Given a target CPT, the equation of well-defined PCI pattern extraction (see Section 3.3) is applied relative to each active value of the effect. In the two batches, since the domain sizes are 4, the active values for the effect are \( e_1 \), \( e_2 \) and \( e_3 \). With \( n = 5 \), there are \( \binom{5}{2} = 10 \) pairs of causes. For each cause pair, there are \( 3 \times 3 = 9 \) pairs of active values. The PCI bit is well-defined as reinforcing if and only if the equation of \( r \) holds for all 9 pairs of active values. The same holds for undermining. A well-defined PCI pattern has between 0 and 10 well-defined PCI bits. This experiment is conducted using a laptop (Intel Core i7-3632QM at 2.2GHz).

The following experiments are conducted using the same laptop.

Experiment 2 examines the accuracy and efficiency of the CPT approximation. According to our method, we select a set of candidate NATs based on the partial PCI pattern. Thus, we need to test whether the selected candidate set is good enough to approximate target CPTs by NAT models. The 100 arbitrary CPTs in Batch 2.1 without leaky causes are randomly generated with \( n = 4 \) and the domain sizes are bounded by 4. The reason we chose \( n = 4 \) is that the total number of NATs is 52, while there are 472 NATs in total when \( n = 5 \). The domain sizes are bounded by 4 but not identical, which allows target CPTs to be more general. Each CPT is approximated by the procedure introduced in Section 3.3, which we refer to as...
NPLC-App. They are also approximated by exhaustively searching all 52 possible NATs, which we refer to as NPLC-Exh.

Experiment 3 compares the effectiveness of CPT approximation using NAT models with the well-known noisy-MAX model as a baseline [27]. Batch 3.1 and the Batch 3.2 consist of 100 arbitrary CPTs with $n = 5$ and $n = 6$, respectively. The domain sizes of CPTs in the two batches are bounded by 4. Including Batch 2.1 with $n = 4$ and domain size bounded by 4, three batches are examined in this experiment. Each CPT is approximated by NPLC-App and noisy-MAX models, referred to as NMAX. NMAX is seen as a special case of NPLC-App. Causal interactions in NMAX are reinforcing. Therefore, NMAX only searches one particular NAT where all causal events are connected by a multi-causal dual NIN-AND gate.

Table 4.1 summarizes the experiments of approximating general CPTs without PLCs, where the number of CPTs is denoted by ‘# of CPTs’ and the domain upper bound 4 is denoted as ‘≤ 4’. CPTs in Batch 1.2 are NAT CPTs. CPTs in other batches are arbitrary CPTs.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>CPT batches</th>
<th>NAT CPTs</th>
<th># of CPTs</th>
<th>$n$</th>
<th>Domain size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 1</td>
<td>Batch 1.1</td>
<td>100</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Batch 1.2</td>
<td>100</td>
<td>5</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td>Experiment 2</td>
<td>Batch 2.1</td>
<td>100</td>
<td>4</td>
<td>≤ 4</td>
<td></td>
</tr>
<tr>
<td>Experiment 3</td>
<td>Batch 2.1</td>
<td>100</td>
<td>4</td>
<td>≤ 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Batch 3.1</td>
<td>100</td>
<td>5</td>
<td>≤ 4</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Batch 3.2</td>
<td>100</td>
<td>6</td>
<td>≤ 4</td>
<td></td>
</tr>
</tbody>
</table>
4.1.2 Evaluation of CPTs with Persistent Leaky Causes

Experiment 4 evaluates the accuracy and efficiency of PLC-App against exhaustively searching all possible NAT models, referred to as PLC-Exh. Batch 4.1 consists of 30 arbitrary CPTs with PLCs with $n = 3$ and domain sizes are bounded by 4. The reason we select $n = 3$ is listed as follows. Because of the PLC, the number of causes in a candidate NAT is $n' = n + 1 = 4$. The total number of NATs we search is 52 (See Table 2.12 in Section 2.5). If we use $n = 4$ and $n' = 5$, the number of NATs in the search space is 472. Considering the total computational cost of PLC-Exh, we pick $n = 3$ and $n' = 4$. Each CPT is approximated by PLC-App and PLC-Exh.

Experiment 5 compares the effectiveness of CPT approximation using NAT models with that using noisy-MAX models. This experiment also demonstrates whether modeling PLCs performs better than not modeling them. Batch 5.1 consists of 50 arbitrary CPTs with PLCs with $n = 5$ causes and domain sizes bounded by 4. Each CPT is approximated by PLC-App, NPLC-App and NMAX.

Table 4.2 summarizes the experiments of approximating general CPTs with PLCs. CPTs in Batch 4.1 and 5.1 are arbitrary CPTs.

<table>
<thead>
<tr>
<th>Experiment</th>
<th>CPT batches</th>
<th># of CPTs</th>
<th>$n$</th>
<th>Domain size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Experiment 4</td>
<td>Batch 4.1</td>
<td>30</td>
<td>3</td>
<td>$\leq 4$</td>
</tr>
<tr>
<td>Experiment 5</td>
<td>Batch 5.1</td>
<td>50</td>
<td>5</td>
<td>$\leq 4$</td>
</tr>
</tbody>
</table>

KL distance is true only for PLC-App, but not for NPLC-App and NMAX. The functions to calculate KL distance can be found in Section 3.2. When $P_M(i) = 0$ and $P_T(i) > 0$, $KL(P_T, P_M)$ is undefined. Strictly speaking, the true $KL(P_T, P_M)$
cannot be obtained for NPLC-App and NMAX. In [27], it is suggested to replace 0 values by a small number for $P(e|c_0^1, c_0^2, ..., c_0^n)$. We define KL by omitting the CPD $P(e|c_0^1, c_0^2, ..., c_0^n)$. For example, a target CPT with $n = 4$ and domain sizes being 3 for all variables has $3^4$ CPDs. We omit the CPD $P(e|c_0^1, c_0^2, ..., c_0^n)$ and calculate the \textit{glorified KL distance} using the remaining $3^4 - 1$ CPDs.

\subsection*{4.1.3 Experiments on CPTs from Real World Bayesian Networks}

Experiment 6 evaluates the effectiveness of CPT approximation using NAT models over CPTs from real world BNs, and compares with noisy-MAX models. From the book website [7], we selected 10 real world BNs. For these BNs, the maximum number of parents per node is at least 3 and the maximum domain size of variables is at least 3. From these BNs, we selected 61 multi-valued CPTs with PLCs as Batch 6.1, where the number $n$ of parents (causes) is between 3 and 6 and the domain sizes of variables are between 2 and 63. Each CPT has at least one variable with the domain size larger than 2 so that it is a multi-valued CPT. The majority of parameter values are not extreme (equals 0 or 1). Table 4.3 summarizes the information of the selected BNs and CPTs.

We run PLC-App, NPLC-App and NMAX on this CPT batch. Considering the different sizes of real world CPTs, we divide the 61 CPTs into 5 groups by size $Sz$. Table 4.4 summaries the groups of real world CPTs with PLCs. The way to process all three methods are the same as we did for randomly generated CPTs with PLCs (see Subsection 4.1.2).
4.2 Experimental Results

4.2.1 Approximation of General CPTs

Experiment 1 extracts PCI patterns based on the definition of multi-valued causal interactions (see Subsection 2.4.6) over the Batch 1.1 and Batch 2.1. The results are shown relative to each active value of the effect. Figure 4.1, 4.2 and 4.3 show the number of well-defined PCI bits relative to $e^1$, $e^2$ and $e^3$, respectively. The results show that in the Batch 1.1 of 100 arbitrary CPTs (Arb), 97 CPTs have 0 well-defined PCI bit. In each of the remaining 3 CPTs, one well-defined PCI bit is extracted relative to each active cause of $e$. Thus, the rate of extracting well-defined PCI bits
relative to each active value of $e$ is 3%. In each CPT of Batch 1.2, 10 well-defined PCI bits are extracted relative to each active value of $e$.

The result shows that the well-defined PCI extraction differs significantly between arbitrary CPTs and NAT CPTs. It means that the flexible PCI pattern extraction introduced in Section 3.3 is necessary for an arbitrary CPT unless it is a NAT CPT.

Experiment 2 runs NPLC-App and NPLC-Exh over Batch 2.1. Figure 4.4 illustrates the resultant ED distance and runtime (RT in seconds) of the experiment.
Among target CPTs, the value of ED shows that NPLC-App yields close results to NPLC-Exh in most CPTs. Two programs even have the same results in some CPTs. RT shows that NPLC-App runs much faster than NPLC-Exh. The reason is that NPLC-App searches a set of candidate NATs obtained by flexible PCI extraction and NPLC-Exh searches all 52 NATs when $n = 4$.

Table 4.5 summarizes the performance of NPLC-App and NPLC-Exh. The space reduction is denoted by SR. It is the ratio of numbers of independent parameters between the target CPT and the approximated NAT model. For instance, if the CPT
has \( n = 4 \) and the domain sizes being 3 for all variables. The space reduction is \((3^5 - 1)/(3 \times 3 \times 4) = 6.75\). The value of SR shows how much space is reduced to approximate a target CPT by a NAT model. The result shows that CPT approximation by NAT models is efficient with the space reduction (14.670). Comparing NPLC-App with NPLC-Exh, we find that although NPLC-App has \( 0.204 - 0.189 \times 100\% \approx 8\% \) larger ED distance, but it runs \( \frac{41.258}{4.456} \approx 9 \) times faster than NPLC-Exh. NPLC-App has \( \frac{22.189 - 17.287}{17.287} \times 100\% \approx 28\% \) larger KL distance than NPLC-Exh.

### Table 4.5: Performance summary of NPLC-App and NPLC-Exh

<table>
<thead>
<tr>
<th></th>
<th>NPLC-App</th>
<th>NPLC-Exh</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>0.204</td>
<td>0.189</td>
</tr>
<tr>
<td>Stdev</td>
<td>0.043</td>
<td>0.036</td>
</tr>
<tr>
<td>ED</td>
<td>22.189</td>
<td>17.287</td>
</tr>
<tr>
<td>KL</td>
<td>4.456</td>
<td>3.849</td>
</tr>
<tr>
<td>RT</td>
<td>14.670</td>
<td>14.670</td>
</tr>
<tr>
<td>SR</td>
<td>6.908</td>
<td>6.908</td>
</tr>
</tbody>
</table>

The result shows that the candidate set includes good NATs even though it only covers a fraction of the search space. This also means that the flexible PCI pattern extraction performs well. We also found that NPLC-App saves plenty of time against NPLC-Exh. In other words, flexible PCI extraction reduces the search space significantly but includes good candidate NATs.

Experiment 3 runs NPLC-App and NMAX over Batch 2.1, Batch 3.1 and Batch 3.2. We demonstrate ED distance and runtime in seconds of each batch in Figure 4.5, Figure 4.6 and Figure 4.7, respectively. The three figures show the same result that NPLC-App yields a smaller error than NMAX. NMAX needs less runtime since it only evaluates one NAT (see Subsection 4.1.1) while NPLC-App processes a set of candidate NATs.
Table 4.6 summarizes ED, KL, RT and SR of NPLC-App and NMAX over three CPT batches. As \( n \) grows, ED distance of NMAX increases from 0.23 to 0.45, and ED of NPLC-App increases from 0.20 to 0.32. In particular, comparing to NMAX,
NPLC-App reduces ED distance to target CPTs by 13% for \( n = 4 \), 27% for \( n = 5 \) and 29% for \( n = 6 \). NMAX is twice the the KL distance of NPLC-App. NMAX runs about 10 times faster than NPLC-App. The space reduction of the two methods are identical with the same \( n \). The value of SR grows significantly as \( n \) grows for both methods.

**Table 4.6: Performance summary of NPLC-App and NMAX**

<table>
<thead>
<tr>
<th></th>
<th>NPLC-App</th>
<th></th>
<th>NMAX</th>
<th></th>
<th>NPLC-App</th>
<th></th>
<th>NMAX</th>
<th></th>
<th>NPLC-App</th>
<th></th>
<th>NMAX</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n = 4 )</td>
<td>Mean</td>
<td>Stdev</td>
<td>Mean</td>
<td>Stdev</td>
<td>Mean</td>
<td>Stdev</td>
<td>Mean</td>
<td>Stdev</td>
<td>Mean</td>
<td>Stdev</td>
<td>Mean</td>
</tr>
<tr>
<td>ED</td>
<td>0.20</td>
<td>0.04</td>
<td>0.23</td>
<td>0.07</td>
<td>0.27</td>
<td>0.08</td>
<td>0.37</td>
<td>0.10</td>
<td>0.32</td>
<td>0.09</td>
<td>0.45</td>
</tr>
<tr>
<td>KL</td>
<td>22.19</td>
<td>32.59</td>
<td>41.96</td>
<td>70.15</td>
<td>126.39</td>
<td>119.45</td>
<td>283.07</td>
<td>248.31</td>
<td>453.82</td>
<td>362.47</td>
<td>866.06</td>
</tr>
<tr>
<td>RT</td>
<td>4.46</td>
<td>3.85</td>
<td>0.47</td>
<td>0.32</td>
<td>14.45</td>
<td>16.54</td>
<td>1.49</td>
<td>1.36</td>
<td>54.12</td>
<td>54.41</td>
<td>4.86</td>
</tr>
<tr>
<td>SR</td>
<td>14.67</td>
<td>6.91</td>
<td>14.67</td>
<td>6.91</td>
<td>36.60</td>
<td>20.73</td>
<td>36.60</td>
<td>20.73</td>
<td>89.96</td>
<td>53.32</td>
<td>89.96</td>
</tr>
</tbody>
</table>

In summary, both NPLC-App and NMAX significantly reduce the space complexity. NPLC-App is more accurate than NMAX within an acceptable range of runtime.

### 4.2.2 Approximating CPTs with Persistent Leaky Causes

Experiment 4 runs PLC-App and PLC-Exh over Batch 4.1. Figure 4.8 illustrates ED and RT in seconds of the experiment. ED shows that among the target CPTs, PLC-App yields close results to PLC-Exh in most CPTs. Particularly, PLC-App returns the same NAT model as PLC-Exh does in 14 out of 30 CPTs. RT shows that PLC-App runs much faster than PLC-Exh.

To further study the accuracy and efficiency of PLC-App, we summarize ED, KL, RT and SR in Table 4.7. ED distance of PLC-App (0.145) is slightly larger than PLC-Exh (0.142). PLC-App has \( \frac{2.919 - 2.791}{2.791} * 100\% \approx 5\% \) larger KL distance than
PLC-Exh. PLC-App only uses 23% of runtime.

<table>
<thead>
<tr>
<th></th>
<th>PLC-App</th>
<th></th>
<th>PLC-Exh</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Stdev</td>
<td>Mean</td>
<td>Stdev</td>
</tr>
<tr>
<td>ED</td>
<td>0.145</td>
<td>0.033</td>
<td>0.142</td>
<td>0.032</td>
</tr>
<tr>
<td>KL</td>
<td>2.919</td>
<td>1.918</td>
<td>2.791</td>
<td>1.826</td>
</tr>
<tr>
<td>RT</td>
<td>13.807</td>
<td>11.840</td>
<td>59.132</td>
<td>39.948</td>
</tr>
<tr>
<td>SR</td>
<td>5.468</td>
<td>1.392</td>
<td>5.468</td>
<td>1.392</td>
</tr>
</tbody>
</table>

The result shows that the new PCI pattern extraction for CPTs with PLCs introduced in Section 3.4 significantly reduces the number of candidate NATs for subsequent search. The percentage of time saving is limited by the small \( n \). It is expected to be more efficient for larger \( n \).

Experiment 5 compares PLC-App, NPLC-App and NMAX over Batch 5.1. Figure 4.9 illustrates ED and RT in seconds of the all three methods. The result shows that the error of PLC-App and NPLC-App are smaller than NMAX. PLC-App has the smallest error. PLC-App needs more RT in most CPTs than NPLC-App. RT of NMAX is the smallest within three methods.

In Table 4.8, the performance of PLC-App, NPLC-App and NMAX is summa-
Figure 4.9: Euclidean distance (left) and runtime (right) for PLC-App, NPLC-App and NMAX over Batch 5.1

rized. ED distances of PLC-App (0.234) and NPLC-App (0.256) are smaller than NMAX (0.381). PLC-App results in the smallest error. Even though KL distances of NPLC-App and NMAX are glorified (see Subsection 4.1.2), PLC-App still has the smallest KL distance (96.181). In addition, we notice that the value SR of NPLC-App and NMAX are identical and larger than that of PLC-App. Since PLC-App explicitly models PLCs, the number of parameters in NAT models generated by PLC-App is more than that by NPLC-App and NMAX, which leads to a smaller space reduction. The value of RT shows that PLC-App is nearly doubles the runtime of NPLC-App. Because of the PLC, the NAT CPT generated by PLC-App doubles the size of that by NPLC-App (see Section 3.2). The extra computation needed to model PLCs increases the runtime.

Table 4.8: Performance summary of PLC-App, NPLC-App and NMAX

<table>
<thead>
<tr>
<th></th>
<th>PLC-App</th>
<th>NPLC-App</th>
<th>NMAX</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>Stdev</td>
<td>Mean</td>
</tr>
<tr>
<td>ED</td>
<td>0.234</td>
<td>0.074</td>
<td>0.256</td>
</tr>
<tr>
<td>KL</td>
<td>96.181</td>
<td>88.654</td>
<td>126.470</td>
</tr>
<tr>
<td>RT</td>
<td>56.318</td>
<td>60.508</td>
<td>21.971</td>
</tr>
<tr>
<td>SR</td>
<td>33.590</td>
<td>16.414</td>
<td>36.726</td>
</tr>
</tbody>
</table>
In summary, CPT approximation by NAT models has higher accuracy than by noisy-MAX. Explicitly modeling PLCs further improves the accuracy when PLCs exist.

### 4.2.3 Approximation of CPTs from Real World Bayesian Networks

Experiment 6 runs PLC-App, NPLC-App and NMAX over Batch 6.1 of CPTs from real world BNs. The result is summarized in Table 4.9 which is grouped by CPT size. According to the value of ED, we see that PLC-App and NPLC-App have smaller errors than NMAX, and PLC-App has the smallest error in all CPT

<table>
<thead>
<tr>
<th>Size</th>
<th>PLC-App Mean</th>
<th>PLC-App Stddev</th>
<th>NPLC-App Mean</th>
<th>NPLC-App Stddev</th>
<th>NMAX Mean</th>
<th>NMAX Stddev</th>
</tr>
</thead>
<tbody>
<tr>
<td>Small</td>
<td>ED 0.137</td>
<td>0.091</td>
<td>0.185</td>
<td>0.067</td>
<td>0.205</td>
<td>0.074</td>
</tr>
<tr>
<td></td>
<td>KL 13.698</td>
<td>32.069</td>
<td>14.328</td>
<td>35.456</td>
<td>19.792</td>
<td>47.808</td>
</tr>
<tr>
<td></td>
<td>RT 12.809</td>
<td>12.257</td>
<td>4.488</td>
<td>8.806</td>
<td>0.847</td>
<td>1.024</td>
</tr>
<tr>
<td>Medium</td>
<td>ED 0.183</td>
<td>0.095</td>
<td>0.210</td>
<td>0.060</td>
<td>0.246</td>
<td>0.071</td>
</tr>
<tr>
<td></td>
<td>KL 48.195</td>
<td>36.661</td>
<td>42.868</td>
<td>30.896</td>
<td>62.776</td>
<td>43.706</td>
</tr>
<tr>
<td>Large</td>
<td>ED 0.217</td>
<td>0.089</td>
<td>0.226</td>
<td>0.087</td>
<td>0.244</td>
<td>0.098</td>
</tr>
<tr>
<td></td>
<td>KL 211.017</td>
<td>131.538</td>
<td>219.400</td>
<td>137.476</td>
<td>279.443</td>
<td>188.657</td>
</tr>
<tr>
<td></td>
<td>RT 86.071</td>
<td>78.217</td>
<td>61.653</td>
<td>40.555</td>
<td>8.470</td>
<td>5.259</td>
</tr>
<tr>
<td>Very Large</td>
<td>ED 0.167</td>
<td>0.074</td>
<td>0.175</td>
<td>0.077</td>
<td>0.185</td>
<td>0.081</td>
</tr>
<tr>
<td></td>
<td>KL 279.790</td>
<td>203.445</td>
<td>284.209</td>
<td>214.142</td>
<td>333.145</td>
<td>225.559</td>
</tr>
<tr>
<td></td>
<td>RT 367.830</td>
<td>230.646</td>
<td>368.703</td>
<td>434.143</td>
<td>45.120</td>
<td>56.379</td>
</tr>
<tr>
<td>Massive</td>
<td>ED 0.187</td>
<td>0.047</td>
<td>0.190</td>
<td>0.046</td>
<td>0.255</td>
<td>0.058</td>
</tr>
<tr>
<td></td>
<td>RT 4089.622</td>
<td>2945.405</td>
<td>1577.831</td>
<td>788.058</td>
<td>120.576</td>
<td>35.303</td>
</tr>
<tr>
<td></td>
<td>SR 117.124</td>
<td>56.598</td>
<td>121.154</td>
<td>58.714</td>
<td>121.154</td>
<td>58.714</td>
</tr>
</tbody>
</table>
groups. Based on KL distance, the solution also holds. As we can see, the largest ED distance obtained by PLC-App of real world CPTs (0.217 in group Large) is smaller than that of randomly generated CPTs in Table 4.8 (0.234). This shows PLC-App performs better when applied over real world CPTs than in randomly generated CPTs. In addition, space reduction shows that as the CPT size increases, the CPT approximation by NAT models becomes more efficient.

We summarize the result of comparing three methods over CPTs from real world BNs as follows.

1. As the number of parameters in CPTs grows, approximating CPTs by multi-valued NAT models significantly reduces the space complexity.

2. CPT approximation by multi-valued NAT models is more accurate than the well-known noisy-MAX models.

3. For a CPT with a PLC, explicitly modeling the PLC leads to a smaller error of the CPT approximation than not modeling it.
Chapter 5

Conclusion

5.1 Summary

A Bayesian network represents causal dependence and conditional independence among domain variables. The strength of dependence is quantified by a CPT for each variable. The size of each CPT is exponential on the number of parents of the variable. Causal independence models such as NAT models are proposed to reduce the number of parameters in a CPT. Approximating a CPT by a NAT model significantly reduces the space complexity and speeds up BN inference.

The contributions of this thesis are listed as follows.

- We extended the CPT approximation to over multi-valued variables by setting new rules for PCI pattern extraction.
- We developed constrained gradient descent searching for parameters in each candidate NATs.
- To approximate a CPT with a persistent leaky cause (PLC), we developed a novel method to express the causal interactions among normal causes and the PLC.
• We developed a new method to overcome the difficulty in extracting causal interactions between two normal causes when approximating a CPT with a PLC.

• The experimental results demonstrated that the methods to approximate CPTs with or without PLCs by NAT models are efficient and more accurate than the well-known noisy-MAX.

• The experiments showed that explicitly modeling PLCs improves the accuracy of the approximation when target CPTs have PLCs.

• The experiments are also carried out over CPTs from real world BNs. The CPT approximation by NAT models performs well in such CPTs.

5.2 Limitation and Future Work

The following limitation in this work can be identified. The KL distance was computed effectively over each CPD and then accumulated over all CPDs. To compare of three methods over CPTs from real world BNs, we grouped CPTs by size. When comparing CPTs in the same group size, cumulative KL distance correctly shows the effectiveness of all three methods. However, the KL distance as defined does not compare well across groups of different sizes, as each has different number of CPDs. Average KL distance would facilitate a better comparisons between different CPT groups.

A number of future work related to this research can be identified. Tensor-based
decomposition [11, 12] is another way to approximate CPTs. It is worthwhile to compare accuracy, space reduction and time efficiency between CPT approximations with NAT models and with tensor-based decomposition.

In this thesis, target CPTs are over ordinal variables, such as fever which has the domain \{normal, low, high\}. The NAT model approximation over categorical variables (also known as nominal variables), such as gender which has the domain \{male, female\}, can be further studied.

How to take advantage of NAT models in BN inference has been studied [16, 21]. We have approximated arbitrary CPTs by NAT models. The approximated NAT models can be further examined by comparing posteriors from BNs with arbitrary CPTs and BNs with NAT model approximation of CPTs.
Bibliography


