CPT Approximation with NIN-AND Tree Causal Models

by

Qing Liu

A Thesis
Presented to
The University of Guelph

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

Guelph, Ontario, Canada

© Qing Liu, January, 2014
ABSTRACT

CPT Approximation with NIN-AND Tree Causal Models

Qing Liu
University of Guelph, 2014

A Bayesian network (BN) is a probabilistic graphical model widely used in artificial intelligence (AI) to support uncertain knowledge representation and uncertain reasoning. In BNs, conditional probability tables (CPTs) are used to store quantitative knowledge. The size of a discrete variable’s CPT increases exponentially on the number of related causes. Causal models are proposed to specify a CPT with fewer parameters. Non-Impeding Noisy-AND tree (NIN-AND tree) causal modeling is an expressive causal model which can encode two types of causal interactions. Replacing a CPT with a NIN-AND tree causal model can save storage space, and speed up inference. Being motivated by the advantages of using NIN-AND tree causal modeling, we develop the techniques to approximate a given CPT with NIN-AND tree causal modeling in this research. In particular, we develop a suite of algorithms to approximate an arbitrary CPT of binary variables with a NIN-AND tree model. Based on the experimental results, the methods proposed in this research can result in reasonably good approximation accuracy.
Acknowledgments

I would like to express genuine gratitude to my advisor Dr. Yang Xiang, and my advisory committee member Dr. Fangju Wang for their invaluable encouragement and guidance. I would also like to thank my parents, who have supported me more than 20 years. Finally, I would like to thank all the faculty members in the School of Computer Science of University of Guelph for making the school such a wonderful place.
# Table of Contents

List of Tables vi

List of Figures vii

1 Introduction 1
   1.1 Problem Statement ................................. 1
   1.2 Thesis Statement ................................ 1
   1.3 Overview ........................................ 1
   1.4 Organization of Thesis ................................. 4

2 Background 5
   2.1 Basic Concepts of Graph ............................... 5
      2.1.1 Directed Graph ................................ 6
      2.1.2 Directed Acyclic Graph ............................ 7
   2.2 Basic Concepts of Probability .......................... 7
      2.2.1 Frequency Probability and Bayesian Probability ....... 7
      2.2.2 Basic Rules of Probability ........................ 8
      2.2.3 Conditional Independence and Reasoning By Case ....... 10
   2.3 Bayesian Networks ................................. 11
      2.3.1 Basic Concepts of Bayesian networks ................. 12
      2.3.2 Inference Using BN ............................... 15
   2.4 Causal Models: Noisy-OR and Noisy-AND ................. 19
      2.4.1 Noisy-OR ........................................ 20
      2.4.2 Noisy-AND ....................................... 24
   2.5 Non-Impeding Noisy-AND Tree Causal Modeling .......... 26
      2.5.1 Causal Interactions ............................... 27
      2.5.2 NIN-AND Gates .................................... 28
      2.5.3 NIN-AND Tree and Pairwise Causal Interaction Function ... 33
      2.5.4 Specify A CPT with A NIN-AND Tree Model ............ 36
      2.5.5 Minimal NIN-AND Trees and Root Labeled Trees ........ 38
   2.6 Leak Cause ........................................ 42

3 Approximating CPTs with NIN-AND Tree Models 44
   3.1 Problem Analysis and Approximation Process .............. 45
   3.2 Distance Measurement .................................. 48
   3.3 Minimal NIN-AND Tree Indexing and BST Construction .......... 49
3.3.1 Minimal NIN-AND Tree Indexing ........................................ 50
3.3.2 BST Construction ............................................................ 57
3.4 Search Key Extraction and Search Using A BST .................. 60
  3.4.1 Search Key Extraction and Scoring ................................. 60
  3.4.2 Search NIN-AND Trees Using A BST ............................. 66
3.5 CPT Approximation with Gradient Descent Given A NIN-AND Tree 78
  3.5.1 Numerical Derivative ...................................................... 79
  3.5.2 Gradient Descent ............................................................ 81

4 Experimental Design and Results ........................................ 87
  4.1 Experimental Design ......................................................... 87
    4.1.1 Tasks of Experiments .................................................. 87
    4.1.2 Data Set ..................................................................... 88
    4.1.3 Optimal CPT Approximation with Noisy-OR and NIN-AND Trees 89
    4.1.4 Average Runtime of Gradient Descent Algorithm ............. 91
    4.1.5 Evaluating Effectiveness of PCI Scoring ....................... 92
    4.1.6 Evaluating Effectiveness of Search Key Extraction ........... 93
  4.2 Experimental Results ......................................................... 94
    4.2.1 Experimental Result of Optimal CPT Approximation .......... 94
    4.2.2 Experimental Result of Average Runtime of Gradient Descent Algorithm 96
    4.2.3 Experimental Result of Evaluating PCI Scoring Algorithm ... 97
    4.2.4 Experimental Result of Evaluating Search Key Extraction Algorithm 98

5 Conclusion ................................................................. 106
  5.1 Summary ................................................................. 106
  5.2 Contributions ........................................................... 107
  5.3 Limitations ................................................................. 108
  5.4 Future Work ............................................................... 108

Bibliography ............................................................. 109
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>The CPT of <em>cough</em> in Figure 2.2</td>
<td>13</td>
</tr>
<tr>
<td>2.2</td>
<td>The JPD with 3 binary variables</td>
<td>14</td>
</tr>
<tr>
<td>2.3</td>
<td>The CPT of <em>fire</em></td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>The CPT of <em>smoke</em></td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>The CPT of <em>alarm</em></td>
<td>16</td>
</tr>
<tr>
<td>2.6</td>
<td>The given SCPs</td>
<td>22</td>
</tr>
<tr>
<td>2.7</td>
<td>The CPT of the noisy-OR example</td>
<td>24</td>
</tr>
<tr>
<td>2.8</td>
<td>The CPT of the 3 causes example</td>
<td>26</td>
</tr>
<tr>
<td>2.9</td>
<td>The CPT of the direct NIN-AND gate</td>
<td>31</td>
</tr>
<tr>
<td>2.10</td>
<td>The SCPs of the NIN-AND tree shown in Figure 2.9</td>
<td>36</td>
</tr>
<tr>
<td>2.11</td>
<td>The CPT of the NIN-AND tree shown in Figure 2.9</td>
<td>38</td>
</tr>
<tr>
<td>2.12</td>
<td>The CPT of the NIN-AND tree shown in Figure 2.9</td>
<td>40</td>
</tr>
<tr>
<td>2.13</td>
<td>Number of root labeled trees</td>
<td>41</td>
</tr>
<tr>
<td>3.1</td>
<td>A 2 causes target CPT</td>
<td>62</td>
</tr>
<tr>
<td>3.2</td>
<td>A target CPT</td>
<td>65</td>
</tr>
<tr>
<td>4.1</td>
<td>The data set consisting of CPTs</td>
<td>89</td>
</tr>
<tr>
<td>4.2</td>
<td>The Wilcoxon rank sum test for 5-cause experiment results</td>
<td>105</td>
</tr>
</tbody>
</table>
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>(a) is an undirected graph. (b) is a directed graph.</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>A simple DAG</td>
<td>12</td>
</tr>
<tr>
<td>2.3</td>
<td>A causal chain of 3 variables</td>
<td>15</td>
</tr>
<tr>
<td>2.4</td>
<td>Causal chain with virtual boxes</td>
<td>17</td>
</tr>
<tr>
<td>2.5</td>
<td>$die_1$ and its causes</td>
<td>23</td>
</tr>
<tr>
<td>2.6</td>
<td>3 Causes Example</td>
<td>25</td>
</tr>
<tr>
<td>2.7</td>
<td>A direct NIN-AND gate</td>
<td>29</td>
</tr>
<tr>
<td>2.8</td>
<td>A dual NIN-AND gate</td>
<td>31</td>
</tr>
<tr>
<td>2.9</td>
<td>An example of NIN-AND tree</td>
<td>34</td>
</tr>
<tr>
<td>2.10</td>
<td>Two NIN-AND Trees</td>
<td>39</td>
</tr>
<tr>
<td>2.11</td>
<td>(a) a minimal NIN-AND tree with dual leaf gate; (b) a minimal NIN-AND tree with direct leaf gate; (c) a root labeled tree.</td>
<td>41</td>
</tr>
<tr>
<td>2.12</td>
<td>Causes of headache (with leak node)</td>
<td>43</td>
</tr>
<tr>
<td>3.1</td>
<td>Function t</td>
<td>46</td>
</tr>
<tr>
<td>3.2</td>
<td>Framework of approximation process</td>
<td>47</td>
</tr>
<tr>
<td>3.3</td>
<td>A root labeled tree and its related NIN-AND tree</td>
<td>56</td>
</tr>
<tr>
<td>3.4</td>
<td>The BST construction with 3 causes</td>
<td>57</td>
</tr>
<tr>
<td>3.5</td>
<td>The 3-cause BST</td>
<td>59</td>
</tr>
<tr>
<td>3.6</td>
<td>The NIN-AND tree selected</td>
<td>66</td>
</tr>
<tr>
<td>3.7</td>
<td>Solid curve: experimental results; Dotted curve: estimated results.</td>
<td>68</td>
</tr>
<tr>
<td>3.8</td>
<td>A simple example of NIN-AND tree search</td>
<td>71</td>
</tr>
<tr>
<td>3.9</td>
<td>A direct leaf gate NIN-AND tree and a dual leaf gate NIN-AND tree</td>
<td>71</td>
</tr>
<tr>
<td>3.10</td>
<td>Caption for trace</td>
<td>72</td>
</tr>
<tr>
<td>3.11</td>
<td>Caption for twotraces</td>
<td>76</td>
</tr>
<tr>
<td>3.12</td>
<td>Compare Equation (3.4) and (3.5)</td>
<td>81</td>
</tr>
<tr>
<td>3.13</td>
<td>Gradient descent gradually approaches to the local minimum.</td>
<td>82</td>
</tr>
<tr>
<td>3.14</td>
<td>An example of gradient descent when the local minimum outside of the boundary</td>
<td>86</td>
</tr>
<tr>
<td>4.1</td>
<td>Flow chart of the experiment of optimal CPTs approximation</td>
<td>91</td>
</tr>
<tr>
<td>4.2</td>
<td>Flow chart of the experiment evaluating PCIs scoring algorithm</td>
<td>93</td>
</tr>
<tr>
<td>4.3</td>
<td>Flow chart of the experiment evaluating search key extraction algorithm</td>
<td>94</td>
</tr>
<tr>
<td>4.4</td>
<td>WEDs from approximating with NIN-AND tree causal modeling (left side box) and with the noisy-OR (right side box).</td>
<td>95</td>
</tr>
<tr>
<td>4.5</td>
<td>Average runtime of gradient descent algorithm</td>
<td>96</td>
</tr>
</tbody>
</table>
4.6 Experimental results for evaluating PCIs scoring algorithm . . . . . . . 98
4.7 WEDs for approximating 3-cause CPT . . . . . . . . . . . . . . . . . . 99
4.8 Average runtime for approximating 3-cause CPTs . . . . . . . . . . . . 100
4.9 WEDs for approximating 4-cause CPT . . . . . . . . . . . . . . . . . . 101
4.10 Average runtime for approximating 4-cause CPTs . . . . . . . . . . . . 102
4.11 WEDs for approximating 5-cause CPT . . . . . . . . . . . . . . . . . . 103
4.12 Average runtime for approximating 5-cause CPTs . . . . . . . . . . . . 104
Chapter 1

Introduction

1.1 Problem Statement

To take advantage of NIN-AND tree causal modeling, CPTs in a BN can be substituted by NIN-AND tree models. The problem that this thesis addresses is how to approximate an arbitrary CPT of binary variables with a NIN-AND tree model.

1.2 Thesis Statement

We propose a PCI function based representation of NIN-AND trees, which is used to select a subset of NIN-AND trees. Using the selected NIN-AND trees and an optimization method, the NIN-AND tree models giving good approximation accuracy can be found.

1.3 Overview

An intelligent agent should be able to represent the knowledge of the environment concisely and reason rationally. In the early period of AI, much research has been done with rule-based AI systems. Later on, scientists realized that rule-based
systems may not be able to work in uncertain environments. The main challenge of applying rule-based systems in uncertain environments is to concisely represent uncertain background knowledge with certain rules, for which no real solution based on rule-based AI has been proposed [12]. In 1980s’, Judea Pearl published his probabilistic reasoning study, which attracted AI scientists’ interests and increased the acceptance of probabilistic reasoning in AI [8]. The Bayesian network (BN) was introduced at that time [6]. BNs overcome many problems of rule-based AI systems and early probabilistic systems because of the capability of encoding uncertain background knowledge and enabling efficient uncertain reasoning.

A Bayesian network (BN) is a graphical model consisting of nodes and directed edges (detailed in Section 2.1). Each node represents an environmental variable. A Bayesian network quantifies the strength of the dependency between adjacent nodes with conditional probabilities, which are stored in conditional probability tables (CPTs). Each CPT is associated with one node. The complexity of specifying a CPT increases exponentially based on the number of its parents (The parent $N_p$ of variable $N_c$ is the cause of the variable $N_c$, and we say $N_c$ is the effect or child of $N_p$). This property of CPTs can rise difficulty in both perspectives of knowledge acquisition and inference.

In some special cases, the complexity of specifying a CPT may not increase exponentially on the number of related causes. When we say the complexity of specifying CPTs increases exponentially, we are assuming that the parameters of the CPT are arbitrarily valued. However, the parameters of a CPT may interact with each other in special ways (standard patterns), and scientists take advantage of these
standard patterns to address the problem of exponentially increasing complexity of CPT specification. If there are standard patterns, a CPT can be specified with fewer parameters. Hence, specifying all parameters in the CPT is not necessary in this situation. Causal independence is studied as a type of standard pattern expected to exist commonly. Specifying the CPT satisfying causal independence can be done with linear complexity based on the number of related causes (detailed in Sections 2.4 and 2.5). The techniques based on causal independence assumptions are called causal models. For example, the noisy-OR is the most commonly used causal model, and up to 50% of CPTs with binary variables could be reasonably approximated by the noisy-OR [20].

For most causal models (including the noisy-OR), there is a limitation. They assume that causal interactions are reinforcing, which means the impacts of several causes on the effect will reinforce each other. However, in the real world, the impacts of causes can undermine each other as well, which is called undermining causal interaction. When the problem domain strictly follows the assumption of reinforcing causal interaction, few error will be introduced with applying the causal models. However, this is not realistic in most situations. In fact, error would always be generated after applying causal model because of the inadequacy of the causal interaction assumption. Therefore, considering both types of causal interaction can decrease the errors being generated. NIN-AND tree causal modeling is the causal model which is capable of encoding both undermining and reinforcing causal interactions explicitly [16]. Hence, NIN-AND tree causal modeling is an expressive causal model. In addition, the recent work of Xiang enables NIN-AND tree causal modeling to speed up
inference in BNs [14].

In this research, we develop techniques to optimally approximate any given CPT (having binary variables only) with a NIN-AND tree model to take advantage of NIN-AND tree causal modeling. Unlike other causal models, NIN-AND tree causal modeling introduces NIN-AND tree topologies (or simply called NIN-AND trees). For each CPT, a NIN-AND tree has to be picked to properly represent the causal interactions. However, the number of possible NIN-AND trees increases exponentially based on the number of the parents. To simplify the problem, the technique of dynamically eliminating undesired NIN-AND trees is developed first. Then, we develop the technique to approximate a CPT with a fixed NIN-AND tree. Moreover, the approximation accuracy comparison experiment is conducted between NIN-AND tree causal modeling and the noisy-OR. The experimental results show that approximation accuracy with NIN-AND tree causal modeling is notably improved.

1.4 Organization of Thesis

The thesis report is organized as follows. In Chapter 2, the necessary background knowledge is explained. Chapter 3 illustrates the methods which we propose to optimally approximate a CPT with a NIN-AND tree model. Chapter 4 presents the experimental study and report the experimental results. Finally, Chapter 5 concludes the research, analyzes the limitations, and discusses the possible future work.
Chapter 2

Background

This chapter explains the background knowledge required for the rest of this thesis. Because Bayesian networks are founded from graph theory and Bayesian probability theory, in Sections 2.1 and 2.2, basic concepts of graph theory and Bayesian probability theory are explained. Section 2.3 discusses concepts of Bayesian network and the inference process in Bayesian network with a simple example. Section 2.4 introduces causal models in terms of the noisy-OR and the noisy-AND. Section 2.5 outlines NIN-AND tree causal modeling. Section 2.6 briefly introduces the concept of leak cause.

2.1 Basic Concepts of Graph

In the area of computer science, a graph is a visualization tool, which can be used to model structures of problems concisely. A graph consists of a set of nodes or vertices representing variables of a problem domain (in the context of this thesis) and a set of edges encoding the interactions between the variables. Hence, a graph can be defined with the notation $G = (V, E)$, where $V$ denotes the set of nodes and $E$ denotes the set of edges.
Edges of a graph can be either undirected or directed. Graphs with directed edges are called *directed graph*, while graphs with undirected edges are called *undirected graphs* (see Figure 2.1). Because Bayesian networks are directed graphs, here we only focus on directed graph.

### 2.1.1 Directed Graph

As defined above, a directed graph consists of a set of nodes and a set of directed edges. Each directed edge links a pair of nodes, and encodes the interaction between them. Suppose that we have a pair of adjacent nodes $x$ and $y$, and the edge $e$ directed from $x$ to $y$. Then, edge $e$ is denoted by an ordered pair $(x, y)$. Node $x$ is called the *tail* of $e$. Node $y$ is called the *head* of $e$. Moreover, node $x$ is the *parent* of node $y$, and node $y$ is a *child* of node $x$. Suppose that we have two nodes $u$ and $v$ of a directed graph. If there is a path directed from $u$ to $v$ and all nodes on the path only appears once, then node $v$ is a *descendant* of node $u$. Let us look at two examples shown in Figure 2.1 (b). We can find nodes $a$ and $b$ and the edge linking them, $e$ or $(b, a)$. Then

![Figure 2.1: (a) is an undirected graph. (b) is a directed graph.](image)
we say $b$ is the tail of $e$ and $a$ is the head of $e$. Likewise, we can say $a$ is the child of $b$, and $b$ is the parent of $a$. In addition, there are only two paths directed from $c$ to other nodes, which are $\langle c, b \rangle$ and $\langle c, b, a \rangle$. Hence, the set of the nodes besides $c$ on these paths is $\{b, a\}$, which is also the set of descendants of $c$.

2.1.2 Directed Acyclic Graph

A directed acyclic graph (DAG or ADG) is a directed graph which is acyclic or contains no simple cycle. A simple cycle can be defined as a path starts from node $a$ and ends up with the same node $a$, and no repeated node can be found on the path besides of node $a$. In Figure 2.1 (a), $\langle c, b, d, e, c \rangle$ is a simple cycle. A directed graph contains a simple cycle when we can find a path $\pi$ where every node $n_i$ on $\pi$ is the head of one edge $e_j$ in $\pi$, while $n_i$ is the tail of another edge $e_m$ in $\pi$. In Figure 2.1 (b), no simple cycle can be found. Therefore, (b) is a DAG.

2.2 Basic Concepts of Probability

2.2.1 Frequency Probability and Bayesian Probability

The term probability can be interpreted and used in different ways. In this subsection, we explain two common ways to interpret probability.

Probability interpreted based on frequency is called frequentist probability. Frequentist probability is objective, and can only be derived when enough repeated cases have been observed. Frequency is the ratio of the number of occurrences of event $N_A$ over the number of trials $N$, which is written as $f(A) = \frac{N_A}{N}$. Frequentist probability
is the limit of the frequency when the number of trials approaches to infinity, which
is equivalent to \( P_f(A) = \lim_{N \to \infty} \frac{N_A}{N} \). Therefore, when the event \( A \) cannot be easily
repeated, frequentist probability \( P_f(A) \) is undefined. Another way to interpret prob-
ability is the Bayesian interpretation. Probability under the Bayesian interpretation
is called the \textit{Bayesian probability}. Bayesian probability is subjective. The Bayesian
view of probability sees probability as someone’s degree of belief (or simply someone’s
belief) of the event. Therefore, we say Bayesian probability is based on the knowledge
of the event, and it is not necessary to observe cases to have a degree of belief. For
example, to determine the probability of getting a head when flip a fair coin, frequent-
tists have to toss enough times to determine the probability, \( P_f(H) \), which would be
extremely close to 0.5. However, Bayesian scientists can obtain Bayesian probability
\( P_B(H) = 0.5 \) based on the knowledge that the coin is fair without doing trials.

In the rest of the thesis, we adopt Bayesian interpretation.

\subsection*{2.2.2 Basic Rules of Probability}

In this subsection we introduce some basic rules of Bayesian probability to be
used in this thesis. We use lowercase letters to denote variables, and capital letters
to denote sets of variables. We may also use subscripts to differentiate variables (or
sets), for example \( v_1, ..., v_n \). To denote a value of a variable, we add a subscript to
the variable, for example the \( i_{th} \) value of variable \( v \) or \( v_m \) is \( v_i \) or \( v_{mi} \). We define a
problem domain as a set of discrete variables \( D = \{v_1, ..., v_n\} \). All variables we refer
to in this subsection belong to \( D \).

Probabilities are real numbers valued between 0 and 1, which is equivalent to
$0 \leq \text{Prob} \leq 1$. The probability has to be 1 when it represents certainty. For example, for a binary variable $x$, the probability $P(x = x_1 \lor x = x_2)$ always equals to 1.

There are two important concepts, conditional probability and distribution. Conditional probabilities are the probabilities in the form similar to $P(x = x_i | y = y_j)$, where notation “$|$” should be read as conditional on or given. This conditional probability represents the belief on the value of variable $x$ being $x_i$ given the knowledge or observation that variable $y$ is in value $y_j$. We use single variables in this example, but it can be generalized to multiple variables.

A probability distribution over the variable $x$ is a set of probabilities, one for each value of variable $x$. The probability distribution over $x$ is denoted by $P(x | v_i = v_{ic}, \ldots, v_n = v_{nb})$, where $v_i = v_{ic}, \ldots, v_n = v_{nb}$ is the knowledge or observation of the values of other variables. Under Bayesian interpretation, an unconditional probability $p(v)$ should be viewed as a conditional probability where the relevant knowledge is left implicit. Therefore, the distribution over variable $x$ can also be written as $P(x)$. A probability distribution involving multiple variables is called a joint probability distribution (JPD), and is denoted by $P(x, y)$ when it contains two variables. A JPD is a set of joint probabilities (each joint probability is also called a parameter of the JPD). Each joint probability corresponds to one possible combination of values of all variables involved.

Given a JPD of two variables $x$ and $y$, the distribution over variable $x$ can be derived by marginalization with the following formula.

$$P(x) = \sum_i P(x, y = y_i)$$ (2.1)
Negation rule shows the relationship between the belief on variable $x$’s value being $x_i$ and the belief on variable $x$’s value not being $x_i$. If we observe the value $y_i$ of variable $y$, negation rule can be written as

$$P(x \neq x_i|y = y_j) = 1 - P(x = x_i|y = y_j). \quad (2.2)$$

Product rule is an important rule in Bayesian probability calculation. Suppose that we observe variable $y$’s value being on $y_j$. Product rule can be defined as

$$P(x = x_i \land y = y_j) = P(x = x_i|y = y_j)P(y = y_j). \quad (2.3)$$

Bayes’ theorem reveals the interaction between the variables before “|” and the variables after “|”. Bayes’ theorem is derived from product rule. With the variables $x$ and $y$, Bayes’ theorem can be defined as the following equation.

$$P(x = x_i|y = y_j) = \frac{P(y = y_j|x = x_i)P(x = x_i)}{P(y = y_j)} \quad (2.4)$$

2.2.3 Conditional Independence and Reasoning By Case

In this subsection, we focus on conditional independence, Markov property, and chain rule, which underlies inference mechanism of BNs.

Conditional independence indicates a type of interactions among variables. Suppose that $x$ is conditionally independent of $y$ given $z$, which is denoted by $I(x, z, y)$. Then observation of the value $y$ provides no information to the value of $x$ whenever we observe the value of $z$. We use single variables here, but it can be generalized to multiple variables. In addition, $I(x, z, y)$ holds if and only if the following equations
are satisfied.

\[ P(x = x_i | y = y_j, z = z_m) = P(x = x_i | z = z_m) \]  
(2.5)

\[ P(y = y_j | x = x_i, z = z_m) = P(y = y_j | z = z_m) \]  
(2.6)

Combining the last two equations with product rule, we will have

\[ P(x = x_i, y = y_j | z = z_m) = P(x = x_i | z = z_m)P(y = y_j | z = z_m). \]  
(2.7)

Given the variables and defined conditional probabilities, reasoning by case can be applied. Suppose that we have binary variables \(x, y\) and \(z\), and conditional probabilities \(P(x = x_i | y = y_j, z = z_m)\) and \(P(y = y_i | z = z_m)\) are known. Then conditional probability \(P(x = i | z = z_m)\) can be derived by applying reasoning by case, which is

\[ P(x = i | z = z_m) = \]

\[ P(x = x_i | y = y_j, z = z_m)P(y = y_j | z = z_m) + P(x = x_i | y \neq y_j, z = z_m)P(y \neq y_j | z = z_m), \]

In the last equation, \(P(y \neq y_j | z = z_m)\) is calculated by \(1 - P(y = y_j | z = z_m)\) based on the negation rule.

\section*{2.3 Bayesian Networks}

In this subsection, we will introduce some basic concepts and properties of Bayesian networks. In addition, an example is given to illustrate the features of Bayesian networks.
2.3.1 Basic Concepts of Bayesian networks

Bayesian networks (BN) is a tool that supports uncertain knowledge representation and uncertain reasoning with probability [10, 9]. A BN has both graphical and probabilistic components. A BN can be denoted by a triplet \((V, G, P)\), where \(V\) denotes variables in the problem domain, \(G\) is a graph, and \(P\) denotes local conditional probabilities. \(G\) is an acyclic directed graph (AGD, or DAG). Each node of \(G\) in our research represents a discrete variable \(v\) in the problem domain. The edge linking a pair of adjacent nodes represents the causal dependence between these nodes. For example, Figure 2.2 is a DAG. In the figure, variable cough is dependent of cold, allergy, and cp. We say cough is the parent/cause of cp; meanwhile, cough is the child/effect of cold and allergy respectively.

![Figure 2.2: A simple DAG](image)

Each node \(v\) of \(G\) is associated with a conditional probability table (CPT). The CPT of \(v\) is a set of conditional probability distributions over \(v\), each of which is conditioned on one combination of the values of \(v\)'s parents. Each conditional probability
of a CPT is also called a parameter. Table 2.1 is the CPT of cough. In the table, $T$ and $F$ denote the values of binary variables. For example, $\text{cold} = T$ denotes the value of cold is true, and the symptom of the disease is present. $\text{cold} = F$ denotes the value of cold is false, and the symptom of the disease is absent. The number of parameters of a CPT of $v$ can be calculated by the equation $n_v = (d - 1)k^j$, where $d$ denotes the number of possible values of $v$, $k$ denotes the number of possible values of $v$’s parents, and $j$ is the number of $v$’s parents. Because we only consider binary variables here, the equation is equivalent to $n_v = k^j$. Hence, the number of parameters of a CPT is exponential on the number of parents of the variable.

| cold | allergy | $P(\text{cough} = T|\text{cold}, \text{allergy})$ |
|------|---------|------------------------------------------|
| F    | F       | 0.2                                      |
| F    | T       | 0.7                                      |
| T    | F       | 0.8                                      |
| T    | T       | 0.94                                     |

In early probabilistic AI, JPDs were used to handle uncertainty in environments. However, scientists soon realized the intractability of knowledge acquisition and inference using JPD. This is because the number of parameters of JPD increases exponentially on the number of variables. For example, suppose that the problem domain contains $r$ variables, and each variable has $k$ possible values. Then, the number of parameters of a JPD is up to $k^r$. Table 2.2 is a JPD involving 3 binary variables in terms of fire, smoke, and alarm. Therefore, the number of parameters of the JPD is $2^3 = 8$. When $r$ grows, specifying such a JPD can be too expensive. For example, when $j$ grows to 100, it is almost impossible to acquire enough parameters.
(this problem is known as intractability of knowledge acquisition).

BNs solve this problem by encoding conditional independence with DAGs. In a BN, each variable is conditionally independent of its non-descendant variables given its parents. Hence, the DAG of a BN does not just encode causal dependency, but also explicitly encodes conditional independence among variables. By taking advantage of encoded conditional independence, a BN normally has much fewer parameters compared with a JPD having the same amount of variables, and inference using BNs performs efficiently. Suppose that we have 10 binary variables, each variable has up to two parents. The total parameters of CPTs of such a BN is up to $10 * 2^2 = 40$. However, the number of parameter of such a JPD is $2^{10} = 1024$. Therefore, BNs dramatically reduce the amount of parameters to acquire, and solve the intractability of knowledge acquisition problem. In addition, given a BN, a unique JPD with all variables can be defined by reasoning by case. This property is shown by the equation below.

$$P(V) = \prod_{v \in V} P(v|\pi(v)).$$  \hspace{1cm} (2.8)

<table>
<thead>
<tr>
<th>fire</th>
<th>smoke</th>
<th>alarm</th>
<th>$P($fire, smoke, alarm)</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>0.57</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>0.1</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>0.00375</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>0.0125</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>0.1425</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>0.025</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>0.03375</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>0.1125</td>
</tr>
</tbody>
</table>
2.3.2 Inference Using BN

In this subsection, we make two examples of reasoning in a simple BN (a causal chain of 3 variables shown in Figure 2.3). In the first example, inference is done using the JPD, which is built based on Equation (2.8) with the CPTs of the BN. The CPTs of this BN is given in Table 2.3, Table 2.4 and Table 2.5. The JPD is shown in Table 2.2. For the second example, inference is done using message passing, which directly deals with CPTs.

![Figure 2.3: A causal chain of 3 variables](image)

Table 2.3: The CPT of fire

<table>
<thead>
<tr>
<th>$P(fire = T)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.25</td>
</tr>
</tbody>
</table>

Table 2.4: The CPT of smoke

| fire | $P(smoke = T|fire)$ |
|------|---------------------|
| F    | 0.05                |
| T    | 0.5                 |


Table 2.5: The CPT of $\text{alarm}$

| smoke | $P(\text{alarm} = T | \text{smoke})$ |
|-------|-------------------------------|
| F     | 0.2                           |
| T     | 0.9                           |

Reasoning by Using JPDs

Suppose that $\text{fire} = T$ is observed, and we want to reason about the value of $\text{alarm}$ being true, i.e., $P(\text{alarm} = T | \text{fire} = T)$. The reasoning process using the JPD is shown below.

1. We want to convert $P(\text{alarm} = T | \text{fire} = T)$ into the form which does not contain conditional probabilities. Then, using product rule, we have

   $$P(\text{alarm} = T | \text{fire} = T) = \frac{P(\text{alarm} = T, \text{fire} = T)}{P(\text{fire} = T)}.$$

2. Calculate $P(\text{alarm} = T, \text{fire} = T)$ and $P(\text{fire} = T)$ by marginalization. Then we have

   $$P(\text{alarm} = T, \text{fire} = T) = \sum_{i} P(\text{alarm} = T, \text{fire} = T, \text{smoke} = \text{smoke}_i),$$
   $$P(\text{fire} = T) = \sum_{i} \sum_{j} P(\text{alarm} = \text{alarm}_i, \text{fire} = T, \text{smoke} = \text{smoke}_j).$$

3. By combining the last two steps, we have

   $$P(\text{alarm} = T | \text{fire} = T) = \frac{\sum_{i} P(\text{alarm} = T, \text{fire} = T, \text{smoke} = \text{smoke}_i)}{\sum_{j} \sum_{i} P(\text{alarm} = \text{alarm}_j, \text{fire} = T, \text{smoke} = \text{smoke}_i)}.$$

These joint probabilities are the parameters of the JPD. Hence, we get $P(\text{alarm} = T | \text{fire} = T) = 0.1375/0.25 = 0.55$. However, in the last equation, the denominator
is \( \sum_{j} \sum_{t} P(\text{alarm} = \text{alarm}_j, \text{fire} = T, \text{smoke} = \text{smoke}_t) \), of which the number of involved parameters is \( 2^{m-n} = 2^{4-2} = 4 \), where \( m \) is the number of variables in the domain, and \( n \) is the number of variables instantiated. This means the number of involved parameters grows exponentially with \( m \).

**Reasoning by Message Passing**

For BNs, inference can be done by message passing. Message passing is a method passing messages between each pair of adjacent computation units on the path, where each unit contains a variable \( v \) and \( v \)’s parents. We can use a virtual box to represent a computation unit. A message of \( v \) is the belief on \( v = v_i \) given the observation. With these ideas, Figure 2.3 can be converted to Figure 2.4. For each unit (or a box), it accepts an input message, and generates an output message based on the input. The input of the first box is the observation, and the output of the last box is the inference result. Suppose that we want to obtain the belief \( P(\text{alarm} = T | \text{fire} = T) \) by message passing, where \( \text{fire} = T \) is the observation. The reasoning process is shown below.

![Figure 2.4: Causal chain with virtual boxes](image-url)
1. As shown in Figure 2.4, in the starting box consisting of fire and smoke, the input message is the observation, which is \( P(\text{fire} = T|\text{fire} = T) \). Then, the output message \( P(\text{smoke} = T|\text{fire} = T) \) can be obtained based on the input message and the CPT of smoke. In particular,

\[
P(\text{smoke} = T|\text{fire} = T) \\
= P(\text{smoke} = T|\text{fire} = T)P(\text{fire} = T|\text{fire} = T) \\
+ P(\text{smoke} = T|\text{fire} = F)P(\text{fire} = F|\text{fire} = T),
\]

where \( P(\text{fire} = F|\text{fire} = T) \) can be obtained by the negation rule and equals to \( 1 - P(\text{fire} = T|\text{fire} = T) \). Hence, \( P(\text{smoke} = T|\text{fire} = T) = 0.5 \times 1 + 0.05 \times (1 - 1) = 0.5 \).

2. Then, \( P(\text{smoke} = T|\text{fire} = T) \) is used as the input message of the next box consisting of smoke and alarm. Using the same idea, we have

\[
P(\text{alarm} = T|\text{fire} = T) \\
= P(\text{alarm} = T|\text{smoke} = T)P(\text{smoke} = T|\text{fire} = T) \\
+ P(\text{alarm} = T|\text{smoke} = F)P(\text{smoke} = F|\text{fire} = T) \\
= P(\text{alarm} = T|\text{smoke} = T)P(\text{smoke} = T|\text{fire} = T) \\
+ P(\text{alarm} = T|\text{smoke} = F)(1 - P(\text{smoke} = T|\text{fire} = T)) \\
= 0.9 \times 0.5 + 0.2 \times (1 - 0.5) = 0.55.
\]

We can find that the amount of computation involved in each box is the same (2 multiplications and 2 additions). This means that if we have \( n \) variables in the causal chain, the total amount of computation is up to \( 4n \). Therefore, we can conclude that
the complexity of message passing is linear on the number of variables. In addition, reasoning by message passing obtains the same result as reasoning using the JPD in this example. It shows that inference by message passing can be done correctly in the BN. For message passing in more complex BNs, see [13].

2.4 Causal Models: Noisy-OR and Noisy-AND

Using BNs can dramatically reduce the number of parameters to be obtained. However, the number of parameters of a CPT increases exponentially on the number of parents. If the data set is not big enough or experts’ knowledge is not available, it still can be difficult to gain all the parameters of the CPTs of the BN [10].

Scientists try to reduce the number of parameters of CPTs by looking for standard patterns existing in the CPTs to be acquired. Causal models are one type of techniques proposed for this purpose. Instead of allowing arbitrary distributions in CPTs, CPTs of causal models are distributed based on the rules called causal independence. If a CPT satisfies causal independence, the impact on the effect of a cause is independent of the impact of other causes. By introducing causal independence, causal models enable to specify all parameters of CPTs with fewer parameters, whose size can be linear on the number of causes.

To describe causal independence, we introduce the following notations. Let $e$ denote the effect variable (binary). Let $C$ denote the set of all cause variables (binary) of $e$. Let $C_i$ denote a subset of $C$. Let $c_i$ denote the $i_{th}$ cause variable of $C$. Let $e^+$, $c_i^+$, and $C_i^+$ denote the presence of effect $e$, cause $c_i$ and all causes of $C_i$ respectively.
Let $e^-$, $c_i^-$, and $C_i^-$ denote the absence of effect $e$, cause $c_i$ and all causes of $C_i$ respectively. Let $e \leftarrow c_i (e \not\leftarrow c_i)$ denote a causal event that effect is produced (not produced) by cause $c_i$, when all other causes are absent. For example, Suppose that $e$ have three binary causes. Then $P(e^+ \leftarrow c_1^+ c_3^+)$ denotes the causal probability of producing $e$ when $c_1$ and $c_3$ are present and the unlisted cause ($c_2$) is absent. It is equivalent to conditional probability $P(e^+|c_1 = \text{present}, c_2 = \text{absent}, c_3 = \text{present})$.

In the next two subsections, we consider the two common causal models that are most relevant to this thesis, the noisy-OR and the noisy-AND. The noisy-OR was pioneered by Good [3], and Pearl[10] introduced it in BNs. The noisy-AND was proposed by Díez [2]. A number of alternative causal models have also been proposed, including the noisy-MAX [5], the noisy-MIN [1], the recursive noisy-OR [7], and the noisy-ADD [4].

### 2.4.1 Noisy-OR

The noisy-OR is one of the most well-known causal models, which deals with binary variables. A CPT with $n$ binary causes has $2^n$ parameters. However, if the CPT follows the assumptions of the noisy-OR, it can be specified with $n$ parameters, which is linear on the number of the causes. The noisy-OR have three assumptions [10].

1. All causes of the effect are modeled.
   
   This assumption is to ensure the probability of producing the effect is 0 when all modeled causes are absent, namely $P(e^+ \leftarrow C^-) = 0$. However, when build a
CPT, scientists may not explicitly model a cause of an effect if they think that the impact of this cause on the effect is weak. For example, having iced water in winter may cause headache, but the probability $P(headache^+ ← icedwater^+)$ is very small. Hence, scientists may not model it as an explicit cause of $headache$.

In practice, scientists may create another cause called other (or called leak cause [20]) to represent all not explicitly modeled causes.

2. The causes can fail to cause the effect.

This assumption indicates that the probability of producing the effect is less than 1 if some causes are present. This assumption is equivalent to $P(e^+ ← C_i^+) < 1$, where $C_i ⊆ C$.

3. Failure independence.

Failure independence is the most important assumption of the noisy-OR. It means, for a noisy-OR model, a cause $c_i$ failing to produce $e$ is independent of the event that another cause $c_j$ failing to produce $e$. For example, suppose that the probability of landing up point 1 each time tossing a die is known as $P(die_1^+ ← t^+) = q$. The probability of not landing up point 1 after rolling 3 times is denoted by $P(die_1^+ ∉ t_{1^+}, t_{2^+}, t_{3^+})$. It can be obtained with failure independence and $P(die_1^+ ← t^+)$. Because each toss is independent of the other toss, the probability of not landing up point 1 in three tosses equals to the product of probabilities of not landing up point 1 in each toss. Hence, we have

$$P(die_1^+ ∉ t_{1^+}, t_{2^+}, t_{3^+}) = P(die_1^+ ∉ t_{1^+})P(die_1^+ ∉ t_{2^+})P(die_1^+ ∉ t_{3^+}) = q^3.$$
This equation is an example of failure independence. In general, if failure independence is satisfied, we have the following equation:

\[ P(e^+ \not\leftarrow C^+) = \prod_{c_i \in C} P(e^+ \not\leftarrow c_i^+). \] (2.9)

By combining the last equation with the negation rule, we have

\[ P(e^+ \leftarrow C^+) = 1 - P(e^+ \not\leftarrow C^+) = 1 - \prod_{c_i \in C} (1 - P(e^+ \leftarrow c_i^+)). \] (2.10)

The probability of producing \( e \) when only one cause is present, \( P(e^+ \leftarrow c_i^+) \), is called a single causal probability (SCP). The number of SCPs in a CPT is equal to the number of causes. The equation above shows that all other parameters of a CPT can be obtained by multiplying related SCPs if failure independence is satisfied, which is the basic idea of specifying a CPT with SCPs for the noisy-OR. The example below illustrates the steps of specifying a CPT with the noisy-OR.

Suppose that we want to have the die landing up with point 1, we can roll the die 2 times. The structure of the problem is given in Figure 2.5. The SCPs are given in Table 2.6. Then, a CPT can be obtained by the following steps.

| \( t_1 \) | \( t_2 \) | \( P(die_1^+|t_1, t_2) \) |
|---------|---------|----------------|
| F       | F       | -              |
| F       | T       | 0.166          |
| T       | F       | 0.166          |
| T       | T       | -              |

1. The first parameter to be obtained is \( P(die_1^+|t_1^-, t_2^-) = P(die_1^+ \leftarrow t_1^-, t_2^-) \).
Based on the first assumption of the noisy-OR, we have $P(\text{die}_1^+ \leftarrow t_1^-, t_2^-) = P(\text{die}_1^+ \leftarrow C^-) = 0$.

2. At this point, $P(\text{die}_1^+|t_1^+, t_2^+) = P(\text{die}_1^+ \leftarrow t_1^+, t_2^+)$ is the only unknown parameter, and it can be obtained based on failure independence, Equation (2.10). Hence, we have the equations below.

$$P(\text{die}_1^+ \leftarrow t_1^+, t_2^+) = 1 - P(\text{die}_1^+ \not\leftarrow t_1^+, t_2^+)$$
$$= 1 - P(\text{die}_1^+ \not\leftarrow t_1^+)P(\text{die}_1^+ \not\leftarrow t_2^+)$$
$$= 1 - (1 - P(\text{die}_1^+ \leftarrow t_1^+))(1 - P(\text{die}_1^+ \leftarrow t_2^+))$$
$$= 1 - (1 - 0.166)(1 - 0.166) = 0.304444$$

Finally, we build the CPT shown in Table 2.7 with two SCPs. As the number of causes grows up, the advantage of the noisy-OR becomes more obvious. For example, when there are 7 causes, the CPT will have $2^7 = 128$ parameters. However, it can be specified with only 7 SCPs using the noisy-OR. This feature dramatically reduces the number of parameters to be acquired. In addition, we can find in Table 2.7, the
probability of producing the effect when \( n \) causes are present is always higher than the probability of producing the effect when \( m \) (\( m < n \)) causes are present. This property is called *reinforcing causal interaction*, which is encoded by the noisy-OR [16].

| \( t_1 \) | \( t_2 \) | \( P(\text{die}_1^+ | t_1, t_2) \) |
|---|---|---|
| F | F | 0 |
| F | T | 0.166 |
| T | F | 0.166 |
| T | T | 0.304444 |

### 2.4.2 Noisy-AND

The noisy-AND is also a well-known causal model. If the assumptions of the noisy-AND are satisfied, we also can construct a CPT with related SCPs. Similar to noisy-OR, the noisy-AND has three assumptions [2].

1. All causes of the effect are modeled.

2. The effect is present if and only if all causes are present.

   This assumption implies that the probability of producing the effect is 0 whenever a cause is absent. This property can be represented by the equation \( P(e^+ \leftarrow C_i^+) = 0 \), where \( C_i \neq C \). Because of this property of the noisy-AND, we say the noisy-AND is impeding.

3. Success independence.

   Success independence is opposite to failure independence. For success inde-
pendence, a cause $c_i$ succeeding to produce $e$ is independent of the event that another cause $c_j$ succeeding to produce $e$. For example, after a car accident, the recovery process can be modeled with a BN shown by Figure 2.6 [2]. In the figure, $c_i$ denotes the event that "patient starts to be treated injury $i$", $e$ represents the event that "patient is complete recovered", and let $z_i$ denote the event that "injury $i$ is recovered". Then, success independence can be applied as

$$P(e^+ \leftarrow c_1^+, c_2^+, c_3^+) = P(z_1^+ \leftarrow c_1^+)P(z_2^+ \leftarrow c_2^+)P(z_3^+ \leftarrow c_3^+).$$

In general, we have the following equation,

$$P(e^+ \leftarrow C^+) = \prod_{c_i \in C} P(z_i^+ \leftarrow c_i^+). \quad (2.11)$$

Figure 2.6: 3 Causes Example

Suppose that in Figure 2.6 $P(z_1^+ \leftarrow c_1^+) = 0.9$, $P(z_2^+ \leftarrow c_2^+) = 0.8$ and $P(z_3^+ \leftarrow c_3^+) = 0.7$ are given. If the CPT of $e$ satisfies the assumptions of the noisy-AND, the CPT can be specified by the following steps.

1. Based on the second assumption of the noisy-AND, the first 7 parameters, which has at least one cause being absent, can be specified as 0. (The parameters are shown in Table 2.8.)
2. Based on success independence, the last parameter, \( P(e^+|c_1^+, c_2^+, c_3^+) \) (equivalent to \( P(e^+ \leftarrow c_1^+, c_2^+, c_3^+) \)) can be obtained by

\[
P(e^+ \leftarrow c_1^+, c_2^+, c_3^+) = P(z_1^+ \leftarrow c_1^+)P(z_2^+ \leftarrow c_2^+)P(z_3^+ \leftarrow c_3^+)
\]

\[
= 0.9 \times 0.8 \times 0.7
\]

\[
= 0.504.
\]

Table 2.8: The CPT of the 3 causes example

| \( c_1 \) | \( c_2 \) | \( c_3 \) | \( P(e^+|c_1, c_2, c_3) \) |
|---|---|---|---|
| F | F | F | 0 |
| F | F | T | 0 |
| F | T | F | 0 |
| F | T | T | 0 |
| T | F | F | 0 |
| T | F | T | 0 |
| T | T | F | 0 |
| T | T | T | 0.504 |

To sum up, we obtain all 8 parameters of the CPT with only 3 parameters, which is linear on the number of causes. In addition, for a CPT specified with the noisy-AND, a parameter is larger than 0 if and only if all causes are present. Hence, we say that the noisy-AND also encodes reinforcing causal interaction [16].

2.5 Non-Impeding Noisy-AND Tree Causal Modeling

*Non-impeding noisy-AND tree causal modeling* is also called NIN-AND tree causal modeling, and it is the causal model proposed by Xiang and Jia [16].

If variable \( x \)'s CPT can be represented by NIN-AND tree causal modeling, the CPT of \( x \) can be specified fewer parameters (normally SCPs), whose size is linear
on the number of $x$’s parents. Instead of only encoding the reinforcing causal interaction, NIN-AND tree causal modeling is capable of encoding both undermining and reinforcing causal interactions. To express different causal interactions with one model, NIN-AND tree causal modeling introduces \textit{NIN-AND trees} (or NIN-AND tree topologies). Because of the capability of encoding both types of causal interactions, NIN-AND causal modeling is more expressive [16]. With Xiang’s recent study [15], NIN-AND tree causal modeling can also be used to speed up inference. Moreover, NIN-AND tree causal modeling can be generalized to multi-dimensional variables [14]. However, in this research, we focus on studying binary variables.

\textbf{2.5.1 Causal Interactions}

Intuitively, there are two types of causal interactions, reinforcement and undermining. If $P(e^+ \leftarrow C_i^+) \text{ is smaller than or equals to } P(e^+ \leftarrow C_i^+, C_j^+)$, the causal interaction between causal events $e^+ \leftarrow C_i^+$ and $e^+ \leftarrow C_j^+$ is reinforcement. For example, either being tired or having a fever can result in headache. If the person is tired while having fever, it is more likely he is suffering from headache. The probability of having headache caused by being tired alone is denoted by $P(\text{headache}^+ \leftarrow \text{tired}^+) = p$, and the probability of having headache caused by fever alone is denoted by $P(\text{headache}^+ \leftarrow \text{fever}^+) = q$. The person having two symptoms at the same time is more likely to have headache, which means $P(\text{headache}^+ \leftarrow \text{tired}^+, \text{fever}^+) \text{ is larger than both } p \text{ and } q$. Then, the causal interaction between causal events $\text{headache}^+ \leftarrow \text{tired}^+$ and $\text{headache}^+ \leftarrow \text{fever}^+$ is
reinforcement. A general representation of reinforcement is

\[
P(e \leftarrow C^+_i, C^+_j) \geq \text{Max}(P(e \leftarrow C^+_i), P(e \leftarrow C^+_j)).
\]

On the opposite, if \(P(e^+ \leftarrow C^+_i, C^+_j)\) is larger than \(P(e^+ \leftarrow C^+_i)\), the causal interaction between causal events \(e^+ \leftarrow C^+_i\) and \(e^+ \leftarrow C^+_j\) is reinforcement. For example, either eating crab or eating peanuts is good for health. However, if a person eats peanuts while having crab, the person may get diarrhea. A mathematic representation is

\[
P(\text{healthy}^+ \leftarrow \text{peanuts}^+, \text{crab}^+) < MIN(P(\text{healthy}^+ \leftarrow \text{crab}^+), P(\text{healthy}^+ \leftarrow \text{peanuts}^+)).
\]

Then, the causal interaction between causal events \(\text{healthy}^+ \leftarrow \text{crab}^+\) and \(\text{healthy}^+ \leftarrow \text{peanuts}^+\) is undermining. In general, undermining between \(e^+ \leftarrow C^+_i\) and \(e^+ \leftarrow C^+_j\) can be represented by

\[
P(e\leftarrow C^+_i, C^+_j) < \text{MIN}(P(e \leftarrow C^+_i), P(e \leftarrow C^+_j)).
\]

### 2.5.2 NIN-AND Gates

As we described, NIN-AND tree causal modeling introduces NIN-AND trees to encode causal interactions. A NIN-AND tree consists of a number of NIN-AND gates. A NIN-AND gate is designed to encode one particular type of causal interaction.

There are two types of NIN-AND gates, **direct NIN-AND gates** (shown in Figure 2.7) and **dual NIN-AND gates** (shown in Figure 2.8). A NIN-AND gate can be denoted by \(g = (\text{Gate}, \text{In}, \text{Out}, \text{Link})\). **Gate** indicates the gate node of \(g\). The symbol of **Gate** is similar to the symbol of logic AND gates. **In** denotes a set of input nodes. An input node is labeled with a causal event \(e^+ \leftarrow C^+_i\) (for direct NIN-AND gates) or
\( e^+ \not\leftarrow C_i^+ \) (for dual NIN-AND gates). Every \( g \) has an output node, denoted by \( \text{Out} \). Let \( C \) denotes the set of all the causes associated with \( g \). Then, the output node is labeled with a causal event \( e^+ \leftarrow C^+ \) (for direct NIN-AND gates) or \( e^+ \not\leftarrow C^+ \) (for dual NIN-AND gates). We use Link to denote a set of forward links. Every input node is connected to \( \text{Gate} \) by a forward link, and the \( \text{Gate} \) is connected to the output node by a forward link. Forward links are directed edges in the direction of normal digital gates. However, for the purpose of simplicity, we draw all the links as undirected edges.

Let us look at dual gates and direct gates in details in the following subsections.

**Direct NIN-AND Gate**

![Direct NIN-AND Gate Diagram](image)

**Figure 2.7: A direct NIN-AND gate**

Direct NIN-AND gates are used to encode undermining causal interactions. A direct NIN-AND gate has 3 assumptions shown below.

1. The causes can fail to cause the effect.

   This assumption implies that the probabilities of producing the effect is in
the range (0, 1) when at least one cause is present. Because the noisy-AND forces the parameters having absent causes to be 0, this assumption makes direct NIN-AND gates different from the noisy-AND. Therefore, we use the term non-impeding to describe it [16].

2. Success conjunction [16].

Success conjunction implies that causal event \( e^+ \leftarrow C_1^+, ..., C_n^+ \) is equivalent to the conjunction of all related single causal events, which is \((e^+ \leftarrow C_1^+) \land ... \land (e^+ \leftarrow C_n^+)\). Success conjunction of a direct NIN-AND gate assumes that all of its associated causes satisfy success conjunction.

3. Success independence.

If an causal event \( e^+ \leftarrow C_i^+ \) is independent of the causal event \( e^+ \leftarrow C_j^+ \), these two causal events satisfy success independence. A direct NIN-AND gate assumes that all of its associated causes satisfy success independence. Hence, we have the equation

\[
P((e^+ \leftarrow C_1^+) \land ... \land (e^+ \leftarrow C_n^+)) = P(e^+ \leftarrow C_1^+) \cdot ... \cdot P(e^+ \leftarrow C_n^+).
\]

By combining success conjunction and success independence, a general form of equation can be generated.

\[
P(e^+ \leftarrow C_1^+, ..., C_n^+) = \prod_{i=1}^{n} P(e^+ \leftarrow C_i^+),
\]

(2.12)

where \( e^+ \leftarrow C_1^+, ..., C_n^+ \) is the causal event that labels the output node, and \( e^+ \leftarrow C_i^+ \) is the single causal event that labels each input node.

Suppose that \( e \) has two causes \( c_1 \) and \( c_2 \), and the SCPs \( P(e^+ \leftarrow c_1^+) = 0.9 \) and \( P(e^+ \leftarrow c_2^+) = 0.8 \) are known. In addition, the problem domain satisfies the
assumptions of direct NIN-AND gates. Based on the first assumption of direct NIN-AND gates, the first parameter of the CPT, \( P(e^+ \leftarrow c_1^-, c_2^-) \), is automatically assigned with 0. Then, there is only one unknown parameter, \( P(e^+ \leftarrow c_1^+, c_2^+) \). Based on Equation (2.12), it equals to the product of two SCPs, which is \( 0.9 \times 0.8 = 0.72 \). Then, the CPT of the direct NIN-AND gate is specified (shown in Table 2.9). It is obvious that \( 0.72 < MIN(0.8, 0.9) \), which shows that direct NIN-AND gates encode undermining causal interaction.

| \( c_1 \) | \( c_2 \) | \( P(e^+ | c_1, c_2) \) |
|----------|----------|-----------------|
| F        | F        | 0               |
| F        | T        | 0.8             |
| T        | F        | 0.9             |
| T        | T        | 0.72            |

Table 2.9: The CPT of the direct NIN-AND gate

Dual NIN-AND Gate

Dual NIN-AND gates are used to encode reinforcing causal interaction. A dual
NIN-AND gate has the following assumptions, which are the same as the noisy-OR.

1. The causes can fail to cause the effect.

2. Failure conjunction.

Failure conjunction implies that causal event \( e^+ \not\leftarrow C_1^+ , \ldots , C_n^+ \) is equivalent to the conjunction of all related single causal event, which is \( e^+ \not\leftarrow C_1^+ \land \ldots \land e^+ \not\leftarrow C_n^+ \). Failure conjunction of a dual NIN-AND gate assumes that all of its associated causes satisfy failure conjunction.

3. Failure independence.

If an causal event \( e^+ \not\leftarrow C_i^+ \) is independent of the causal event \( e^+ \not\leftarrow C_j^+ \), these two causal events satisfy failure independence. A dual NIN-AND gate assumes that all of its associated causes satisfy failure independence. Hence, we have the equation

\[
P((e^+ \not\leftarrow C_1^+) \land \ldots \land (e^+ \not\leftarrow C_n^+)) = P(e^+ \not\leftarrow C_1^+) \ldots P(e^+ \not\leftarrow C_n^+).
\]

By combining failure conjunction and failure independence, a general form of equation can be generated.

\[
P(e^+ \not\leftarrow C_1^+, \ldots , C_n^+) = \prod_{i=1}^{n} P(e^+ \not\leftarrow C_i^+), \tag{2.13}
\]

where \( e^+ \not\leftarrow C_1^+, \ldots , C_n^+ \) is the causal event of the output node, and \( e^+ \not\leftarrow C_i^+ \) is the single causal event of each input node. For the belief computation, we also can use the equation below.

\[
P(e^+ \leftarrow C_1^+, \ldots , C_n^+) = 1 - \prod_{i=1}^{n} (1 - P(e^+ \leftarrow C_i^+)). \tag{2.14}
\]

Dual NIN-AND gates have similar assumptions as the noisy-OR, and encode reinforcing causal interaction. Hence, dual NIN-AND gates are close to the noisy-OR.
Because NIN-AND tree causal modeling also encodes undermining causal interaction with direct NIN-AND gates, we say that NIN-AND tree causal modeling is a generalization of the noisy-OR [16].

2.5.3 NIN-AND Tree and Pairwise Causal Interaction Function

In this subsection, we discuss NIN-AND trees and the pairwise causal interaction function.

NIN-AND Tree

A NIN-AND tree $T$ is a tree topology consisting of direct NIN-AND gates and dual NIN-AND gates. Supposed that $T$ consists of some NIN-AND gates $g_i$. If $g_1$ is connected to $g_2$, the output node of $g_1$ is connected to an input node of $g_2$ with either a forward link or a negation link. The definition of forwards links is given in the previous subsection. A negation link is a link with a circle at the head. A negation link is needed if and only if $g_1$ and $g_2$ are different NIN-AND gates. In particular, the causal event on the head of a negation link is the negated form of the causal event on the tail of the negation link. For example, suppose that $g_1$ and $g_2$ are connecting by a negation link, and the output node of $g_1$ is labeled with $e^+ \leftarrow c_i^+$. Then, the input node of $g_2$ is labeled with $e^+ \not\leftarrow c_i^+$. (However, the label $e^+ \not\leftarrow c_i^+$ is often omitted as it can be obtained directly based on the label of the node on the tail of the negation link.)

For a NIN-AND gate of a NIN-AND tree $T$, if the NIN-AND gate associates with all causes of $T$, this NIN-AND gate is called a leaf gate of $T$. If an input node
of a NIN-AND gate is not connecting with another NIN-AND gate, this input node is called a root. Figure 2.9 shows a NIN-AND tree having 3 causes. The NIN-AND tree consists of 2 NIN-AND gates: a direct NIN-AND gates $g_2$ and a dual NIN-AND gate $g_1$. The NIN-AND gate $g_2$ in this figure is the leaf gate. The input nodes of $g_1$ and the input node of $g_2$ on the left side are the roots.

![Figure 2.9: An example of NIN-AND tree](image)

It is assumed by a NIN-AND tree that all causes in the problem domain are modeled. This means that every cause can be found in a root. It also implies that the probability of producing the effect has to be 0 when all causes are absent. This property can be described by the equation $P(e^+ \leftarrow C^-) = 0$ [16].
Pairwise Causal Interaction Function

Pairwise causal interaction (PCI) is the type of the causal interaction between a pair of causes of an effect. For a NIN-AND tree $T$, each pair of causes $(c_i, c_j)$ is corresponding to a PCI, and is denoted by $pci(c_i, c_j)$. In the path from roots to the leaf, if $g_i$ is the first NIN-AND gate whose output causal event (the causal event labeled on the output node) contains both causes $c_1$ and $c_2$, the value of $pci(c_1, c_2)$ is determined by the type of $g_i$. In particular, if $g_i$ is a direct NIN-AND gate, which encodes undermining causal interaction, we have $pci(c_1, c_2) = umd$, where $umd$ means undermining causal interaction. If $g_i$ is a dual NIN-AND gate, which encodes reinforcing causal interaction, we have $pci(c_1, c_2) = ref$, where $ref$ denotes reinforcing causal interaction. For example, suppose that we want to determine the value of $pci(c_1, c_3)$ of the NIN-AND tree shown in Figure 2.9. The first NIN-AND gate on the path from roots to the leaf is $g_1$, the output causal event of $g_1$ does not contain $c_1$ and $c_3$ at the same time. Therefore, we look at the next NIN-AND gate, which is $g_2$. In $g_2$, we find $c_1$ and $c_3$ at the output causal event of $g_2$. Therefore, $pci(c_1, c_3) = umd$ is determined by the undermining causal interaction encoded by $g_2$.

Let us consider the situation that we have a set $C$ of $n$ causes, $\{c_1, ..., c_n\}$. Then, there are $\binom{n}{2} = \frac{n(n-1)}{2}$ possible pairs of causes $(c_i, c_j)$. Suppose that we have another set $I$, $\{umd, ref\}$. Then, a PCI function (PCIF) can be defined. A PCIF is a function from $C$ to $I$, which can be denoted by $f : C \rightarrow I$. Set $C$ is the domain, and $I$ is the codomain. For instance, if $C = \{c_1, c_2, c_3\}$, then a PCIF may specify PCIs $pci(c_1, c_2) = umd$, $pci(c_1, c_3) = umd$, and $pci(c_2, c_3) = ref$. 
2.5.4 Specify A CPT with A NIN-AND Tree Model

NIN-AND causal modeling allows to specify a CPT with fewer parameters (norm-
ally SCPs) after a NIN-AND tree is specified. We hereby give the definition of
NIN-AND tree models. A NIN-AND tree model $M$ consists of a NIN-AND tree $T$
and a set of SCPs, denoted by $P$. Hence, a NIN-AND causal model is denoted by
$M = (T, P)$. After obtaining $M$, we can specify a CPT based on $M$.

Suppose that the NIN-AND tree shown in Figure 2.9 represents the structure of
the problem. The SCPs are given in Table 2.10. Then, we can obtain all unknown
parameters of the CPT with the following steps.

Table 2.10: The SCPs of the NIN-AND tree shown in Figure 2.9

<table>
<thead>
<tr>
<th>$c_1$</th>
<th>$c_2$</th>
<th>$c_3$</th>
<th>$P(e^+ \leftarrow c_1, c_2, c_3)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>F</td>
<td>F</td>
<td>F</td>
<td>-</td>
</tr>
<tr>
<td>F</td>
<td>F</td>
<td>T</td>
<td>0.9</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>F</td>
<td>0.8</td>
</tr>
<tr>
<td>F</td>
<td>T</td>
<td>T</td>
<td>-</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>F</td>
<td>0.7</td>
</tr>
<tr>
<td>T</td>
<td>F</td>
<td>T</td>
<td>-</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>F</td>
<td>-</td>
</tr>
<tr>
<td>T</td>
<td>T</td>
<td>T</td>
<td>-</td>
</tr>
</tbody>
</table>

1. Because all causes are modeled, the probability of producing the effect without
any cause being present has to be 0. Hence, the first parameter of the CPT is
assigned with 0.

2. The next unknown parameter is $P(e^+|c_1^-, c_2^+, c_3^+)$. Because $g_1$ is the first
NIN-AND gate whose output causal event associates with both $c_2$ and $c_3$, we
obtain $pci(c_2, c_3) = ref$ based on the type of $g_1$, and calculate $P(e^+ \not\leftarrow c_2^+, c_3^+)$
at $g_1$. Because of the undermining causal interaction encoded by $g_1$, failure conjunction and failure independence can be applied, which yields

$$P(e^+ \not\leftarrow c_2^+, c_3^+) = P(e^+ \not\leftarrow c_2^+)P(e^+ \not\leftarrow c_3^+) = (1 - 0.8)(1 - 0.9) = 0.02.$$ 

Because of the negation link, the right hand side input node of $g_2$ is implicitly labeled with the causal event $e^+ \leftarrow c_2^+ c_3^+$. Hence, we calculate

$$P(e^+ \leftarrow c_2^+ c_3^+) = 1 - 0.02 = 0.98$$

at this input node of $g_2$. Because the other cause of $g_2$ is absent, the causal event of the output node of $g_2$ is equivalent to $e^+ \leftarrow c_2^+, c_3^+$. Hence, parameter $P(e^+|c_1^-, c_2^+, c_3^+) = P(e^+ \leftarrow c_2^+, c_3^+) = 0.98$ is obtained at the output node of $g_2$.

3. For the next unknown parameter $P(e^+|c_1^+, c_2^-, c_3^+)$, only $c_3$ of $g_1$ is present. Hence, the causal event of output node of $g_1$ is equivalent to $e^+ \not\leftarrow c_3^+$, and probability $P(e^+ \not\leftarrow c_3^+) = 1 - 0.9 = 0.1$ is obtained at the output node of $g_1$.

Because of the negation link, the causal event implicitly labeled on the right hand side input node of $g_2$ is $e^+ \leftarrow c_3^+$. Hence, $P(e^+ \leftarrow c_3^+) = 0.9$ is calculated at this input node. Because $c_2$ is absent, the causal event of the output node of $g_2$ is equivalent to $e^+ \leftarrow c_1^+, c_3^+$. Then, success conjunction and success independence encoded by $g_2$ can be applied as

$$P(e^+ \leftarrow c_1^+, c_3^+) = P(e^+ \leftarrow c_1^+)P(e^+ \leftarrow c_3^+) = 0.7 * 0.9 = 0.63.$$ 

Finally, we get $P(e^+|c_1^+, c_2^-, c_3^+) = P(e^+ \leftarrow c_1^+, c_3^+) = 0.63$ at the output node of $g_2$. The next unknown parameter $P(e^+ \leftarrow c_1^+, c_2^+)$ can be obtained with the similar manner.
4. The last unknown parameter is $P(e^+|c_1^+, c_2^+, c_3^+)$. Based on the previous steps, we know that the probability calculated at the output node of $g_1$ is $P(e^+ \neq c_2^+, c_3^+) = 0.02$. Then, $P(e^+ \leftarrow c_2^+, c_3^+) = 0.98$ is calculated at the right hand side input node of $g_2$. Because all causes of the causal event $e^+ \leftarrow c_1^+, c_2^+, c_3^+$ of the output node of $g_2$ are present, the causal probability $P(e^+ \leftarrow c_1^+, c_2^+, c_3^+)$ is obtained at the output node with success conjunction and success independence, which is

$$P(e^+ \leftarrow c_1^+, c_2^+, c_3^+ = P(e^+ \leftarrow c_1^+)P(e^+ \leftarrow c_2^+, c_3^+) = 0.7 \times 0.98 = 0.686.$$  

Therefore, we have $P(e^+|c_1^+, c_2^+, c_3^+) = P(e^+ \leftarrow c_1^+, c_2^+, c_3^+) = 0.686$.

Eventually, a CPT is specified and is shown in Table 2.11.

Table 2.11: The CPT of the NIN-AND tree shown in Figure 2.9

| $c_1$ | $c_2$ | $c_3$ | $P(e^+|c_1^+, c_2^+, c_3^+)$ |
|------|------|------|-------------------------------|
| F    | F    | F    | 0                             |
| F    | F    | T    | 0.9                           |
| F    | T    | F    | 0.8                           |
| F    | T    | T    | 0.98                          |
| T    | F    | F    | 0.7                           |
| T    | F    | T    | 0.63                          |
| T    | T    | F    | 0.56                          |
| T    | T    | T    | 0.686                         |

2.5.5 Minimal NIN-AND Trees and Root Labeled Trees

We may get the same CPT by using a set of SCPs with different NIN-AND trees. In this case, the NIN-AND trees are equivalent to each other. Figure 2.10 shows two different NIN-AND trees. Suppose that we have a set of SCPs
\[ P(e^+ \leftarrow c_1^+) = 0.5, P(e^+ \leftarrow c_2^+) = 0.6 \text{ and } P(e^+ \leftarrow c_3^+) = 0.8. \] We denote this set of SCPs by \( P \). The NIN-AND tree shown in the graph (a) is denoted by \( T \), and the NIN-AND tree shown in the graph (b) is denoted by \( T' \). Therefore, we have two NIN-AND tree models \( M = (T, P) \) and \( M = (T', P) \). Using these two NIN-AND tree models, we get the same CPT (shown in Table 2.12). Hence, we say \( T \) is equivalent to \( T' \).

It is not desirable to have two equivalent NIN-AND trees, and we want to keep the number of NIN-AND trees given \( n \) causes as small as possible. Therefore, the concept of minimal NIN-AND trees is introduced. The NIN-AND trees in the last example are associated with the same PCIF, which specifies \( pci(c_1, c_2) = umd, pci(c_1, c_3) = umd \), and \( pci(c_2, c_3) = umd \). In fact, equivalent NIN-AND trees are always associated with
Table 2.12: The CPT of the NIN-AND tree shown in Figure 2.9

| $c_1$ | $c_2$ | $c_3$ | $P(e^+|c_1, c_2, c_3)$ |
|-------|-------|-------|-----------------------|
| F     | F     | F     | 0                     |
| F     | F     | T     | 0.8                   |
| F     | T     | F     | 0.6                   |
| F     | T     | T     | 0.48                  |
| T     | F     | F     | 0.5                   |
| T     | F     | T     | 0.4                   |
| T     | T     | F     | 0.3                   |
| T     | T     | T     | 0.24                  |

the same PCIF. Hence, two NIN-AND trees given $n$ causes should be associated with two different PCIFs. This kind of NIN-AND trees are called minimal NIN-AND trees [18]. To ensure that a NIN-AND tree $T$ is a minimal NIN-AND tree, each NIN-AND gate of $T$ can only connect to the different type NIN-AND gate [18]. For example, if $T$ is a minimal NIN-AND tree, the NIN-AND gates of $T$ connecting with a dual NIN-AND gate $g_1$ of $T$ must be direct NIN-AND gates. In the last example, the NIN-AND tree shown in graph (b) is a minimal NIN-AND tree with one NIN-AND gate.

Given this property of minimal NIN-AND trees, it can be concluded that if the type of leaf NIN-AND gate of $T$ is labeled, the types of all other NIN-AND gates of $T$ are also determined [17]. This implies that two minimal NIN-AND trees can share the same tree structure but have different types of labels. In Figure 2.11, graphs (a)(b)(c) are three NIN-AND trees with the same structure. The NIN-AND tree $T$ shown in graph (a) and the NIN-AND tree $T'$ shown in graph (b) are two minimal NIN-AND trees with different types of leaf gates, which also determines that the types of all the NIN-AND Gates of $T$ are different from the types of $T'$s. To further
Figure 2.11: (a) a minimal NIN-AND tree with dual leaf gate; (b) a minimal NIN-AND tree with direct leaf gate; (c) a root labeled tree.

To simplify the NIN-AND tree representation, we can leave the leaf NIN-AND gate of a minimal NIN-AND tree unlabeled, and only label the associated causes of each root. Then, such a minimal NIN-AND tree is called a root labeled tree [18]. The NIN-AND tree shown in graph (c) is a root labeled tree. The number of root labeled tree is the half of the number of minimal NIN-AND trees [18]. The number of root labeled trees increases exponentially on the number of causes. Table 2.13 shows the number of root labeled trees with 3 to 9 causes [17].

<table>
<thead>
<tr>
<th>#causes</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>#root labeled trees</td>
<td>4</td>
<td>26</td>
<td>236</td>
<td>2752</td>
<td>39208</td>
<td>660032</td>
<td>12818912</td>
</tr>
</tbody>
</table>

Table 2.13: Number of root labeled trees
2.6 Leak Cause

The conditional probabilities given that all causes are absent, $P(e^+ \leftarrow C^-)$, are larger than 0 in many real-world CPTs. This is because that some causes are not modeled in the CPT, and these causes still have the impact on the effect. There are two main causes of headache, bad diet and stress. In addition, having iced water in winter may cause headache, but the probability $P(headache^+ \leftarrow icedwater^+)$ is very small. Hence, cause icedwater is normally not modeled in the CPT of headache. Besides having iced water, headache still have many other causes, where each cause has a non-significant impact on the effect (for example, awkward sleeping posture may cause stiff neck and headache). As a result, even though the impact of such a cause is very small, the joint impact of all these causes may not be that small. The first parameter of a CPT $P(headache^+ \leftarrow C^-)$ is the impact of all un-modeled causes on headache. As a result, it is significantly larger than 0.

However, the causal models assume that all causes are modeled ($P(e^+ \leftarrow C^-) = 0$). To address this conflict, scientists propose leak cause, denoted by $c_L$. A leak cause represents all un-modeled causes, and we call a leak cause as ”other”. A leak cause is always present, and can fail to produce the effect [20]. We can add a cause node (called leak node) in the graph to represent the leak cause (Figure 2.12 shows headache and its causes including leak cause). The probability associated with the leak cause is called leak probability, which measures the impact of all un-modeled causes on the effect. After introducing leak cause, all causes of the CPT are modeled, which ensures $P(e^+ \leftarrow C^-) = 0$. 


Henrion (1989) proposed a method to specify a CPT with the leak cause [5]. (This process is also called \textit{parameterization}.) Díez (1993) [1] proposed another parameterization method with leak cause, which is arithmetically equivalent with Henrion’s method [20].
Chapter 3

Approximating CPTs with NIN-AND Tree Models

As we described in the previous sections, NIN-AND tree causal modeling can specify a CPT with fewer parameters, whose number is linear on the number of causes. Because a general CPT’s size is exponential on the number of causes, using NIN-AND tree causal modeling can improve space efficiency significantly. In addition, NIN-AND tree causal modeling can speed up inference [15]. Instead of keeping a complete CPT, experts may want to keep a NIN-AND tree model (a set of SCPs and a NIN-AND tree) because of these two reasons. However, the CPT specified with a NIN-AND tree model may not be exactly the same as the original CPT (target CPT) because the original CPT may not strictly follow the assumptions of NIN-AND tree causal modeling. In such cases, error will to be introduced. To minimize the error, we develop the techniques to optimally approximate a target CPT with a NIN-AND tree model.

In this chapter, we discuss the methodologies of approximating an arbitrary CPT with a NIN-AND tree model. In Section 3.1, problems of CPT approximation are analyzed and the approximation process is outlined. Section 3.2 introduces the distance measurement method. Section 3.3 discusses the methods for indexing minimal NIN-AND tree, and search tree construction. Section 3.4 shows the algorithms of search
key extraction and scoring, and NIN-AND trees search using the search tree. Section 3.5 presents the numerical methods for optimally approximating a target CPT given a NIN-AND tree.

### 3.1 Problem Analysis and Approximation Process

In this section, the problems which have to be considered during the process of CPT approximation are analyzed. After analyzing the problems, the process of CPT approximation is outlined and illustrated.

The goal of CPT approximation is to find a NIN-AND tree model which can be used to specify a CPT $\text{cpt}'$ being close to the target CPT $\text{cpt}$. Whether $\text{cpt}'$ is close to $\text{cpt}$ or not is measured by a distance function, denoted by $d(\text{cpt}, \text{cpt}')$. The smaller the value of $d$, the better the approximation accuracy. Hence, a problem here is to find a distance function.

A $n$-cause target CPT should be approximated with a NIN-AND tree model $M = (T, P)$, where $T$ is a minimal NIN-AND tree being associated with $n$ causes, and $P$ is a set of SCPs, $P = \{x_1, ..., x_n\}$, where $x_i$ is the SCP for cause $c_i$. Alternatively, we can represent $P$ as a vector $P = (x_1, ..., x_n)$ for the vector of causes $C = (c_1, ..., c_n)$. Without confusion, we will switch between the set and vector representations of $P$ freely. Based on the algorithm shown in Section 2.5.4, we can say that $\text{cpt}'$ is specified by a function $t(P)$, where the function $t$ is determined by the structure of the NIN-AND tree $T$. This is shown by Figure 3.1.

Because $\text{cpt}'$ of $d(\text{cpt}, \text{cpt}')$ is obtained by a function $t(P)$, $d$ is defined only if $t$ is
defined, and then \( d \) is a function about \( P \) and \( \text{cpt} \). Once \( d \) is defined, many numerical methods can be used to minimize the distance function, and find the corresponding values of \( P \). A problem here is to find a numerical method to minimize \( d \) and obtain \( P \) when \( t \) is defined, namely when the NIN-AND tree \( T \) is given.

Once we can minimize \( d \) with a given \( T \), we can optimally minimize distance between \( \text{cpt} \) and \( \text{cpt}' \). Firstly, we enumerate all possible minimal NIN-AND trees, and define corresponding distance functions. Secondly, based on each distance function \( d \), the minimum of \( d \) and the corresponding \( P \) can be obtained by the numerical method. The last step is to compare all obtained minimums of distance functions, and keep the smallest one and its corresponding \( T \) and \( P \). A NIN-AND tree model \( M = (T, P) \) with optimal approximation is then obtained.

The number of minimal NIN-AND trees increases exponentially on the number of causes \( n \) (shown in Table 2.13). This means that enumerating all possible minimal NIN-AND trees may not be practical when \( n \) is big. To address this issue, we need to generate a small subset of candidate NIN-AND trees efficiently and focus the distance-based evaluation only on this subset. To achieve this, we index minimal NIN-AND
trees, and build search trees based on these indexes. In addition, how to extract search key based on \( cpt \) has to be considered carefully.

To sum up, we develop the techniques to solve the problems analyzed above. The flowchart of the approximation process is shown in Figure 3.2.

---

**Figure 3.2: Framework of approximation process**
3.2 Distance Measurement

In this section, we discuss the distance measurement method. In the easiest way, we can define a distance between two CPTs as a sum of distances of corresponding probability distributions (parameters) in both CPTs [19], which means that Euclidean distance (ED) can be used to measure the approximation result. Suppose that we have a target CPT \( \text{cpt} \) and another CPT \( \text{cpt}' \) as the approximation of \( \text{cpt} \). A parameter of \( \text{cpt} \) is denoted by \( p_i \) (\( i \) starts from 0), and a parameter of \( \text{cpt}' \) is denoted by \( p_i' \). If \( \text{cpt} \) has \( n \) causes (then \( \text{cpt} \) has \( 2^n \) parameters), ED between \( \text{cpt} \) and \( \text{cpt}' \) is a function denoted by \( d_{ED}(\text{cpt}, \text{cpt}') \) or simply \( d_{ED} \). Then, the ED function between two CPTs is defined as

\[
d_{ED} = \sqrt{\sum_{i=0}^{2^n-1} (p_i - p_i')^2}.
\]  

(3.1)

Because the first parameter of \( \text{cpt}' \) has to be 0, the ED function can also be defined as the following form.

\[
d_{ED} = \sqrt{p_0^2 + \sum_{i=1}^{2^n-1} (p_i - p_i')^2}
\]  

(3.2)

We believe that a distance is more meaningful when it represents the average distance between a parameter of \( \text{cpt} \) and a parameter of \( \text{cpt}' \). Hence, we decide to use weighted Euclidean distance (WED) or uniformly weighted Euclidean distance (because the weights are uniform, and there is no standard terms for such a distance function, we may also call it average distance) to measure the distance between CPTs. We denote a WED function by \( d(\text{cpt}, \text{cpt}') \) or \( d \). The function of \( d(\text{cpt}, \text{cpt}') \) is defined
as the equation below.

\[
d(cpt, cpt') = \sqrt{\frac{1}{2^n}p_0^2 + \sum_{i=1}^{2^n-1} \frac{1}{2^n}(p_i - p'_i)^2}.
\] (3.3)

In this equation, the distance between each pair of parameters is weighted with \(1/(2^n)\). Because \(2^n\) is the total number of parameters in a CPT, \(d\) represents the average distance between a parameter \(p\) of \(cpt\) and a parameter \(p'\) of \(cpt'\). In the rest of the thesis, we use Equation (3.3) to measure the distance between two CPTs.

### 3.3 Minimal NIN-AND Tree Indexing and BST Construction

As we explained, the number of minimal NIN-AND trees increases exponentially on the number of causes. It is not practical to go through all minimal NIN-AND trees when the number of causes is big. To deal with this issue, we want to directly select a subset of minimal NIN-AND trees fitting targets CPTs without enumerating all minimal NIN-AND trees. For this purpose, we decide to build search trees to search quality minimal NIN-AND trees. To build a search tree, we want to index minimal NIN-AND trees, and let each leaf node of a search key be associated with the index of a minimal NIN-AND tree.

In this section, the methods for indexing minimal NIN-AND trees, and for constructing binary search trees (BST) based on the indexes of NIN-AND trees are discussed.
3.3.1 Minimal NIN-AND Tree Indexing

Indexing With PCIF

There are two expectations for the indexing method,

1. There is an one-to-one mapping between each NIN-AND tree and an index.

2. An index represents the features of the corresponding NIN-AND tree.

To achieve these two expectations, pairwise causal interaction function is used to index NIN-AND trees. As we explained in the previous sections, a minimal NIN-AND tree is associated with a \( PCIF \), and vice versa. This property of PCIF satisfies the first requirement. In addition, a PCIF specifies the causal interaction, encoded by the NIN-AND tree, between each pair of causes. These causal interactions reflect the features of the NIN-AND tree. Therefore, the second requirement can be satisfied if we use PCIF to index a NIN-AND tree.

To make a PCIF as a representable index, we use the following method. We use 0 to represent undermining causal interaction, and 1 to denote reinforcing causal interaction. Then, \( pci(c_i, c_j) = udm \) is equivalent to \( pci(c_i, c_j) = 0 \) and \( pci(c_i, c_j) = ref \) is equivalent to \( pci(c_i, c_j) = 1 \). The values of PCIs specified by the PCIF are equivalent to some binary digits. Suppose that we have a 4-cause NIN-AND tree. It is associated with a PCIF which specifies \( pci(c_1, c_4) = udm, pci(c_2, c_4) = ref, pci(c_3, c_4) = udm, pci(c_1, c_3) = ref, pci(c_2, c_3) = udm, \) and \( pci(c_1, c_2) = ref \). After substituting \( udm \) by 0 and \( ref \) by 1, we have \( pci(c_1, c_4) = 0, pci(c_2, c_4) = 1, pci(c_3, c_4) = 0, pci(c_1, c_3) = 1, pci(c_2, c_3) = 0, \) and \( pci(c_1, c_2) = 1 \). If we set up a rule to index each PCI, these binary
digits can be represented as a vector consisting of binary digits. Then, each PCIF is associated with such a vector. In our research, we set up the following rules to index PCIs.

1. The first PCI $pci(c_i, c_j)$ is $pci(c_1, c_2)$.

2. The PCI after $pci(c_i, c_j)$ is $pci(c_i, c_{j+1})$ if $j < n$, where $n$ is the total number of causes.

3. The PCI after $pci(c_i, c_j)$ is $pci(c_{i+1}, c_{i+2})$ if $i < n - 1$ and $j = n$.

4. The last PCI is $pci(c_{n-1}, c_n)$.

Based on these rules, the PCIs are ordered as $pci(c_1, c_2) = 1$, $pci(c_1, c_3) = 1$, $pci(c_1, c_4) = 0$, $pci(c_2, c_3) = 0$, $pci(c_2, c_4) = 1$, and $pci(c_3, c_4) = 0$, which can be represented as a vector $(1, 1, 0, 0, 1, 0)$, where the $i$th element is the value of the $i$th PCI $pci(c_m, c_n)$. Then, we can use 110010 as the index of the NIN-AND tree. (when the number of causes $n$ is big, it may be inconvenient to have $n(n - 2)/2$ binary digits as the index. In this case, we can convert these binary digits into hexadecimal or even higher base.)

After figuring out the method of indexing a NIN-AND tree, another thing which has to be considered is the storage space. Basically, we want to store root labeled trees instead of minimal NIN-AND trees. As we discussed in the section of minimal NIN-AND trees, the only difference between two minimal NIN-AND trees sharing the same tree structure is the types of their leaf gates. Hence, by storing root labeled trees, we can save 50% space.
In our research, we use root labeled trees to represent the minimal NIN-AND trees having the same structure. However, root labeled trees are not associated with PCIFs because no NIN-AND gates are specified and the causal interactions are unknown. Hence, a root labeled tree cannot be indexed. To address this problem, we assume that a root labeled tree has a direct leaf gate when we index it. (For example, when we index the root labeled trees shown in Figure 2.11 (c), we assume that it is equivalent to the direct leaf gate NIN-AND tree shown in Figure 2.11 (b).)

**Algorithms of Minimal NIN-AND Tree Indexing**

The algorithms (Algorithm 1 and Algorithm 2) are proposed to index a given root labeled tree.

Function $\text{GetPCI}(\text{RLT, Gate, CI, } c_1, c_2)$ is a recursive function shown in Algorithm 2. The argument $\text{Gate}$ points to a NIN-AND gate node. The NIN-AND gate node stores a label of its output causal event $\text{OutEvent}$ and a pointer to the parent $p_i$ of the gate node. The argument $\text{CI}$ is the type of the causal interaction encoded by $\text{Gate}$ (0 indicates undermining, 1 indicates reinforcement).

This function is used to get the value of $\text{pci}(c_i, c_j)$. It can return 3 possible values, 0, 1 or $\text{Not found}$, where $\text{udm} = 0$, $\text{ref} = 1$, and $\text{NotFound}$ indicates that the output causal event $\text{OutEvent}$ of $\text{Gate}$ does not contain both $c_i$ and $c_j$. As explained in Section 2.5.3, $\text{pci}(c_i, c_j)$ is determined by the first NIN-AND gate whose output causal event contains both causes. Hence, the objective of the function is to find this gate. The function always calls the leaf gate node first (Shown in Algorithm 1). When a gate node is called by the function, each parent of this gate node will be
Algorithm 1 Indexing a root labeled tree

Require: $RLT$: a root labeled tree; $Leaf$: the leaf gate node of $RLT$;
$n$: the number of causes;

Ensure: $Index$: array of PCIs of $RLT$;

$k=0$;
create $Index$ as an array of $n(n-1)/2$ elements;

for $i=1$ to $n-1$ do
    for $j=i+1$ to $n$ do
        $Index[k] = GetPCI(RLT, Leaf, 0, ”c_i”, ”c_j”); //Algorithm 2$
        $k=k+1$;
    end for
end for

return $Index$.

called to continue the recursion. The function has four base cases (stop criterion).
The first base case is that $OutEvent$ of the current gate node $Gate$ does not contain both $c_i$ and $c_j$, which also implies that $OutEvent$ of $Gate$’s parents cannot contain both causes. Then, $Notfound$ is returned (The output causal event of a leaf gate node contains all causes, which guarantees that $pci(c_i, c_j)$ can be valued, and $Notfound$ will not be returned as the final result.). The second base case is that $OutEvent$ of the current gate node contains both causes, and the gate node has no parents, which implies that the current gate node is the first gate containing both causes. Hence, the type of causal interaction encoded by $Gate$ is returned. The third base case is that a parent of the current gate node returns $m$ ($m \neq Notfound$), which
Algorithm 2 $GetPCI(RLT, Gate, CI, c_1, c_2)$

if $Gate.OutEvent$ does not contain $c_1$ and $c_2$ then return $NotFound$;

if $Gate$ do not have parent gates then then return $CI$; // $Gate$ encodes $pci(c_1, c_2)$

else

for each $p_i$ of $Gate$ do

$CITemp = GetPCI(RLT, p_i, 1 - CI, c_1, c_2);$;

if $CITemp \neq NotFound$ then // an ancestor of $Gate$ encodes $pci(c_1, c_2)$

return $CITemp$;

end for

return $CI$; // $Gate$ encodes $pci(c_1, c_2)$

end if

implies $pci(c_i, c_j) = m$ is encoded by an ancestor of the current gate node. Then, $m$ is returned by the current gate node. The last based case is that $OutEvent$ of the current gate node contains both causes, and all parents of the current gate node return $NotFound$, which implies that the current gate node encodes $pci(c_i, c_j)$. The type of the causal interaction encoded by the current gate node is returned.

Shown in Algorithm 1, when the leaf gate node is called, the argument $CI$ is given as 0, which implies that the leaf gate is assumed as a direct leaf gate. The question is that when the parent gate node of the leaf gate node is called, what the value of argument $CI$ should be. Base on the property of minimal NIN-AND trees, if a NIN-AND gate $Gate_1$ connected to another NIN-AND gate $Gate_2$, $Gate_1$ and $Gate_2$ are two different types of NIN-AND gates. Hence, the causal interaction encoded by $Gate$’s parent gate is different from the causal interaction encoded by $Gate$. Suppose
that \( CI \) of Gate is \( m \) \((udm = 0, \ ref = 1)\). Then, \( CI \) of Gate’s parent gate node equals to \( 1 - m \). Similarly, in our problem, \( CI \) of the leaf gate node’s parent gate node is \( 1 - 0 = 1 \).

An Example of Minimal NIN-AND Tree Indexing

Let us look at an example. Suppose that a root labeled tree is given in Figure 3.3 (a). To index this root labeled tree, we assume that the type of its leaf gate is direct. Then, it is equivalent to the NIN-AND tree \( T \) shown in Figure 3.3 (b) when we index it. Because \( T \) is a 3-cause NIN-AND tree, the number of PCIs is \( 3 \times (3 - 1)/2 = 3 \), which means that the index will contain 3 digits.

The first PCI of \( T \) should be specified is \( pci(c_1, c_2) \). Based on the algorithms, we first look at the leaf gate \( g'_2 \). In the causal event labeled on it, both \( c_1 \) and \( c_2 \) are contained. If it does not have parent gates, it can be immediately determined that the leaf gate encodes \( pci(c_1, c_2) \). However, \( g'_2 \) has a parent gate \( g'_1 \), and hence \( g'_1 \) has to be considered. In the causal event labeled on \( g'_1 \), \( c_1 \) and \( c_2 \) cannot be found together. Hence, it returns \( Notfound \) to \( g'_2 \). Because the only parent gate of \( g'_2 \) returns \( Notfound \) and the causal event labeled on \( g'_2 \) contains both causes, the algorithms conclude that \( pci(c_1, c_2) \) is determined by the causal interaction encoded by \( g'_2 \), which is \( udm \). PCI \( pci(c_1, c_2) = udm \) is obtained. Based on the same idea, the second PCI \( pci(c_1, c_3) = udm \) and third PCI \( pci(c_2, c_3) = ref \) can be determined. Finally, the index of the root labeled tree shown in Figure 3.3 (a) is obtained as 001.

After indexing all minimal NIN-AND trees, they are stored using the indexes
Figure 3.3: A root labeled tree and its related NIN-AND tree

as file names. For example, a minimal NIN-AND tree is indexes as 010, then this minimal NIN-AND tree file is stored in the disk with file name 000.rlt.
3.3.2 BST Construction

Based on the indexes of minimal NIN-AND trees, we decide to use binary search trees (BST) as search trees for selecting NIN-AND trees. CPTs with \( n \) causes are approximated using \( n \) causes NIN-AND trees. Hence, for each \( n \), we create a BST. The idea of 3-cause BST construction is shown in Figure 3.4.

![BST construction diagram](image)

**Figure 3.4: The BST construction with 3 causes**

A BST has the following characteristics.

1. The number of layers of a BST is \( N + 1 \), where \( N \) is the number of PCIs specified
by a PCI function. (If the number of causes is \( n \), \( N \) equals to \( n \times (n - 1)/2 \).)

2. Each node at depth \( m \) (\( m \geq 1 \)) is labeled by the value of the \( m_{th} \) PCI. (root node is at depth 0, and is not labeled.)

3. In particular, each non-leaf node at depth \( m - 1 \) has 2 children at most. The left child of this node is labeled with \( pci_{,m} = 0 \), which indicates that the value of \( m_{th} \) PCI equals to 0. The right child of this node is labeled with \( pci_{,m} = 1 \), which indicates that the value of \( m_{th} \) PCI is 1.

4. A leaf node may not be associated with a root labeled tree. A 6-cause NIN-AND tree has \( 6 \times (6 - 1)/2 = 15 \) PCIs, which means that the 6-cause BST can have \( 2^{15} = 32768 \) leaf nodes. However, there are only 2752 root labeled trees for 6 causes. In this case, most leaf nodes of the BST are not related to root labeled trees. To save storage and improve efficiency, we delete these leaf nodes. As a result, we also delete related links and all non-leaf nodes which have no children. The 3-cause BST deleted unnecessary nodes and links is shown in Figure 3.5.

5. Each node is also labeled with an index pair \((i_1, i_2)\), which bounds indexes of root labeled tree files being associated with the subtree. For example, in Figure 3.4, the index pair of the left child of root is \((0, 2)\), which indicates that from the \(0_{th}\) root labeled tree file to the \(2_{th}\) root labeled tree file, all root labeled tree files are under this node. A special case is that the elements of the index pair of a leaf node are always the same, because a leaf node can only be associated with one root labeled tree file.
A leaf node of a BST is associated with a root labeled tree file. If the leaf node is reached during search, then the corresponding root labeled tree is selected. Suppose that PCIs of a NIN-AND tree is given by $pci(c_1, c_2) = 0$, $pci(c_1, c_3) = 0$ and $pci(c_2, c_3) = 1$, and we want to search the NIN-AND tree with the BST. By following the PCI sequence 001 from the root, we arrive at the leaf associated with root labeled tree with index 1. Because we index root labeled trees by assuming direct leaf gates, the NIN-AND tree searched is a direct leaf gate NIN-AND tree.

Figure 3.5: The 3-cause BST
3.4 Search Key Extraction and Search Using A BST

In the last section, we describe the method for indexing all root labeled trees based on corresponding PCIFs, and constructing BSTs as search trees. The next step is to extract search key, and using the search key and the BST to select NIN-AND trees which fit the target CPT. In this section, we discuss the techniques for dealing these two problems.

3.4.1 Search Key Extraction and Scoring

In this subsection, the algorithms of search key extraction and scoring are discussed. We do not consider leaky probability in this research by assuming that all causes are explicitly modeled. There are two effects of this assumption. One is that the first parameter of the CPT $cpt'$ specified with a NIN-AND tree model is always 0. (The first parameter of a CPT is the probability of producing the effect when all modeled causes are absent, which is denoted by $P(e^+ \leftarrow C^-)$.) The other effect is that we will not add a leak cause node to represent un-modeled causes. This means that the process of specifying a CPT with a NIN-AND tree model exactly follows the steps shown in the example of Section 2.5.4.

In our algorithm, a search key can either be represented by a set consisting of a number of pairs. Suppose that the target CPT $cpt$ has 3 causes, $c_1, c_2$ and $c_3$. Then the PCIF of $cpt$ specifies a vector of PCIs $(pci(c_1, c_2) = 0, pci(c_1, c_3) = 0, pci(c_2, c_3) = 1)$, which are ordered based on the rules described in Section 3.3.1. According to each PCI, a pair is defined. In this example, the search key is a set consisting of 3 pairs
The first element of the $i_{th}$ pair is the index of the $i_{th}$ PCI. For example, the element 1 of the first pair (1, 0) is the index of the first PCI $pci(c_1, c_2)$. The second number of $i_{th}$ pair is the value of the $i_{th}$ PCI ($umd = 0$ and $ref = 1$). For example, the element 0 of (1, 0) indicates that the value of the first PCI $pci(c_1, c_2)$ equals to $udm$.

After define a search key, the next problem is to extract search key based on the target CPT $cpt$. According to the definition, if

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

the causal interaction between causal events $e^+ \leftarrow c_1^+$ and $e^+ \leftarrow c_2^+$ is reinforcing. If

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

the causal interaction between causal events $e^+ \leftarrow c_1^+$ and $e^+ \leftarrow c_2^+$ is undermining.

The search key of $cpt$ can be immediately defined based on the definitions if we only considers SCPs and DCPs (double causal probability, which is the probability of producing the effect when two causes are present). For example, the target CPT is given in Table 3.1. Immediately, we have $P(e^+ \leftarrow c_1^+, c_2^+) > Max(P(e^+ \leftarrow c_1^+), P(e^+ \leftarrow c_2^+))$. Hence, the causal interaction between causal events $e^+ \leftarrow c_1$ and $e^+ \leftarrow c_2$ is reinforcing. PCI $pci(c_1, c_2)$ is assigned with $ref$, and the search key is extracted as $\{(1, 1)\}$.

Based on this idea, we propose the algorithm for search key extraction. The algorithm only considers SCPs and DCPs of the target CPT $cpt$, and obtain the search key based on the definitions. In particular, we look at each DCP of $cpt$, and find its
corresponding SCPs. Suppose that the DCP we are looking at is \( P(e^+ \leftarrow c_1^+, c_j^+) = q \). Then, we find its related SCPs, \( P(e^+ \leftarrow c_i^+) = p_i \) and \( P(e^+ \leftarrow c_j^+) = p_j \).

In this thesis, the objective is to approximate target CPTs with NIN-AND tree models. For an arbitrary target CPT, a pair of causes may not satisfy the reinforcing and undermining causal interactions as defined in Section 2.5.1. To have a good NIN-AND tree approximation, if a pair of causes is almost reinforcing, then we approximate their relation by reinforcing. If their relation is almost undermining, we approximate the relation by undermining. More specifically, if

\[
P(e^+ \leftarrow c_1^+ c_2^+) \geq \frac{P(e^+ \leftarrow c_1^+) + P(e^+ \leftarrow c_2^+)}{2},
\]

then we set \( pci(c_1, c_2) = ref \). Similarly, if

\[
P(e^+ \leftarrow c_1^+ c_2^+) < \frac{P(e^+ \leftarrow c_1^+) + P(e^+ \leftarrow c_2^+)}{2},
\]

we set \( pci(c_1, c_2) = udm \).

In the algorithm, we do not only extract PCIs, but also score each PCI. Score can be used to measure how close the interaction between a pair of causes is to the well-defined reinforcing and undermining. The score of each PCI is the 1 norm distance between the DCP and the average of the related SCPs, and is denoted by \( s(c_i, c_j) \).
Suppose that we have two SCPs \( p_1 = P(e^+ \leftarrow c^+_1) \) and \( p_2 = P(e^+ \leftarrow c^+_2) \), and the corresponding DCP \( q = P(e^+ \leftarrow c^+_1, c^+_2) \). The score of PCI \( pci(c_1, c_2) \) is defined as

\[
 s(c_1, c_2) = |q - \frac{p_1 + p_2}{2}|.
\]

For example, the two SCPs are valued as \( P(e^+ \leftarrow c^+_1) = 0.3 \) and \( P(e^+ \leftarrow c^+_2) = 0.6 \). If \( q = 0.7 \), \( pci(c_1, c_2) \) is scored as \( 0.7 - (0.6 + 0.3)/2 = 0.25 \). Because \( 0.7 > Max(0.6, 0.3) \), \( pci(c_1, c_2) = ref \) is well-defined. Now assume that \( q = 0.55 \). Because 0.55 is between 0.6 and 0.3, we can use the method in the last paragraph to specify \( pci(c_1, c_2) = ref \ (0.55 > (0.6 + 0.3)/2) \). Then, \( pci(c_1, c_2) \) is scored as \( 0.55 - (0.6 + 0.3)/2 = 0.1 \), which is smaller than 0.25. If \( q = 0.5 \), \( pci(c_1, c_2) \) is specified as \( ref \) based on the same idea. It is scored as \( 0.5 - (0.6 + 0.3)/2 = 0.05 \). When \( q = 0.7 \), the PCI is well-defined, and has the highest score 0.25. For the case \( q = 0.55 \), \( q \) is more close to \( Max(0.3, 0.6) \) compared with \( q = 0.5 \). This is reflected in the scores, \( 0.1 > 0.05 \).

The algorithm of search key extraction and scoring is shown in Algorithm 3.
**Algorithm 3** Search key extraction and scoring with SCPs and DCPs

**Require:** \( cpt \): the target CPT; \( n \): the number of causes;

**Ensure:** \( PCIS \): a set consisting of the value of each PCI of \( cpt \);
\( S \): a set consisting of the score of each PCI \( s(c_i, c_j) \);

\[
PCIS = \emptyset, \quad S = \emptyset;
\]

for \( i = 1 \) to \( n - 1 \) do

for \( j = i + 1 \) to \( n \) do

Find the corresponding DCP \( q = P(e^+ \leftarrow c_i^+, c_j^+) \);

Find the related SCPs \( p_i = P(e^+ \leftarrow c_i^+), p_j = P(e^+ \leftarrow c_j^+) \);

if \( q \geq \frac{p_i + p_j}{2} \) then

\[
pci(c_i, c_j) = 1; \quad // \text{udm=0, ref=1}
\]

else

\[
pci(c_i, c_j) = 0;
\]

end if

\[
PCIS = PCIS \cup \{pci(c_i, c_j)\};
\]

\[
s(c_i, c_j) = |q - \frac{p_i + p_j}{2}|;
\]

\[
S = S \cup \{s(c_i, c_j)\};
\]

end for

end for

return the pair \((PCIS, S)\).
Based on this algorithm, we can get one set consisting of the value of each PCI, and another set containing the score of each PCI (the set of scores is called a score set). Suppose that the values of PCIs are extracted as $pci(c_1, c_2) = 0$, $pci(c_1, c_3) = 0$, and $pci(c_2, c_3) = 1$. Then the search key is $\{(1, 0), (2, 0), (3, 1)\}$. Let us look at an example with Algorithm 3. Suppose that we have a target CPT given in Table 3.2. We can extract the search key with the following steps.

| $c_1$ | $c_2$ | $c_3$ | $P(e^+|c_1, c_2, c_3)$ |
|-------|-------|-------|------------------------|
| F     | F     | F     | 0.1                    |
| F     | F     | T     | 0.68                   |
| F     | T     | F     | 0.65                   |
| F     | T     | T     | 0.7                    |
| T     | F     | F     | 0.64                   |
| T     | F     | T     | 0.59                   |
| T     | T     | F     | 0.61                   |
| T     | T     | T     | 0.63                   |

1. Find the first DCP, $P(e^+ \leftarrow c_2^+, c_3^+) = 0.7$. Its related SCPs are $P(e^+ \leftarrow c_2^+) = 0.65$ and $P(e^+ \leftarrow c_3^+) = 0.68$. Because $0.7 > Max(0.65, 0.68)$, we have $pci(c_2, c_3) = ref$ and $s(c_2, c_3) = |0.7 - (0.65 + 0.68)/2| = 0.035$.

2. We find the second DCP, $P(e^+ \leftarrow c_1^+, c_3^+) = 0.59$. Its related SCPs are $P(e^+ \leftarrow c_1^+) = 0.64$ and $P(e^+ \leftarrow c_3^+) = 0.68$. Because $0.59 > Min(0.64, 0.68)$, we have $pci(c_1, c_3) = umd$ and $s(c_1, c_3) = |0.59 - (0.64 + 0.68)/2| = 0.07$. Based on the same idea, $pci(c_1, c_2) = udm$ and $s(c_1, c_2) = 0.035$ can be determined.

3. The search key $\{(1, 0), (2, 0), (3, 1)\}$ is obtained, which implies $pci(c_1, c_2) = umd$, $pci(c_1, c_3) = umd$ and $pci(c_2, c_3) = ref$. (The corresponding minimal NIN-AND
tree is shown in Figure 3.6.) The score set is obtained as \{0.035, 0.07, 0.035\}.

\[ e^+ \not= c_2^+ e^+ \not= c_3^+ \]

\[ e^+ \leftarrow c_1^+ \]

\[ e^+ \not= c_2^+ c_3^+ \]

\[ e^+ \leftarrow c_1^+ c_2^+ c_3^+ \]

Figure 3.6: The NIN-AND tree selected

3.4.2 Search NIN-AND Trees Using A BST

In this subsection, we discuss the algorithm of searching NIN-AND trees with the extracted search key and the BST.

Introducing Time Constraint

Using the search key extraction algorithm, a search key containing the value of each PCI can be obtained. With such a search key, at most one NIN-AND tree would be retrieved. However, we may not want to use all extracted PCIs to search. This is because that if we use less extracted PCIs to search, more NIN-AND trees will be selected. Using more NIN-AND trees for approximation is more likely to have good approximation accuracy. Hence, we may want to use the PCIs having the highest scores to search. (This is also the purpose that we score each PCI during the search
key extraction.) Hereby, we define that a search key containing all PCIs is also called a *full search key*. If we use a subset of PCIs to search, they form a *partial search key*.

We develop an algorithm to allow a time constraint $TC$ to be specified by the user. The algorithm decides the maximum number of PCIs that can be used for search. Thus, we keep as many NIN-AND trees as possible and get the best approximating accuracy under the time constraint.

The basic idea of this algorithm is based on the property of a BST. A BST is a binary search tree, where a non-leaf node can have up to two children. Suppose that a subtree rooted at a node has $n$ leaf nodes. The number of leaf nodes under each child of this node can be roughly estimated as $n/2$. Hence, each time we go from a node to its child, the number of candidate NIN-AND trees underneath will be estimated as reducing by half. Hence, we use $NT/2^{NPCI}$ to estimate the number of selected NIN-AND trees, where $NT$ is the number of minimal NIN-AND trees, and $NPCI$ is the number of PCIs used for search. With each selected NIN-AND tree $T$ and the target CPT $cpt$, a WED function can be defined, and has to be minimized. Suppose that each minimization takes time $t$. The total approximation time is estimated as $t*NT/2^{NPCI}$. If $t*NT/2^{NPCI} < TC$ and $t*NT/2^{NPCI+1} > TC$, the number of PCI to be used for search is determined as $NPCI$.

To examine this algorithm, we set up a simple experiment. We use a 4-cause BFST, and generate every possible partial search keys for 4 causes. Each partial search key is used to search NIN-AND trees, and the number of selected NIN-AND trees $\#ST$ are recorded. For the partial search keys specifying the same number of PCIs, the $\#ST$s obtained with these search key are averaged. The average is reported as
the experiment result. Thus, for each possible \( NPCI \) that \( NPCI \leq 4 \times (4 - 1)/2 = 6 \), an average number of selected NIN-AND tree is obtained. In Figure 3.7, the solid curve shows the average numbers of selected trees obtained in the experiment. The dotted curve shows the estimated numbers of selected trees based on the estimating method above. We can find that the two curves are very close to each other, which means that using this method to estimate the number of selected trees is reasonable. (However, when the number of causes grows, this method may not work well. In this case, we can do experiments similar to the one described above, and use the experimental results to estimate the number of NIN-AND trees that can be selected.)

![Graph](image)

Figure 3.7: Solid curve: experimental results; Dotted curve: estimated results.

The algorithm performs the above task is given in Algorithm 4.
Algorithm 4 Determining the number of PCIs used for search

Require: $TC$; $t$; $NT$; $n$;

Ensure: $NPCI$;

$$Max = \text{Ceil}(TC/t); \quad \text{//the maximum number of NIN-AND trees can be used}$$

$$NPCI = n(n - 1)/2;$$

while true do

$$estimate = NT/2^{NPCI}; \quad \text{//estimated number of selected NIN-AND trees}$$

if $estimate <= Max$ then break;

$NPCI --$;

end while

return $NPCI$;

For example, the $TC$ is given as 5000 milliseconds, and $t = 100$. The target CPT has 5 causes, which means that the number of minimal NIN-AND trees is 472. Based on the algorithm, we find $472/2^3 < 5000/100$ and $472/2^4 > 5000/100$. Therefore, the number of PCIs to be used is 3.
Search Algorithms

If the number of PCIs to be used for search is \( NPCI = 2 \), we form a partial search key consists of 2 PCIs with the highest scores. Suppose that the search key obtained is \{ \( (1, p_1), (2, p_2), (3, p_3) \) \}, and the score set is \( (w_1, w_2, w_3) \), where \( w_1 > w_3 > w_2 \). The partial search key should consist of two PCIs having the highest scores, and is defined as \{ \( (1, p_1), (3, p_3) \) \}. Then, we can start search the NIN-AND trees based on this partial search key.

In a BST, nodes at depth \( d \) represents the \( d \)th PCI \( pci_d \) of the PCIF. Suppose that the partial search key is extracted as \{ \( (i, 0) \) \}. If \( pci_m \) is not specified, we should follow each child of nodes at depth \( m - 1 \) of the BST. In this example, the PCIs before \( pci_i \) are not specified, and hence we go through all the nodes at depth smaller or equal to \( i - 1 \). Because the value of \( pci_i \) is specified as 0, we follow the left child for all nodes at depth \( i - 1 \), and reach the nodes at depth \( i \). Because there is no more specified PCIs, all the nodes at depth \( i \) are selected. Then, we stop search, and collect all index pairs of selected nodes, which contain the indexes of selected root labeled trees. The BST is shown in Figure 3.8. The root-labeled trees, whose indexes are contained in the indexes pairs \( (0, i_1), (i_2 + 1, i_3), ..., (i_{n-2} + 1, i_{n-1}) \), are selected.

Another issue is that the root labeled trees are indexed by assuming that they have direct leaf gates. Hence, if the extracted search key actually refers to a NIN-AND tree with a dual leaf gate, we cannot directly find the root labeled tree. This issue can be shown with the following example.

Suppose that we want to search two NIN-AND trees sharing the same tree struc-
Figure 3.8: A simple example of NIN-AND tree search

ture but having different types of leaf gates. The NIN-AND trees are shown in Figure 3.9. These two trees have the same tree structure but different types of leaf gates. The PCIs of the direct leaf gate NIN-AND tree shown in Figure 3.9 (a) is $pci(c_1, c_2) = 0$, $pci(c_1, c_3) = 0$ and $pci(c_2, c_3) = 1$. The PCIs of the dual leaf gate NIN-AND tree shown in Figure 3.9 (b) is $pci(c_1, c_2) = 1$, $pci(c_1, c_3) = 1$ and $pci(c_2, c_3) = 0$.

Figure 3.9: A direct leaf gate NIN-AND tree and a dual leaf gate NIN-AND tree

The first objective here is to select the root labeled tree having the same tree
structure, which is the second root labeled tree (indexed 1) shown in Figure 3.10. If we use the full search key consisting of PCIs of the direct leaf gate NIN-AND tree \{(1, 0), (2, 0), (3, 1)\} to search, the second root labeled tree will be selected along the path drawn on Figure 3.10. However, if we use the full search key based on the dual leaf gate NIN-AND tree \{(1, 1), (2, 1), (3, 0)\} to search, it will not find the second root labeled tree.

![Image of a tree structure](image)

**Figure 3.10: Caption for trace**

However, a property of NIN-AND tree can be used here. Suppose that we have a root labeled tree $RLT$. The direct leaf gate NIN-AND tree of $RLT$ may be associated with a PCIF which specifies $pci(c_i, c_j) = p$. Then, the dual leaf gate NIN-AND tree of $RLT$ is associated with a PCIF which specifies $pci(c_i, c_j) = 1 - p$. Hence, if the
search key $SK$ refers to a dual leaf gate NIN-AND tree, we can reverse the value of each PCI of the search key, and search with this reversed search key $RSK$. The leaf gates of root labeled trees obtained with $RSK$ are labeled as dual leaf gates.

Based on these ideas, we develop the search method. The detail is given in Algorithms 5 and 6.

For this method, the inputs are $SK$: search key $SK = \{(b_1, p_1)\ldots(b_k, p_k)\}$, $RSK$: reversed search key $RSK = \{(b_1, rp_1)\ldots(b_k, rp_k)\}$, where $rp_i = 1 - p_i$, and $BST$ : the BST.

The final outputs are $CandidateDirect$: a set of indexes for direct leaf gate NIN-AND trees that are selected using the search key, and $CandidateDual$: a set of indexes for dual leaf gate NIN-AND trees that are selected.

We also define four lists to store intermediate results, which are $Direct$: queue of BST nodes corresponding to NIN-AND tree candidates with direct leaf gate, $Dual$: queue of BST nodes corresponding to NIN-AND tree candidates with dual leaf gate, and $CDirect$: temp queue used for backing up nodes in $Direct$, and $CDual$: temp queue used for backing up nodes in $Dual$. 
**Algorithm 5** Select the BST nodes

init queue $Direct$ and $Dual$ to include the root node only;

for $i = 1$ to $N$ do, //N is the depth of the BST

if $SK$ is empty then, break;

if $Direct$ and $Dual$ are empty then,

$Direct = CDirect$ and $Dual = CDual$; break;

$CDirect = Direct$ and $CDual = Dual$; //make copies for $Direct$ and $Dual$;

if there exists $(b_m, p_m)$ in $SK$ such that $b_m = i$ then,

remove $(b_m, p_m)$ from $SK$,

for each node $x$ in $Direct$ do,

remove $x$ from $Direct$;

append the child node $y$ of $x$ to $Direct$, where $y = p_m$;

remove $(b_m, rp_m)$ from $RSK$,

for each node $x$ in $Dual$ do,

remove $x$ from $Dual$;

append the child node $y$ of $x$ to $Dual$, where $y = rp_m$;

else ///$b_i$ is not an extracted bit

for each node $x$ in $Direct$ do,

remove $x$ from $Direct$;

append each child node of $x$ to $Direct$;

for each node $x$ in $Dual$ do,

remove $x$ from $Dual$;

append each child node of $x$ to $Dual$;

end for

return the pair ($Direct$, $Dual$);
Algorithm 6 Select NIN-AND trees

Require: Dual and Direct returned from Algorithm 5;

Ensure: CandidateDirect and CandidateDual;

init set CandidateDirect and CandidateDual to empty;

for each node $x$ in Direct with the index pair $(ix_1, ix_2)$;
    add elements $ix_1, ix_1 + 1, ..., ix_2$ to the set CandidateDirect;

for each node $x$ in Dual with the index pair $(ix_1, ix_2)$;
    add elements $ix_1, ix_1 + 1, ..., ix_2$ to the set CandidateDual;

return the pair (CandidateDirect, CandidateDual);

Suppose that we have the a 3-cause BST, and a partial search key extracted is \{(2,0)\}. The process of search is illustrated in Figure 3.11. For the purpose of simplicity, we divide the search process into two stages, one using the partial search key, and the other one using the reversed search key.
The details of the search process according to the algorithms are given below.

1. For the partial search key $SK \{ (2, 0) \}$, the reversed search key $RSK$ is $\{ (2, 1) \}$.

2. Consider the first iteration. In $SK$, the first PCI is not specified. Hence, starting from the root, we remove root from $Direct$ and $Dual$, and append the children of the root (node $n_1$ and node $n_2$) to each list. The first iteration now ends.

3. Consider the second iteration. The second PCI is specified as 0 in $SK$. For each node in $Direct$ (node $n_1$ and node $n_2$), append its left child to $Direct$. Remove
node $n_1$ and node $n_2$ from $Direct$, and remove $\{(2, 0)\}$ from $SK$. Then, $Direct$ contains node $n_3$ and node $n_5$.

4. The second PCI is specified as 1 in $RSK$. For each node in $Dual$ (node $n_1$ and node $n_2$), append its right child to $Dual$ (if the node does not have right child, then do nothing). Remove node $n_1$ and node $n_2$ from $Dual$, and remove $\{(2, 1)\}$ from $RSK$. Then, $Dual$ contains node $n_4$. The second iteration now ends.

5. Consider the third iteration. Because $Sk$ has no specified PCIs, the nodes selection algorithm is finished, and returns $(Direct, Dual)$ to the NIN-AND tree selection algorithm.

6. For each node in $Direct$, add the indexes in its indexes pairs to the set $CandidateDirect$. Based on node $n_3$, we add 0 and 1 to the set $CandidateDirect$. Based on node $n_5$, we add 3 to the set $CandidateDirect$.

7. For each node in $Dual$, add the indexes in its indexes pairs to the set $CandidateDual$. Based on node $n_4$, we add 2 to the set $CandidateDirect$.

8. Then, the root labeled trees with the indexes containing in $CandidateDirect$ and $CandidateDual$ are selected. In particular, the root labeled trees in $CandidateDirect$ (with indexes 0, 1, 3) are assigned with direct leaf gates. The root labeled tree in $CandidateDual$ (with index 2) is assigned with a dual leaf gate.

Finally, there are 4 NIN-AND trees selected. In particular, 3 direct leaf gate NIN-AND trees are selected, and have the same tree structures as the 0th, 1st and 3rd root labeled trees respectively. The last NIN-AND tree is a dual leaf gate NIN-AND
tree with the same tree structure as the $2_{nd}$ root labeled tree.

### 3.5 CPT Approximation with Gradient Descent Given A NIN-AND Tree

After selecting a subset of minimal NIN-AND trees, we can use the numerical method to approximate the target CPT with each NIN-AND tree selected. The NIN-AND tree $T$ giving the smallest distance $d$ and the corresponding set of SCPs $P$ is the output NIN-AND model $M = (T, P)$. In this section, the numerical method for optimally approximating a target CPT given a NIN-AND tree is proposed.

When the NIN-AND tree structure is determined, the WED function is defined. A WED function is continuous, multi-dimensional and nonlinear. *Gradient descent* and *Newton’s method* are two basic numerical optimization methods which can be used in this situation. Both methods require partial derivatives, and Newton’s method also uses second-order derivatives (*Hessian matrix*).

Derivatives can be obtained by two ways. One way is based on closed form formulas of derivatives. The other way is based on numerical method. As explained, WED function $E$ is defined based on the NIN-AND tree selected. Because the number of minimal NIN-AND trees increases exponentially, the number of possible WED functions can be large, which means that it is not easy to analyze derivatives by closed form formulas. Therefore, we use numerical method to calculate derivatives.

The equations of second-order derivatives calculation by numerical method is more complex than the equations of first-order derivatives. Because the numerical
method for obtaining derivatives is very sensitive, more errors may be introduced when the equations are complex [11]. To make results more reliable, we want to avoid using numerical method to obtain second-order derivatives. In addition, Netwon’s method asks to obtain Hessian matrix and obtaining Hessian matrix is very expensive (when the number of causes is \( n \), the size of Hessian is \( n \times n \)). Hence, we decide to use gradient descent as the optimization method.

### 3.5.1 Numerical Derivative

The method is based on the definition of derivative. Suppose that \( f(x_1, ..., x_n) \) is a function with multiple variables. Then the partial derivative of \( f \) at point \( p = (a_1, ..., a_n) \) along the direction of \( x_1 \) can be obtained by

\[
\frac{\partial f}{\partial x_1}(a_1, ..., a_n) \approx \frac{f(a_1 + h, a_2, ..., a_n) - f(a_1, ..., a_n)}{h}, \tag{3.4}
\]

or

\[
\frac{\partial f}{\partial x_1}(a_1, ..., a_n) \approx \frac{f(a_1 + h, a_2, ..., a_n) - f(a_1 - h, a_2, ..., a_n)}{2h}. \tag{3.5}
\]

Parameter \( h \) is a small increment on the direction of the \( x_1 \) and the limit of \( h \) is \( h \to 0 \).

Equation (3.5) is significantly better than Equation (3.4) if two function evaluations is affordable [11]. We experimented with evaluation of Equation (3.4) and Equation (3.5). If the numerical derivatives obtained based on Equation (3.4) is reliable, Equation (3.5) will not be used in our thesis, vice versa.

In our experiment, we set up two functions, one 4-cause WED function \( d_1 \) given the direct NIN-AND gate, and one 4-cause WED function \( d_2 \) given the dual NIN-
AND gate. The partial derivative functions of $d_1$ and $d_2$ are deduced as $f_1$ and $f_2$ respectively. Then, we randomly generate 10000 points. For each point, 6 results are generated. One is the partial derivative obtained based on Equation (3.4) and $d_1$, and is stored in the sample set $A.1$ of group $A$. The second result is the partial derivative obtained with Equation (3.4) and $d_2$, and is also stored in the sample set $A.1$ of group $A$. The third result is the partial derivative obtained with Equation (3.5) and $d_1$, and is stored in the sample set $B.1$ of group $B$. The forth result is the partial derivative obtained with Equation (3.5) and $d_2$, and is also stored in the sample set $B.1$ of group $B$. The last two are the partial derivatives obtained by directly using $f_1$ and $f_2$, and are stored in both the sample set $A.2$ of group $A$, and the sample set $B.2$ of group $B$.

This results are illustrated in Figure 3.12 using q-q plot. The left side graph shows the results of group $A$, and the right side graph shows the results of group $B$. The curves in graphs consist of many spots. For a spot $(x, y)$, $x$ is the partial derivative calculated by the numerical method (from $A.1$ or $B.1$), and $y$ is the corresponding derivative calculated by the closed form formula (from $A.2$ or $B.2$). Comparing these two graphs, the curve in the right side graph is closer to the straight line $y = x$. This means that in this graph, the derivatives obtained by the numerical method is closer to the derivative calculated by the closed form formula. Because the results in the right side graph is related to Equation (3.5), we think that the derivatives calculated with Equation (3.5) is more reliable than Equation (3.4), and should be used in our research.
Gradient descent method (or to be more specific, steepest descent) starts search from a given point, goes along the direction where the function $f$ has the steepest decrease, and finally stops at a point around the local minimum. In this section, we apply the gradient descent method to our problem.

Suppose that for function $f(x_1, ..., x_n)$, where $x_i$ is a real variable (in our problem, $x_i$ is the SCP of $c_i$). The gradient of $f$ at point $p = (a_1, ..., a_n)$, is denoted by $\text{grad}(p)$. Gradient $\text{grad}(p)$ is a vector of $n$ partial derivatives of $f$ at point $p$, $\text{grad}(p) = (\frac{\partial f}{\partial x_1}(a_1, ..., a_n), ..., \frac{\partial f}{\partial x_n}(a_1, ..., a_n))$. Based on Equation (3.5), to obtain a partial derivative, we have to be able to calculate the function $f$’s value at the given point. In our problem, function $f$ is a WED function and each element of a point $p$ is a SCP. An algorithm for specifying a CPT with a set of SCPs and a NIN-AND tree $T$ has already been proposed [16]. Because target CPT $cpt$ is known and the
NIN-AND tree is given in the problem of this subsection, the value of WED functions at a point \( p \) can be calculated. Hence, partial derivatives can be calculated as well. After calculating all partial derivatives, the gradient is calculated.

For every point \( p \), the gradient always points to the direction where \( f(p) \) has the steepest increase. Hence, by going with the opposite direction to the gradient, the gradient descent method can gradually find the local minimum by going along the directions where the function’s value has the most rapid decrease. An example of gradient descent is shown in Figure 3.13. The graph is a contour map of a function \( f(x, y) \), and each ellipse is a curve consisting of the points \((x_i, y_j)\) where \( f(x_m, y_n) = f(x_a, y_b) \). In addition, the function value at the ellipse closer to the local minimal is always higher than the function value at the ellipse less close to the local minimal.

![Gradient descent gradually approaches to the local minimum.](image)
Gradient descent can be used to find a local minimum. However, in our problem, we want to find an optimal optimization solution, which is a global minimum. To solve this problem, the algorithm starts from several different points. If they all converge at the same point, it is highly probable that the WED function has a global minimum at that point. Algorithm 7, the algorithm of gradient descent, is defined below.

Algorithm 7 returns the coordinate of the minimum point, and the minimum. Inputs are: $T$: a NIN-AND tree of $n$ causes, $cpt$: the target CPT, $AP$: the number of points to start with, $\epsilon$: threshold of gradient (if $\text{grad} < \epsilon$, stop iteration), $Step$: the maximum iterations for each execution, $TH$: threshold for judging whether two points can be regarded as the same one.
Algorithm 7 Gradient descent (steepest descent)

Ensure: $Point$ : the coordinate of the minimum (a set of $n$ SCPs);

$MIN$: minimum WED btw $cpt$ and $cpt'$;

init $MIN = 1000$;

for $i=1$ to $AP$ do,

randomly generate point $p_i = (x_{i1}, ..., x_{in})$ // $x_{ij} = P(e^+ \leftarrow c_j^+)$;

initiate step =0

while step $< Step$ do,

$step = step + 1$;

$p_i = GetNextPoint(p_i, T)$; //see Algorithm 8

get gradient $grad_i$ at $p_i$;

If ($grad_i < \epsilon$), break;

end while

If $(i > 1)$ then

$D = |x_{i1} - x_{11}| + ... + |x_{in} - x_{1n}|$;

If $(D > TH)$ then $ReportNotConverge()$; //not converged

calculate WED $d_i$ btw $cpt$ and the CPT $cpt'$ generated with $M(T, p_i)$;

If $(d_i < MIN)$ then $MIN = d_i$, $Point = p_i$;

end for

return the pair $(Point, MIN)$.
The boundary of the WED function is constrained by each variable’s range space, which is (0, 1). It is possible that the minimum point \( P = (x_1, ..., x_n) \) locates outside of the boundary. Therefore, after obtaining a new point in each iteration, we want to check whether the point locates inside of the boundary. If it does not, we force it into the boundary, and assign a value to \( x_i \) (the one outside of the range (0, 1)) as a number close to 1 or 0. This process is described by Algorithm 8. In the algorithm, \( \alpha \) is a scale parameter for distance (in n-dimensional space) moved in each step.

**Algorithm 8 GetNextPoint\((p, T)\)**

**Require:** point \( p = (x_1, ..., x_n) \) is a set of n SCPs; \( T \): a NIN-AND tree;

**Ensure:** a new point \( P = (X_1, ..., X_n) \), which is closer to the local minimum;

obtain gradient \( \text{grad} = (g_1, ..., g_n) \) of WED function for point \( p \) and \( T \);

create a new point \( P = (X_1, ..., X_n) \);

for \( j = 1 \) to \( n \) do

\( X_j = x_j - \alpha \ast g_j \); //"-" ensures descent;

If \( (X_j >= 1) \), \( X_j = 1 - 0.0001 \);

If \( (X_j =< 0) \), \( X_j = 0.0001 \);

end for

return point \( P \).
In addition, an example of Algorithm 7 is illustrated in Figure 3.14. In the example, the local minimum is outside of boundary. Hence, the gradient descent algorithm tends to go outside of boundary. However, because of the Algorithm 8, it is forced to be inside of the boundary, and go along one side of the boundary in this example.

Figure 3.14: An example of gradient descent when the local minimum outside of the boundary
Chapter 4

Experimental Design and Results

In the previous chapter, we have introduced the algorithms of gradient descent (steepest descent), root labeled trees indexing, BSTs construction, search key extraction and searching NIN-AND trees with BSTs. To evaluate these algorithms, we conducted experimental study. In this Chapter, we outline the design of experiments, and illustrate experimental results. In section 4.1, we present the experimental design. In particular, the data set and the setup of the experiments are discussed in this section. In section 4.2, the experimental including the statistical test results are illustrated.

4.1 Experimental Design

In this section, we present the experimental design and experimental results of our research.

4.1.1 Tasks of Experiments

There are two main tasks for the experiments. Because NIN-AND tree causal modeling is a generalization of the noisy-OR, it is predictable that approximation
with NIN-AND tree models can obtain better approximation results than using the noisy-OR. However, how much the improvement NIN-AND tree causal modeling can make is unknown. It is meaningful to get the answer of this question because it can help us to have a better understanding of NIN-AND tree causal modeling. Hence, the first task is to optimally approximate CPTs with NIN-AND tree models, and the noisy-OR respectively. After that, a comparison between the approximation results will be made.

In our research, we propose the methods for search key extraction and PCIs scoring. These methods allow us to improve the computation efficiency. In particular, we can generate a partial search key by using these two methods. As a result, a less number of NIN-AND trees will be selected, compared with enumerating all NIN-AND trees. Approximating a target CPT with more NIN-AND trees will have the better approximation accuracy. On the other hand, selecting a less number of NIN-AND trees improves the computational efficiency. There is a tradeoff between the computational efficiency and the approximation accuracy. Hence, the second task is to demonstrate the tradeoff experimentally.

4.1.2 Data Set

To do the experiments, we create a data set consisting of CPTs, which can be used as target CPTs. Because of the time limit, we decide to focus on CPTs with 3 to 5 causes. Therefore, these CPTs have 3 to 5 causes.

The most convenient way to get enough CPTs is to simulate CPTs. Therefore, the data set that we generate consists of simulated CPTs. A parameter of a simulated
CPT typically is a randomly generated decimal in the range $(0, 1)$. Because we have the assumption that all causes are modeled in this research, the first parameter of a simulated CPT $P(e^+ \leftarrow C^-)$ is the only one that has the value 0. There are 900 simulated CPTs generated in our experiments. In particular, the data set contains 300 3-cause CPTs, 300 4-cause CPTs, and 300 5-cause CPTs, as shown in Table 4.1.

<table>
<thead>
<tr>
<th>#causes</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>#CPTs</td>
<td>300</td>
<td>300</td>
<td>300</td>
</tr>
</tbody>
</table>

### 4.1.3 Optimal CPT Approximation with Noisy-OR and NIN-AND Trees

In this subsection, we outline the experiment for optimally approximate CPTs with NIN-AND tree models. We group the CPTs according to the number of causes. Therefore, we have 3 experiment groups, and each group consists of 300 CPTs with the same number of causes.

Each CPT in a group is optimally approximated with NIN-AND tree causal modeling and the noisy-OR respectively. The experimental results are stored into two groups, and each group corresponds to an approximation method.

The noisy-OR has one causal interaction structure given the number $n$ of causes. With each NIN-AND tree, a WED function can be defined. Suppose that the target CPT $cpt$ has 2 causes, the WED function is

$$d(cpt, cpt') = \sqrt{\frac{1}{3}(x_1 - p_1)^2 + \frac{1}{3}(x_2 - p_2)^2 + \frac{1}{3}(1 - (1 - x_1)(1 - x_2) - p_3)^2}, \quad (4.1)$$
where $x_i$ is the value of $i_{th}$ SCP $P(e^+ \leftarrow c_i^+)$, and $p_i$ is the $i_{th}$ parameter of the target CPT. The distance between the first parameter in both CPTs is always 0, and hence it is deleted from the WED function with the weight adjusted accordingly from $1/2^n$ to $1/2^{(n-1)}$. Hence, given the number of causes, running gradient descent algorithm once can obtain the optimal approximation result for the noisy-OR.

NIN-AND tree causal modeling has more options of causal interaction structures (NIN-AND trees). For example, for a 2-cause target CPT, we have two trivial NIN-AND trees, a direct NIN-AND gate and a dual NIN-AND gate. If we use a direct NIN-AND gate to approximate the CPT, the WED function is given as

$$d(cpt, cpt') = \sqrt{\frac{1}{3}(x_1 - p_1)^2 + \frac{1}{3}(x_2 - p_2)^2 + \frac{1}{3}(x_1 x_2 - p_3)^2}. \quad (4.2)$$

If we use a dual NIN-AND gate to approximate the CPT, the WED function is same as Equation (4.1). (Dual NIN-AND gates are similar to the noisy-OR.)

As shown above, NIN-AND tree causal modeling encode different causal interaction structure by different NIN-AND trees, and a WED function can be defined based on each NIN-AND tree. To optimally approximate a target CPT, we have to run gradient descent algorithm for each NIN-AND tree, and select the NIN-AND model $M = (T, P)$ according to the smallest $d$ value.

Based on this idea, for each simulated CPT, each time we approximate it with a NIN-AND tree till all NIN-AND trees are used. The best approximation result is stored in the sample set $A$. In addition, each CPT is also approximated the noisy-OR, and the result is stored in the sample set $B$.

The last step is to test the statistical significance, and to see whether the dif-
ference between the means of sample sets is generated by chance or not. Because
the sample sets are generated based on different methods and different equations, the
samples in a set is independent of the samples in the other set. In addition, based on
the result of normality test, the distribution of samples is not likely to be Gaussian
distribution. Hence, we decide to use Wilcoxon rank sum test.

The whole procedure of CPTs’ optimal approximation is shown in Figure 4.1.

![Flow chart of the experiment of optimal CPTs approximation](image)

Figure 4.1: Flow chart of the experiment of optimal CPTs approximation

### 4.1.4 Average Runtime of Gradient Descent Algorithm

As we described in the section of search algorithms, an algorithm allowing the
user to input time constraints is proposed. In this algorithm, the user needs to input
the runtime for each convergence of the gradient descent algorithm. Therefore, we
design an experiment to get the average runtime of gradient descent algorithm.

For each \( n \) value (the number of causes), there are 300 target CPTs. For each CPT, we run the gradient descent algorithm with 5 arbitrarily selected NIN-AND trees, and record 5 corresponding runtime. Therefore, the experiment generates \( 300 \times 5 = 1500 \) samples. Then, the mean of the samples is the average runtime. The above test is run for each of \( n = 3, 4, 5 \).

4.1.5 Evaluating Effectiveness of PCI Scoring

In the algorithm of PCIs extraction, each PCI is scored when it is extracted. The PCIs with higher scores are used for search with priority than the PCIs with low scores. To evaluate the algorithm, we want to examine whether approximation with the PCIs having the higher scores can result in better approximation accuracy.

For each CPT, the total number of PCIs specified by search key extraction algorithm is \( N \). Then, we use \( N/2 \) PCIs having the highest scores to select a set of NIN-AND trees. The approximation result using these selected NIN-AND trees is stored in sample set \( A \). At the same time, for each CPT, we randomly pick \( N/2 \) extracted PCIs, which means that these PCIs should have lower scores. Again, we use these PCIs to select NIN-AND trees, and store the approximation result in sample set \( B \). By comparing these two sample sets, we can evaluate the PCIs scoring algorithm. In particular, if the approximation results in sample set \( A \) is better than the results in sample set \( B \), it demonstrates that the PCIs having higher scores can result in better approximation accuracy.

The procedure is shown by Figure 4.2.
4.1.6 Evaluating Effectiveness of Search Key Extraction

As we discussed, when we use more PCIs to search NIN-AND trees, a less number of NIN-AND trees will be selected. As a result, the approximation accuracy may decrease. However, the curve of the accuracy decrease should be reasonably flat because the PCIs extracted are expected to fit the target CPT. If the approximation accuracy decreases dramatically when more PCIs are used, the PCIs may not be extracted properly. Hence, in this experiment, each target CPT is approximated several times. Each time uses a different number of PCIs. The approximation results (WEDs) are recoded.

In this experiment, we also want to see the tradeoff between the computational
efficiency and the approximation accuracy. Hence, it is also important to recode the runtime of each approximation (for the given number of PCIs).

The procedure is shown by Figure 4.3.

![Flow chart of the experiment evaluating search key extraction algorithm](image)

Figure 4.3: Flow chart of the experiment evaluating search key extraction algorithm

4.2 Experimental Results

In this section, we report the experimental results.

4.2.1 Experimental Result of Optimal CPT Approximation

The first experiment is to compare optimal approximation with NIN-AND tree causal modeling and the noisy-OR. Each target CPT is approximated with all NIN-AND trees and the noisy-OR. The approximation result (WEDs) for NIN-AND tree
causal modeling is stored in a sample set. The approximation result for the noisy-OR is stored in the other sample set. Two sample sets are compared finally.

![Boxplot showing WEDs from approximating with NIN-AND tree causal modeling (left side box) and with the noisy-OR (right side box).](image)

Figure 4.4: WEDs from approximating with NIN-AND tree causal modeling (left side box) and with the noisy-OR (right side box).

Figure 4.4 shows experiment results as boxplots. In the figure, the middle line of each box represents the 50\textsuperscript{th} percentile (median) of the corresponding sample set. The upper bound of the box is the 75\textsuperscript{th} percentile, which means that 25\% samples are greater than that value. Similarly, the lower bound of the is the 25\textsuperscript{th} percentile, which means that 25\% samples are smaller than that value. The ends of the axle represents the maximum and minimum of the samples excluding the outliers. The outliers, if any, are shown below the minimum line or above the maximum line.

Figure 4.4 shows that the optimal approximation results with NIN-AND tree causal modeling is significantly better than the optimal approximation results with
the noisy-OR.

The Wilcoxon rank sum test is also conducted. The \( p \)-value was estimated as smaller than \(2.2e-16\), which means that the null-hypothesis that these two sample sets are in the identical distribution is rejected. Hence, we conclude that using NIN-AND tree can significantly increase approximation accuracy.

### 4.2.2 Experimental Result of Average Runtime of Gradient Descent Algorithm

The next experiment is to obtain the average runtime of gradient descent algorithm. With an intel 2.53GHz i5 laptop, experiment results are obtained, and shown in Figure 4.5.

![Figure 4.5: Average runtime of gradient descent algorithm](image-url)
This figure shows that the average runtime roughly ascends along a straight line. This means that the average runtime increases linearly, and may be estimated by assuming that the average runtime follows a linear function, $y = mx$, where $m$ is the slope.

4.2.3 Experimental Result of Evaluating PCI Scoring Algorithm

The next experiment is to evaluate PCIs scoring algorithm. The comparison is done between the WEDs obtained by approximating CPTs with $N/2$ higher score PCIs and the WEDs obtained by approximating CPTs with $N/2$ randomly selected PCIs ($N$ is the total number of possible PCIs). Each method is run with 900 CPTs. Figure 4.6 illustrates the result. The result for the randomly selected PCIs is shown as the boxplot to the left. The result for the high score PCIs is shown as the boxplot to the right. It can be seen that the approximating with PCIs having the highest scores is more accurate.
Figure 4.6: Experimental results for evaluating PCIs scoring algorithm

Wilcoxon rank sum test is also used here. Again, the $p$-value was estimated as smaller than $2.2e^{-16}$. Hence the null hypothesis that these two sample sets are from the same distribution is rejected. Hence, we can conclude that approximating CPTs with the PCIs having higher scores can improve the approximation accuracy. It also means that the scoring algorithm works as expected.

4.2.4 Experimental Result of Evaluating Search Key Extraction Algorithm

The last experiment is to evaluate the search key extraction algorithm. In this experiment, we use different number of PCIs #PCI to select NIN-AND trees, and recode the WEDs obtained by approximating CPTs with #PCI PCIs. At the same time, we also recode the runtime for each approximation. The Wilcoxon rank sum test was conducted for 5-cause experiment results. For each pair of sample sets using
different numbers of PCIs, Wilcoxon rank sum test was conducted. In addition, the Wilcoxon rank sum test between sample sets of using a full search key (using all PCIs) and using the noisy-OR is also conducted. The experiment results are shown in the following figures.

Figure 4.7 shows the experiment result with 3-cause CPTs data set. It can be observed that the approximation accuracy decreases slowly with the increase of \(\#PCI\). This shows that the PCIs for these 3-cause CPTs were extracted properly.

![Figure 4.7: WEDs for approximating 3-cause CPT](image)

Figure 4.7: WEDs for approximating 3-cause CPT
Figure 4.8 shows the average runtime for approximating 3-cause CPTs, where \( N \), the total number of PCIs, equals to 3. It is observable that the runtime increases exponentially when the number of PCIs used for search decreases. This is intuitive because it shows that the number of selected NIN-AND trees increases exponentially when the number of PCIs specified decreases. It is important to notice that the approximation accuracy results using 2 and 3 PCIs are quite similar to the optimal approximation accuracy result (using 0 PCIs). At the same time, the runtime of approximations using 2 and 3 PCIs is about as twice as the runtime of approximation with the noisy-OR, but is still very small. Due to the improved approximation accuracy, NIN-AND tree causal modeling approximation using 2 and 3 PCIs is more advantageous than approximating with the noisy-OR.

Figure 4.8: Average runtime for approximating 3-cause CPTs
Figure 4.9 shows the experiment result with 4-cause CPTs data set, where $N = 6$. It can be observed that the approximation accuracy decreases slowly at beginning, and decreases more rapidly when $\# PCI$ is greater than 3. However, the average WED obtained when all PCIs are specified ($\# PCI = 6$) is about 0.22, and the average WED obtained by enumerating all NIN-AND trees is about 0.19. The difference is not significant. Hence, we think the increasing speed of average WED is reasonable good, which means that the PCIs (especially the PCIs with the highest scores) were extracted properly.

![Figure 4.9: WEDs for approximating 4-cause CPT](image)
Figure 4.10 shows the average runtime for approximating 4-cause CPTs. The runtime increases exponentially when the number of PCIs specified decreases. Especially, the runtime using 0 PCIs can be hundreds times as the runtime of the noisy-OR. However, the runtime of the approximations using more than 3 PCIs increases reasonably flat. In addition, the approximation accuracy using more than 3 PCIs is significantly better than using the noisy-OR. We believe that the tradeoff between the computational efficiency and the approximation accuracy is still good in this case.

![Average runtime for approximating 4-cause CPTs](image)

Figure 4.10: Average runtime for approximating 4-cause CPTs
Figure 4.11 shows the experiment result with 5-cause CPTs data set, where $N = 10$. Similar to the experiment result with 4-cause CPTs, the approximation accuracy decreases slowly at beginning, and decreases more rapidly when $\#PCI$ is greater than 5. However, the approximation accuracy is still significantly better than the approximation accuracy with the noisy-OR. This shows that the PCIs were extracted properly.

![Box plot graph](image-url)

**Figure 4.11:** WEDs for approximating 5-cause CPT
Figure 4.12 shows the average runtime for approximating 5-cause CPTs. The runtime increases as the number of specified PCIs increases. However, when the number specified PCIs is greater than 6, the curve ascends slowly. At the same time, the approximation accuracy using these PCIs is significantly better than the accuracy using the noisy-OR.

![Figure 4.12: Average runtime for approximating 5-cause CPTs](image)

It is important to notice that using full PCIs will only select one NIN-AND tree. The runtime of approximating a CPT with a NIN-AND tree (all PCIs are specified) is very close to the runtime of approximating with the noisy-OR. However, its approximation accuracy is significantly higher than the approximation accuracy using the noisy-OR. Another fact is that the minimum WEDs obtained by approximating CPTs with NIN-AND tree causal modeling can be very small, and can even be close to 0. On the contrast, normally the minimum WEDs obtained by approximating with the noisy-OR is notably greater than 0. This shows that NIN-AND tree causal
modeling have more options of causal interaction structures, and is more likely to fit a target CPT.

Wilcoxon rank sum test was done for this group of experiment results. The Wilcoxon rank sum test results are shown in the Table 4.2.

Table 4.2: The Wilcoxon rank sum test for 5-cause experiment results

<table>
<thead>
<tr>
<th>sample sets</th>
<th>p-value</th>
<th>sample sets</th>
<th>p-value</th>
<th>sample sets</th>
<th>p-value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1</td>
<td>.9057</td>
<td>1-5</td>
<td>.06964</td>
<td>3-10</td>
<td>1.845e − 08</td>
</tr>
<tr>
<td>0-3</td>
<td>.59</td>
<td>1-7</td>
<td>.0009102</td>
<td>5-7</td>
<td>.1125</td>
</tr>
<tr>
<td>0-5</td>
<td>.05844</td>
<td>1-9</td>
<td>1.624e − 08</td>
<td>5-9</td>
<td>5.07e − 05</td>
</tr>
<tr>
<td>0-7</td>
<td>.0006922</td>
<td>1-10</td>
<td>2.262e − 09</td>
<td>5-10</td>
<td>1.215e − 05</td>
</tr>
<tr>
<td>0-9</td>
<td>1.028e − 08</td>
<td>3-5</td>
<td>.1619</td>
<td>7-9</td>
<td>.01214</td>
</tr>
<tr>
<td>0-10</td>
<td>1.417e − 09</td>
<td>3-7</td>
<td>.003002</td>
<td>7-10</td>
<td>.003302</td>
</tr>
<tr>
<td>1-3</td>
<td>.6645</td>
<td>3-9</td>
<td>9.808e − 08</td>
<td>9-10</td>
<td>.6087</td>
</tr>
<tr>
<td>10-noisy-OR</td>
<td>&lt; 2.2e − 16</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Based on the Wilcoxon test result, we can get two conclusions.

1. Distributions of samples obtained by NIN-AND tree causal modeling are not the same distribution of the samples obtained by the noisy-OR.

2. Use different numbers of PCIs $n$ and $m$ to search NIN-AND trees. If $n$ and $m$ are close to each other, the distributions of samples are more similar to each other. On the contrary, when $n$ and $m$ are significantly different from each other, it is not likely that they are from the same distribution.
Chapter 5

Conclusion

5.1 Summary

A Bayesian network consists of a DAG and a set of CPTs. CPTs represent the quantitative knowledge of the BN. A CPT can be either extracted from data sets or specified by experts. However, the number of parameters of a CPT increases exponentially on the number of related causes. Causal models are proposed to specify a CPT with fewer parameters. NIN-AND tree causal modeling is able to represent two types of causal interactions. By replacing a CPT by a NIN-AND tree model, there are two advantages. The first advantage is that the number of parameters kept is exponentially smaller comparing with storing the whole CPT. The second advantage is that NIN-AND tree causal modeling can speed up inference with BNs. Being motivated by these two reasons, we develop the techniques for approximating a CPT with a NIN-AND tree model. In particular, the algorithms for minimal NIN-AND trees indexing, BSTs construction, search key extraction and scoring, tree search with a BST and approximation with a given NIN-AND tree are developed. Note that even though the target CPTs in a BN can be replaced by NIN-AND models after the approximation, they must be fully specified before the approximation process starts.
The experimental study is performed to evaluate the effectiveness of these algorithms. The error introduced by approximation increases slowly when the number of PCIs contained in the search key increases. This shows that the values of PCIs extracted can properly reflect the type of causal interaction. The approximation result with the NIN-AND trees selected by the PCIs having the highest scores is significantly better than the result based on the PCIs having lower scores. This shows that the assigned scores of PCIs can properly reflect the strength of corresponding causal interactions. In addition, the experimental results also show that approximating a CPT with a NIN-AND tree model can be significantly better than approximating with the noisy-OR, and the runtime using these two approximating methods are very similar.

5.2 Contributions

With the techniques developed in this thesis, a complete scheme of CPTs approximation with NIN-AND tree models is proposed. Instead of keeping a whole CPT, it allows to approximate the CPT and store the obtained NIN-AND tree model. The number of parameters of a CPT increases exponentially on the number of causes, and the number of parameters in a NIN-AND model is linear on the number of causes. Hence, keeping NIN-AND models can dramatically improve space efficiency. In addition, by replacing CPTs with NIN-AND tree models, inference can be made more efficient.
5.3 Limitations

In our experiments, the CPTs we simulated have no more than 5 causes. Hence, the scale of the experimental study is limited. The second limitation is that experiments did not use real-world CPTs. The parameters of simulated CPTs are generated randomly, and hence are arbitrarily distributed. However, the parameters of real-world CPTs are the quantitative knowledge of real-world problems, and are not arbitrarily distributed. The third limitation is that only binary variables are considered in our research. The last limitation is that leak cause is not considered. This means that the distance between $p_0$ and $p'_0$ is a constant (which equals to $p_0^2$) determined by the target CPT. If we consider the leak cause, $p'_0$ does not have to be 0, and hence, the distance between $p_0$ and $p'_0$ can be reduced.

5.4 Future Work

Approximating algorithms are designed to approximate a given CPT with a NIN-AND tree model. Hence, it is needed to consider CPTs which contains variables having multiple values. More research is needed on this aspect. To reduce approximation error, leak cause may be introduced as well. In addition, more experiments with real-world CPTs having more than 5 causes may be done in the future. Moreover, the numerical method for approximating a target CPT given a NIN-AND tree is gradient descent (steepest descent in particular). Steepest descent may not be the method having the fastest convergence in our problem. More study can be done for find a better numerical optimization method.
Bibliography


