Mixing ICI and CSI Models for More Efficient Probabilistic Inference

by

Michael Roher

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

ABSTRACT

Mixing ICI and CSI Models for More Efficient Probabilistic Inference

Michael Roher
University of Guelph, 2020

Advisor:
Dr. Yang Xiang

Bayesian Networks (BNs) concisely represent probabilistic knowledge of uncertain environments by encoding causal dependencies and exploiting conditional independencies between variables. The strength of each variable's dependence on its parents is quantified by a conditional probability table (CPT). However, these CPTs suffer from an exponential growth on the number of parents.

To address the exponential growth, various local models have been introduced for representational savings and further inference efficiency. Some exploit context-specific independence (CSI), which concisely encode duplicated probabilities. Others exploit independence of causal influence (ICI), which encode causal relationships between variables. Existing techniques apply only ICI or only CSI in a BN, such that exploiting one model sacrifice savings yielded by the other.

We develop an exact inference framework for BNs modelled with both: We apply Non-Impeding Noisy-AND Trees for ICI, and CPT-trees for CSI. The experimental evaluation demonstrates a significant inference efficiency gain beyond what is attainable by exploiting only one type of model.

Acknowledgments

First and foremost, I would like to offer my deepest gratitude to my supervisor, Yang Xiang. I am grateful for his invaluable guidance, insight and support. I would like to thank my committee members, Pascal Matsakis and Mark Wineberg, for their excellent feedback and constructive comments when reading this thesis. Lastly, I want to express my sincere appreciation of my family and friends for their endless support, proofreading, and laughs.

Table of Contents

List of Tables

List of Figures

List of Abbreviations

ACAL All Children Are Leaves.

BN Bayesian Network.

CIM Causal Independence Model.

CMBN CPT-Tree Modelled Bayesian Network.

CPD Conditional Probability Distribution.

CPT Conditional Probability Table.

CSI Context-Specific Independence.

D+N De-causalization and Normalization Conversion Method.

D+T De-causalization and Transformation Conversion Method.

DAG Directed Acyclic Graph.

DTBN De-causalized and Transformed Bayesian Network.

ED Euclidean Distance.

GHz Gigahertz.

ICI Independence of Causal Influence.

JPD Joint Probability Distribution.

JT Junction Tree.

KL Kullback-Leibler Divergence.

LP Lazy Propagation.

MNCBN Mixed NAT-CSI Bayesian network.

N+N Normalization and Normalization Conversion Method.

N+T Normalization and Transformation Conversion Method.

NAT Non-Impeding Noisy-AND Tree.

NIN-AND Non-Impeding Noisy-AND Tree.

NMBN NAT-Modelled Bayesian Network.

Chapter 1

Introduction

1.1 Overview

Uncertainty is ubiquitous in the real-world. Whether a doctor is diagnosing a patient, a robot vacuum cleaner is sweeping a room, or a gambler is playing a card game in a casino, we are likely making decisions without complete knowledge of an uncertain environment. In the context of artificial intelligence, a common method of representing uncertain knowledge is Bayesian probability theory. Bayesian probability theory models the subjective belief of an agent by a probability of an event, given the knowledge that another event has occurred.

Consider the example of a doctor diagnosing whether or not a patient has a certain disease. The doctor will assess multiple factors, some of which are observed, while others are unknown. The observed factors may include the patient's medical history, blood tests, and any pre-existing conditions. But, these tests are not perfect, and there may be false positives, or symptoms that are imperceivable through these tests. Thus, the doctor may represent the patient's likelihood of suffering from the disease as a probability, conditioned on the medical history, test results and pre-existing conditions.

One method of representing the agent's subjective knowledge of the environment is through a *joint probability distribution* (JPD). Given a set of variables in an environment, the JPD consists of a table specifying every instantiation of all variables in the set. Each instantiation has a corresponding probability specifying the likelihood of the instantiation occurring. In small environments, a JPD may be sufficient for inference. But, as we increase the number of variables, the size of the JPD increases exponentially, quickly leading to models requiring billions of instantiations.

The exponential growth of a JPD is not simply a representational issue. It leads to computational inefficiencies during inference and can ultimately lead to intractability for large environments. One method to reduce the exponential growth is by applying a model that exploits conditional independencies between variables. For example, the likelihood of a patient suffering from a disease may be dependent on a certain genotype, which may itself be affected by the parents' genotype. But, once the patient's genotype is identified, the parents' genotype is no longer relevant.

A JPD does not take advantage of the independencies between variables. This was addressed by Pearl and others who introduced *Bayesian networks* (BNs) [12] to model the structure in the environment. A BN consists of two components: a directed acyclic graph to encode the dependencies and exploit the conditional independencies between variables, and a conditional probability table (CPT) for each node in the graph to quantify the strength of the dependency of the given node on its parent variables. This reduces the exponential explosion on the number of variables from exponential in JPDs to linear in BNs.

However, the CPTs are still exponential over the number of dependencies a node has in the graph. Previous research has noted that a CPT can be expressed more efficiently by replacing it with an alternative structure. Many of these alternative structures can be grouped into two classes:

Independence of Causal Influence (ICI) Encodes the relationship between a variable and its dependencies more efficiently. For example, consider a patient recovering from a headache by taking medicine and increasing water intake. An efficient ICI model would represent the patient's recovery by each treatment individually. That is, it would require the probability of a patient recovering from the headache by only taking medicine, and the probability of a patient recovering from the headache by only increasing their water intake. Operations on these events allow for the probabilities of all other combinations of treatments (i.e., both treatments, or neither treatment) to be computed. These operations reduce the number of instantiations required to specify this model from exponential to linear on the number of treatments. Models in this class include Noisy-OR [12], Noisy-MAX [8], DeMorgan [10] and Non-Impeding Noisy-AND Trees (NAT or NIN-AND) [21]. Further details of ICI models are available in Section 2.7.

Context-Specific Independence (CSI) Encodes duplicate values in probability tables more efficiently. For example, consider a patient's recovery from surgery that is dependent on whether or not they receive physiotherapy, and the skill of the physiotherapist. If the patient completes physiotherapy, then the likelihood of recovery increases. The magnitude of the increase is dependent on the skill of the physiotherapist. A highly skilled physiotherapist will result in a significant increase to the likelihood of recovery, whereas a less skilled physiotherapist will result in a small increase to the likelihood of recovery. On the other hand, if the patient declines physiotherapy, then the likelihood of recovery decreases.

In the most naive form, this would require four instantiations. However, it is observed that when the patient declines physiotherapy, the likelihood of recovery decreases, regardless of the skill of the physiotherapist. A CSI model would encode this model efficiently by exploiting the fact that the variable encoding the physiotherapist's skill is redundant when the patient declines physiotherapy. This would reduce the number of instantiations to three. Models in the CSI class include default tables [3], CPT-trees [3], rule-based CSI [14], and algebraic decision diagrams [4]. Further details of CSI models are available in Section 2.8.

Unfortunately, inference methods for Bayesian networks modelled with CPTs are not directly compatible when alternative structures are used. Most naively, the alternative structures may be expanded into exponentially sized CPTs in order to execute these inference methods. This is undesirable as it discards all savings the alternative structure provides. Instead, significant research has been directed towards identifying techniques that prepare a BN modelled with alternative structures for efficient inference. The specific techniques vary depending on the alternative structure. For instance, some models may have special conversion methods that expand the alternative structure to a probability table while maintaining computational savings.

To our knowledge, all previous research has focused on replacing each probability table in a BN with one class of alternative structures. This restricts the BN to the same class for all variables. Since these alternative structures apply on a per-variable basis, it is plausible they can coexist in the same environment. Restricting ourselves to one class relinquishes the opportunity to exploit the other class in the same BN.

Moreover, if the variable is not well-suited for the alternative structure, then it must be approximated by the alternative structure, or represented by a probability table. Neither of these options are ideal. The approximation is not exact due to the inability of a model from one class to exactly and efficiently represent a model from the other class. On the other hand, a large probability table representation sacrifices the inference efficiency savings yielded by the the other efficient alternative structures in the BN.

The purpose of this thesis is to develop an inference framework for BNs modelled with both ICI and CSI alternative structures. When both exist in a Bayesian network, we apply NAT models for ICI and CPT-tree models for CSI. Each alternative structure is then compiled into a probability table through special conversion methods that preserve computational savings. The result is an efficient BN where each variable is quantified by a tabular representation. This efficient BN is then compatible with any typical BN inference method.

1.2 Contributions

In this thesis, we make four main contributions:

1. We propose a framework to exploit both NAT and CSI local models in probabilistic inference. We evaluate the efficiency of this framework on *Lazy Propagation*, an inference technique shown to attain a two orders of magnitude speedup on very sparse Bayesian networks [11].
2. We empirically demonstrate that one class of alternative structures cannot be efficiently and exactly encoded by the other class; thereby, validating the necessity of this research.
3. We generalize and formalize the CPT-tree network transformation algorithm by specifying a comprehensive algorithm suite. This advances the initial idea presented by Boutillier et al. [3].
4. We establish the existence of CSI in real-world BNs. These results, in conjunction with previous research showing the existence of ICI in real-world BNs [22], demonstrate the coexistence of both ICI and CSI in the real-world.

1.3 Thesis Layout

The remainder of this thesis is laid out as follows. Chapter 2 is a summary of the background knowledge underlying this thesis. Chapter 3 empirically demonstrates that NAT models and CSI models cannot efficiently and exactly encode each other. Chapter 4 formalizes the CPT-tree transformation algorithm. Chapter 5 develops the framework. Chapter 6 identifies CSI in real world BNs and evaluates the impact of the framework on synthetic BNs. Chapter 7 offers concluding remarks and possible future research opportunities.

Chapter 2

Background

In this chapter, we present an overview of the background material underlying this thesis. The chapter is organized as follows: Sections 2.1 to 2.3 review graph theory fundamentals, two common interpretations of probability theory, and potentials, respectively. Section 2.4 reviews joint probability distributions and an accompanying inference method. Section 2.5 reviews Bayesian networks and two accompanying inference methods. Section 2.6 reviews accuracy metrics to evaluate the closeness of an approximation from its ground truth. Finally, we review independence of causal influence in Section 2.7 and context-specific independence Section 2.8, which each may be exploited for further representational and inference savings.

2.1 Graph Theory Fundamentals

A graph, $G = (N, E)$ is a mathematical structure to represent a non-empty set of nodes N connected by a set of edges E. Edges can be undirected or directed. An undirected edge $\{n_i, n_j\}$ where $n_i, n_j \in N$ is a symmetric connection between the two endpoints. A directed edge, (n_i, n_j) where $n_i, n_j \in N$ is an asymmetric connection between the two endpoints. We note the notation is intentionally different: braces represent an unordered pair, while parenthesis represent an ordered pair. If an edge (n_i, n_j) is directed from n_i to n_j, the node n_i is said to be a *parent* (or source) of n_j and n_j is said to be a *child* (or target) of n_i. Two nodes are *adjacent* if they are connected by an edge. A graph with all directed edges is a *directed graph*, a graph with all undirected edges is an *undirected graph*, otherwise the graph is a *hybrid graph*.

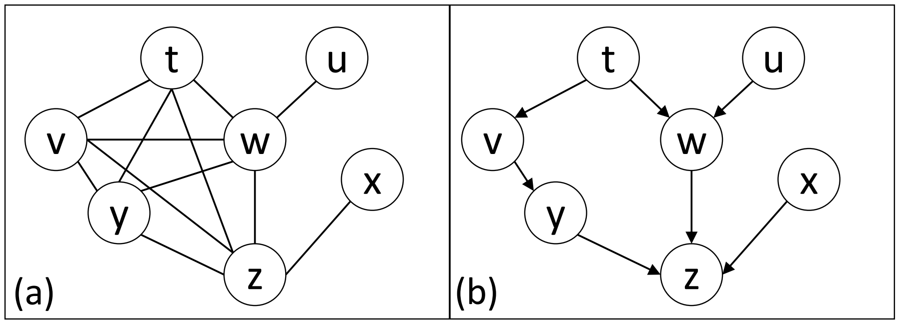


Figure 2.1: (a) An example undirected graph. (b) An example directed graph.

A *walk* in a directed graph $G = (N, E)$, is a sequence of nodes $(n_0, n_1, n_2, \ldots, n_k)$ with a corresponding sequence of edges $((n_0, n_1), (n_1, n_2), \ldots, (n_{k-1}, n_k))$ such that each node n_i in the sequence is in N and each edge (n_i, n_{i+1}) is in the sequence is in E. We denote n_0 and n_k as the start and end of the walk, respectively. A *path* is a walk where each node in the node sequence occurs only once. A *cycle* is a special type of path that starts and ends at the same node (i.e. $n_0 = n_k$). A graph is said to be *cyclic* if it contains at least one cycle, and *acyclic* otherwise.

A *directed acyclic graph* (DAG) is depicted in Figure 2.1 (b) as all edges are directed and there are no cycles within the graph.

A *family* of a node n_i in a directed graph $G = (N, E)$ is $\{n_i, \pi(n_i)\}$ where $\pi(n_i)$ denotes all edges in E pointing to n_i. To eliminate any ambiguity, a family in a graph theory context differs from a family in a genealogy context. In the graph theory context, a family *does not* include the siblings of the child node. By contrast, a family in the genealogy context *does* include the siblings of the child. In Figure 2.1 (b), the family of node w is $\{w, t, u\}$.

A graph is *connected* if there is a path between every pair of nodes. A *tree* is an acyclic connected graph that has exactly one path between every pair of nodes.

A *clique* of an undirected graph G is a fully-connected subgraph within G. For example, the set of nodes $\{t, v, y, z, w\}$ in Figure 2.1 (a) is a clique of size 5.

2.2 Interpretations of Probability

In partially observable and uncertain environments, an agent will not have complete knowledge of the environment. A common approach to operate under uncertainty is through probability theory. Two interpretations of probability include *frequentist* and *Bayesian*.

Frequentist probability interprets probability as the frequency of an event occurring. It is objective and the frequentist probability of an event a converges as the total number of trials approaches infinity: $P(\alpha) = \lim_{n\to\infty} \frac{t}{n}$ where t represents the number of trials where the event α occurs, and n represents the total number of trials. This is commonly approximated by $P(\alpha) \approx \frac{t}{n}$. However, when an event can only occur once or it is impractical to repeat the event in the real-world, the probability of the event is undefined. For example, the probability of the next global pandemic occurring cannot be observed by repeating experiments.

Bayesian probability interprets probability as a subjective value indicating one's degree of belief in the event occurring. The Bayesian interpretation is capable of representing unrepeatable events. For example, the probability of the next global pandemic occurring may be specified by an infectious disease expert's degree of belief.

Throughout this thesis, we will assume the Bayesian interpretation.

2.3 Potentials

In this section, we introduce *potentials*, which are used in subsequent methods. A potential $\phi(M)$ over a set of variables M is a function, which maps from a set of M's values $val(M)$ to a set of non-negative real numbers. Potentials do not necessarily comply with the laws of probability — they may hold values outside of $[0, 1]$. Every CPT is a potential but not every potential is a CPT.

If M_1 and M_2 are sets of variables with the corresponding potentials $\phi_1(M_1)$ and

$\phi_2(M_2)$, then the *product* of potentials is specified by:

$$\phi(M_1, M_2) = \phi_1(M_1) \times \phi_2(M_2)$$

Potentials also support *marginalization* to sum-out a variable.

$$\phi(M_1) = \sum_{m_2 \in val(M_2)} \phi(M_1, M_2 = m_2)$$

2.4 Joint Probability Distributions

To represent uncertain knowledge in an environment, one may choose to use a *joint probability distribution* (JPD). A JPD encodes a probability for each instantiation of a set of variables M. An *instantiation* of M represents an assignment of value(s) to each variable in M. Figure 2.2 presents a JPD for 7 variables u, v, t, w, x, y, z. Each variable is binary with each value of the form: the variable's letter followed by an index (e.g. $\{t_0, t_1\}$). The probability for a given instantiation is obtained by locating the instantiation in the JPD. For instance, if t, u, v, w, x, y, z hold $t_0, u_0, v_0, w_0, x_0, y_0, z_0$, respectively, then we can obtain the probability of this event as $P(t = t_0, u = u_0, v = v_0, w = w_0, x = x_0, y = y_0, z = z_0) = 0.0020$.

However, a critical limitation of JPDs is that the number of probabilities in a JPD is exponential on the number of variables in the environment. The example in Figure 2.2 has 7 binary variables resulting in $2^7 = 128$ probabilities. If the environment instead has 10 variables with a domain size of 5, then the JPD will have $5^{10} = 9,765,625$ probabilities. This limitation is amplified when modelling real-world environments, which may have hundreds of variables. The exponentially increasing number of probabilities has significant time and space implications for storage and any operations to be performed on the JPD.

t	u	v	w	x	y	z	$P(t,u,v,w,x,y,z)$
t_0	u_0	v_0	w_0	x_0	y_0	z_0	0.0020
t_0	u_0	v_0	w_0	x_0	y_0	z_1	0.0002
t_0	u_0	v_0	w_0	x_0	y_1	z_0	0.0004
t_0	u_0	v_0	w_0	x_0	y_1	z_1	0.0006
t_0	u_0	v_0	w_0	x_1	y_0	z_0	0.0020
...	...	...	...	...	...	...	...
t_1	u_1	v_1	w_1	x_0	y_1	z_1	0.0175
t_1	u_1	v_1	w_1	x_1	y_0	z_0	0.0117
t_1	u_1	v_1	w_1	x_1	y_0	z_1	0.1050
t_1	u_1	v_1	w_1	x_1	y_1	z_0	0.0117
t_1	u_1	v_1	w_1	x_1	y_1	z_1	0.0175

Figure 2.2: JPD over seven binary variables t, u, v, w, x, y, z. The JPD has 2^7 instantiations. Due to space reasons, only the first 5 and last 5 instantiations are shown.

2.4.1 Inference By JPD

A common operation on uncertain knowledge representations is inference. Inference is the process of determining the probability of an event, given a set of observations. To demonstrate inference by JPD, suppose we would like to know the posterior distribution $P(z|x = x_0)$ from the JPD in Figure 2.2. The method begins by first updating the JPD to $P(t, u, v, w, y, z|x = x_0)$ by the following *product rule* [20]:

$$P(t, u, v, w, y, z|x = x_0) = \frac{P(t, u, v, w, y, z, x = x_0)}{P(x = x_0)}$$

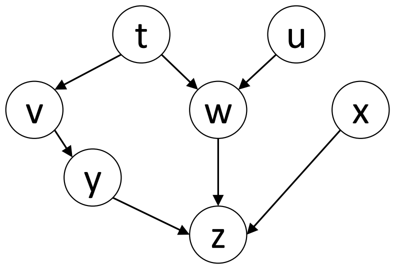
The distribution in the numerator $P(t, u, v, w, y, z, x = x_0)$ is computed by setting the probability of all instantiations inconsistent with the observation $x = x_0$ to 0. In other words, given an instantiation $(t', u', v', w', y', z', x')$, if $x' \neq 0$, its corresponding probability is set to 0. The denominator is subsequently computed as the sum of the remaining terms. We then divide each remaining term by the sum resulting in the distribution $P(t, u, v, w, y, z|x = x_0)$.

We then apply the marginalization operation to sum-out all non-query variables.

$$P(z|x = x_0) = \sum_t \sum_u \sum_v \sum_w \sum_y P(t, u, v, w, y, z|x = x_0)$$

In this example, the exponential nature of JPD requires maintaining a potential of 7 variables or $2^7 = 128$ instantiations. For further details on inference by JPD, refer to [20].

2.5 Bayesian Networks



$P(t)$	
t_0	t_1
0.4	0.6

$P(u)$	
u_0	u_1
0.1	0.9

$P(x)$	
x_0	x_1
0.5	0.5

	$P(v|t)$	
t	v_0	v_1
t_0	0.8	0.2
t_1	0.1	0.9

	$P(y|v)$	
v	y_0	y_1
v_0	0.7	0.3
v_1	0.8	0.2

		$P(w|t,u)$	
t	u	w_0	w_1
t_0	u_0	0.2	0.8
t_0	u_1	0.9	0.1
t_1	u_0	0.3	0.7
t_1	u_1	0.4	0.6

			$P(z|y,w,x)$	
y	w	x	z_0	z_1
y_0	w_0	x_0	0.9	0.1
y_0	w_0	x_1	0.9	0.1
y_0	w_1	x_0	0.2	0.8
y_0	w_1	x_1	0.1	0.9
y_1	w_0	x_0	0.4	0.6
y_1	w_0	x_1	0.4	0.6
y_1	w_1	x_0	0.4	0.6
y_1	w_1	x_1	0.4	0.6

Figure 2.3: A Bayesian network's two components: a DAG (top-left) and a CPT for each node.

2.5.1 Representation

To improve on the exponential explosion of JPDs, Pearl and others [12] proposed Bayesian networks (BN), a more efficient solution to representing uncertain knowledge. Before introducing Bayesian networks [12, 20], it is first necessary to define *conditional independence.*

Conditional Independence *Let A, B and Z be disjoint subsets of variables. A and B are* conditionally independent *given Z, denoted $I(A, Z, B)$, iff*

$$P(A|B, Z) = P(A|Z) \text{ whenever } P(B, Z) > 0.$$

Conditional independence can be exploited by observing that it is unlikely all variables will affect all other variables. Instead, it is likely only a subset of the variables in the environment will directly affect a variable. This is the key idea that allows for Bayesian networks to factor JPDs into a more efficient representation. Formally, a Bayesian network (M, G, P) is a triplet specified in terms of the following:

- M is a set of variables.
- G is a directed acyclic graph whose nodes correspond one-to-one to members of M. Each variable in the graph is conditionally independent of its non-descendants given its parents.
- P is a set of *conditional probability distributions* (CPDs) for each variable $m_i \in M$, specifying the distribution for m_i over its parents $\pi(m_i)$: $P(m_i|\pi(m_i))$. A set of CPDs is said to be a *conditional probability table.*

Intuitively, the Bayesian network can be viewed as a a graph specifying the dependence structure of the variables, with the conditional distributions specifying the strength of each dependence. An example BN is presented in Figure 2.3. Since a BN is a factored JPD, the JPD for a BN can be retrieved by combining the CPDs according to the *chain rule.*

$$P(m_1, m_2, \ldots, m_n) = \prod_{i=1}^{n} P(m_i | \pi(m_i))$$

The BN uses a DAG in conjunction with multiple, smaller CPTs to efficiently represent JPDs. Each CPT has a size exponential on the family size: $O(d^f)$ where d represents the largest domain size of the family and f represents the family size.

2.5.2 Inference by Variable Elimination

In Section 2.4.1, we demonstrate that a JPD is capable of performing inference by performing a potential's marginalization and product operations on the joint distribution. This suffers from intractability as the JPD is exponential on the number of variables. Since a BN factorizes a JPD into smaller CPTs by exploiting conditional independence, a valid yet naive approach would be to perform inference on BN by converting the BN into a JPD and performing inference on the resulting JPD. Suppose, we would like to compute the prior $P(z)$ from the distribution shown in Figure 2.3. For brevity, we omit the introduction of evidence. Refer to [5] for details.

$$P(z) = \sum_{t,u,v,w,x,y} P(t, u, v, w, x, y, z)$$

One method of improving on this approach is *variable elimination*. Similar to inference by JPD, variable elimination is an inference technique that consists of successively summing-out all non-query variables to construct a marginal distribution over the remaining query variables. The key insight, however, is that variables can be marginalized out while keeping the original distribution and all successive distributions in some factored form [4]. This is achieved by rewriting in terms of conditional independencies.

$$P(z) = \sum_{t,u,v,w,x,y} P(z|y, w, x)P(x)P(t)P(v|t)P(y|v)P(w|t, u)P(u)$$

We factor this expression by re-arranging terms and pushing the summations inside the product operations. This allows for the summations to be performed as early as possible and the product operations as late as possible.

$$P(z) = \sum_{y,w,x} P(z|y,w,x)P(x) \sum_{t,v} P(t)P(v|t)P(y|v) \sum_{u} P(w|t,u)P(u)$$

This results in the following computations.

$$\begin{aligned}
\phi_1(w,t,u) &= P(w|t,u)P(u) \\
\phi_2(w,t) &= \sum_{u} \phi_1(w,t,u) \\
\phi_3(w,t,u,y) &= \phi_2(w,t)P(t)P(v|t)P(y|v) \\
\phi_4(w,y) &= \sum_{t,v} \phi_3(w,t,u,y) \\
\phi_5(z,y,w,x) &= \phi_4(w,y)P(z|y,w,x)P(x) \\
\phi_6(z) &= \sum_{y,w,x} \phi_5(z,y,w,x)
\end{aligned}$$

The savings of variable elimination can be observed from these computations. The largest potentials we maintained were $\phi_3(w,t,u,y)$ and $\phi_5(z,y,w,x)$, which each had 4 variables and $2^4 = 16$ instantiations. This is a significant improvement over the inference by JPD approach, which would have required 7 variables and $2^7 = 128$ instantiations.

However, variable elimination suffers from a key limitation: It can only estimate one query at a time. Improvements to variable elimination have integrated alternative data structures to cache calculations to avoid re-calculating different queries on the same evidence [4]. However, if the evidence changes, then the cache must be discarded and the variable elimination process must be restarted. This limits the generality of the inference algorithm.

While some alternative methods discussed in this thesis incorporate variable elimination, we do not make use of it due to the aforementioned limitation.

2.5.3 Dependency Structure Compilation

One method of computing reusable, efficient inference queries in a BN is by converting the DAG of the BN into a Junction Tree (JT). The process is demonstrated below on the DAG pictured in Figure 2.3.

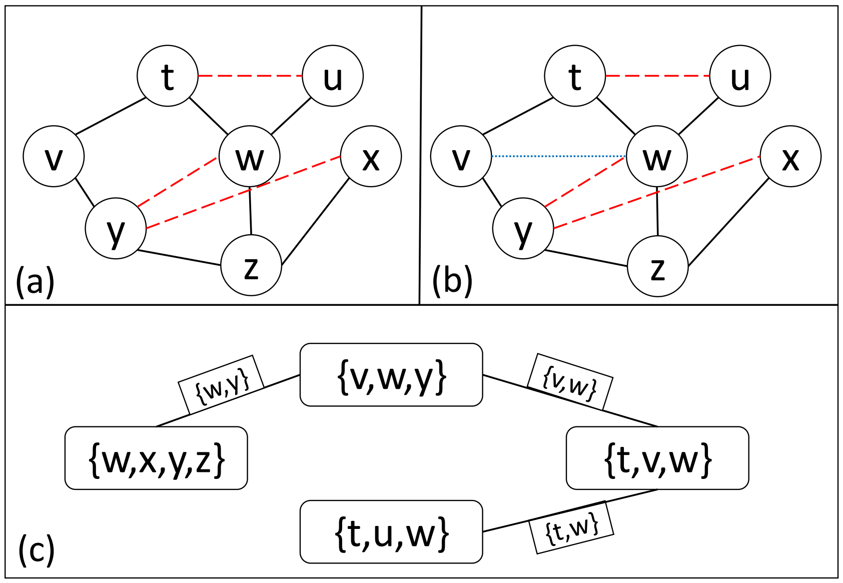


Figure 2.4: (a) Moralization of DAG in Figure 2.3. Edges added by moralization shown in red. (b) Triangulation. Edges added by triangulation shown in blue. (c) Generation of Junction Tree

We first construct the *moralized graph* G_M by transforming a BN's DAG into an undirected graph. The transformation consists of removing directions from each edge in the DAG to convert the directed edges into undirected edges. Then, all parents of each child node are connected by undirected edges. On the example DAG in Figure 2.3, we remove all directionality from the edges then connect all parents of each child node $\{\{t, u\}, \{y, w\}, \{y, x\}\}$. The resulting moralized graph is shown in Figure 2.4 (a) with red, dashed edges indicating the added edges.

Next, we convert the moralized graph into a triangulated graph G_T by breaking up all cycles that are longer than 3 nodes in length. Cycles are broken up by adding edges, such that every node amongst a cycle of size 4 or more has connected adjacent nodes. An example of triangulation is presented in Figure 2.4 (b).

The purpose of triangulation is to guarantee the existence of a junction tree. Formally, a *junction tree* [20] is a triplet (M, Ω, E), specified in terms of:

- M is a non-empty set of variables.
- Ω is a set of clusters such that all variables are contained in at least one cluster.
- E is the set of unordered edges connecting each cluster. Each edge is labelled by the intersection of the two clusters it is connecting. Two clusters Q_1 and Q_2 in Ω may be connected iff Q_1 is not equal to Q_2 and their intersection has at least one variable in common ($Q_1 \cap Q_2 \neq \emptyset$). The intersection $Q_1 \cap Q_2$ must be contained in every cluster on the path between Q_1 and Q_2.

In the context of junction trees, we refer to a *clique* as a clique from the triangulated graph, and use *cluster* and *node* interchangeably to denote a node in the junction tree.

To generate the junction tree, we begin with an empty graph G_{JT}. A node is added to G_{JT} for each clique in G_T, which is not fully contained within a larger clique. Each node is labelled by the variables in the clique. If two clusters have any variable(s) in common, they are included in every cluster on the path between them. Between adjacent clusters, each edge (separator) is labelled with the intersection of the two clusters. The resulting junction tree is shown in Figure 2.4 (c).

The advantage of a junction tree representation is it may be used for multiple inference queries and can incorporate different observations, as long as the topology (structure) of the BN does not change. Once the junction tree is compiled, we can then perform inference.

2.5.4 Inference by Message Passing

Message passing is an inference technique that passes messages over separators between adjacent clusters in the junction tree [1]. The technique is based off the concept of consistency.

Consistency *[20] Let $G_{JT} = (M, \Omega, E)$ be a junction tree representation. Let Q_1 and Q_2 be two adjacent clusters in Ω with the separator S. Let their associated potentials be $\phi(Q_1), \phi(Q_2), \phi(S)$, respectively. Clusters Q_1 and Q_2 are said to be* consistent *for some constants k_1 and k_2 if:*

$$\sum_{Q_1 \backslash S} \phi(Q_1) = k_1 \times \phi(S) = k_2 \times \sum_{Q_2 \S} \phi(Q_2)$$

The junction tree G_{JT} is said to be locally consistent *if every pair of adjacent clusters in* T *are consistent. It is said to be* globally consistent *if for any two clusters (not necessarily adjacent) Q_1 and Q_3, it holds for some constant k that:*

$$\sum_{Q_1 \backslash Q_3} \phi(Q_1) = k \times \sum_{Q_3 \backslash Q_1} \phi(Q_3).$$

Informally, for a junction tree to be locally consistent, the marginals of each pair of adjacent clusters must differ by a scalar multiple. Global consistency is a stricter variation that requires all marginals of each pair of nodes — adjacent or non-adjacent — to differ by only a scalar multiple. It follows that if a junction tree is locally consistent, then it must be globally consistent.

Inference by junction tree passes messages over separators with the objective of making the junction tree locally consistent. In order to make any two adjacent clusters consistent, the algorithm applies the *absorption* operation to update the separator and adjacent node's potentials.

Algorithm 2.1: Absorbtion

Let Q_1 and Q_2 be adjacent clusters with separator S in a junction tree.

Let their associated potentials be $\phi(Q_1), \phi(Q_2), \phi(S)$, respectively.

Q_1 absorbs from Q_2 by performing the following:

1. *Updating the separator potential $\phi(S)$ to $\phi(S)' = \sum_{Q_1 \setminus S} \phi(Q_1)$*
2. *Updating the clique potential $\phi(Q_2)$ to $\phi(Q_2)' = \phi(Q_2) \times \frac{\phi(Q_1)'}{\phi(Q_1)}$*

With absorption introduced, we can now begin summarizing the inference by junction tree algorithm [11, 20]. The initialization step of the algorithm consists of incorporating the CPDs into the junction tree. This involves converting the CPDs into potentials, and assigning each resulting potential to cliques that contain all variables in the potential. Multiple potentials may be assigned to the same cluster. Once all potentials have been assigned, we begin by setting each cluster's potential to the product of all potentials assigned to the cluster.

Next, we update a cluster's potentials for each observation. This is attained by setting the values of all entries inconsistent with the evidence to 0. We then arbitrarily select one node as the root node R. `CollectEvidence` is then recursively invoked on R to receive messages inwardly from the leaves to the root. When the `CollectEvidence` algorithm is invoked on a generic clique C_i, it invokes `CollectEvidence` on all other adjacent cliques $\{C_1, \ldots, C_m\}$. Once these cliques have finished collecting evidence, C_i absorbs from $\{C_1, \ldots, C_m\}$.

Algorithm 2.2: CollectEvidence

Let Q be a cluster in a junction tree G_{JT}.

A caller is either an adjacent cluster or the junction tree G_{JT} itself.

1. *Cluster Q invokes Collect Evidence on each adjacent cluster, except caller.*
2. *After each invoked cluster has completed, Q absorbs from it.*

Once `CollectEvidence` is complete, we distribute the updated evidence outwardly from the root to the leaves by calling the `DistributeEvidence` algorithm.

When invoked on a clique C_i from an adjacent clique C_j, the algorithm C_i absorbs from C_j and then invokes `DistributeEvidence` on all other adjacent cliques.

Algorithm 2.3: DistributeEvidence

```
Let $Q$ be a cluster in a junction tree $G_{JT}$.
A caller is either an adjacent cluster or the junction tree $G_{JT}$ itself.
  1. If caller is a cluster, $Q$ absorbs from it.
  2. Cluster $Q$ invokes DistributeEvidence on each adjacent cluster, except
     caller.
```

Figure 2.5 demonstrates the `CollectEvidence` and `DistributeEvidence` operations on a junction tree. The assigned CPTs are indicated in gray below each cluster. Cluster $\{v, w, y\}$ was arbitrarily chosen as the root cluster. CollectEvidence operations are shown with red arrows. DistributeEvidence operations are shown with blue arrows.

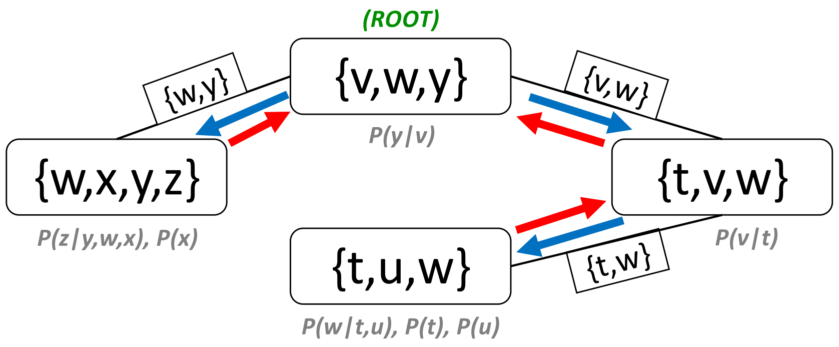


Figure 2.5: CollectEvidence (red arrows) and DistributeEvidence (blue arrows) on the junction tree from Figure 2.4. The assigned CPTs are indicated in gray below each cluster.

The efficiency of inference by message passing is dependent on the *treewidth* of the junction tree.

Treewidth Treewidth *of a junction tree is the size of the largest cluster minus 1.*

For example, the treewidth of the junction tree in Figure 2.5 is 3.

2.5.5 Lazy Propagation

Lazy propagation [11] is an improvement over traditional messaging passing. The key insight is that we do not need to multiply the potentials assigned to the clusters in the initialization step. Instead, we can defer the multiplication until it is required to do so. The result of deferring the multiplications is fewer calculations and faster inference at the cost of occupying more space. For further information, refer to [11].

2.6 Accuracy Metrics

A common method of evaluating the similarity between an approximation and its ground truth is by accuracy metrics. In this thesis, we make use of two accuracy metrics: *Euclidean distance* and *Kullback-Leibler* distance.

2.6.1 Euclidean Distance

Euclidean distance (ED) computes the straight line distance between two vectors. Since we are comparing CPTs, a CPT can be interpreted as a set of indexed CPDs, where each state in the variable's domain represents a dimension in vector space.

Given a ground truth CPT P_{GT}, let P_A represent a CPT that approximates P_{GT}, m represent the number of CPDs in the CPT, and n represent the number of probabilities in each CPD. The ED can be calculated as follows:

$$ED(P_{GT}, P_A) = \sqrt{\sum_{i=0}^{m} \sum_{j=0}^{n} \left(P_{GT}(i,j) - P_A(i,j)\right)^2}$$

The result indicates the similarity of the CPTs. The value is bounded by $[0, 1]$ where a value of 0 represents the CPTs are identical, and a value of 1 represents the CPTs are entirely different. A larger ED value indicates the CPTs differ whereas a smaller ED value indicates the CPTs are more similar. ED treats small differences equally to larger differences.

2.6.2 Kullback-Leibler Divergence

Kullback-Leibler divergence (or Kullback-Leibler distance, or KL distance) [9] computes the distance of the approximation probability distribution P_A from the ground truth probability distribution P_{GT}. The metric captures the randomness of P_A and magnifies the impact of large differences. It is non-symmetric such that the distance of the *ground truth from the approximation* is not equal to the distance of the *approximation from the ground truth.*

Given a ground truth CPT P_{GT}, let P_A represent a CPT that approximates P_{GT} where m represent the number of CPDs in the CPT and n represent the number of probabilities in each CPD. The KL distance can be calculated as follows:

$$KL(P_{GT}, P_A) = \sum_{i=0}^{m} \sum_{j=0}^{n} P_{GT}(i,j) \times \log \left(\frac{P_{GT}(i,j)}{P_A(i,j)} \right)$$

The result indicates the similarity of the CPTs. The KL distance is bounded within $[0, \infty)$, where a larger value indicates a greater difference between the distributions. A value of of 0 represents the distributions are identical (or very similar). A value ≥ 1 represents a large difference between the distributions.

2.7 Independence of Causal Influence Models

A tabular CPT ignores any relationships between parent and child variables resulting in the exponential space complexity on the number of parents. One way of addressing the exponential complexity is by making use of independence of causal influence models (also known as causal independence models). These models represent a child variable as a dependent of its parent variables, such that the parent variables are *causes* and the child variable is an *effect*. The key insight of causal models is that the models are capable of encoding each cause occurring independently to produce the effect. In order to compose a causal independence model, we first need to introduce a causal event.

Causal Event *An event representing a set of active causes either succeeding or failing to produce an active effect.*

To express a causal event, we follow the notation of Xiang [21]. When an active cause $c_i = true$ is binary, it is denoted by c_i^+ while an inactive binary cause is represented by c_i^-. An active binary causal event is represented by e^+ while an inactive binary causal event is denoted by e^-. A causal event of multiple causes successfully producing an effect can be denoted by $e^+ \leftarrow c_1,^+, c_2^+, \ldots, c_k^+$, while $e^+ \not\leftarrow c_1,^+, c_2^+, \ldots, c_k^+$ denotes that the causes failed to produce the effect e. A multi-valued cause is denoted by c_i^j where $i \geq 1$ represents the cause index and $j \geq 0$ represents the intensity of the cause. A multi-valued effect is denoted by e^j where j represents the intensity of the effect. The syntax $e \geq e_i$ (or, conversely $e \leq e_i$) is said to represent all effect values with an intensity value greater (less) than or equal to e_i. The probability of a causal event is denoted $P(e^+ \leftarrow c_1^+, c_2^+, \ldots, c_k^+)$.

The interaction between causes that produce a common effect may be characterized as reinforcing or undermining.

Reinforcing Interaction *[21] Causes which produce a common effect* reinforce *each other if collectively they are at least as effective as when some are active. Let c_1 and c_2 be two causes that produce an effect e.*

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

For instance, let the effect be a diagnosis of lung cancer, and causes of a diagnosis be smoking cigarettes and exposure to asbestos. Individually, smoking a cigarette or exposing oneself to asbestos may increase one's chance of obtaining lung cancer, but the presence of both together, is more harmful and thus, more likely to increase the chance of obtaining lung cancer.

Reinforcing interactions need not occur only between individual variables. The causal interaction can also occur between sets of variables. This allows for recursive mixtures. Two or more sets of variables reinforce each other if they satisfy failure conjunction and failure independence.

Failure Conjunction *[21] Let $R_1, R_2, \ldots, R_m$ be a disjoint set of causes that satisfy* failure conjunction *iff*

$$(e^+ \not\leftarrow \underline{r_1^+}, \ldots, \underline{r_m^+}) = (e^+ \not\leftarrow \underline{r_1^+}) \wedge \cdots \wedge (e^+ \not\leftarrow \underline{r_m^+}).$$

This specifies that the failure of the group of causes is determined by the conjunction of each causal failure event. In other words, for the joint causal event to fail to produce the effect, it requires that every single causal event fail to produce the effect.

Failure Independence *[21] Let $R_1, R_2, \ldots, R_m$ be a disjoint set of causes that satisfy* failure independence *iff*

$$P(e^+ \not\leftarrow \underline{r_1^+}, \ldots, \underline{r_m^+}) = P(e^+ \not\leftarrow \underline{r_1^+}) \times \cdots \times P(e^+ \not\leftarrow \underline{r_m^+}).$$

This specifies that the probability of the failure of the group is the product of the individual failure probabilities. Hence, reinforcement between sets requires that each cause fail individually and states the probability of a set of causal failures is their product.

By contrast, an interaction between causes may be characterized as undermining.

Undermining Interaction *[21] Causes which produce a common effect* undermine *each other if they are collectively less effective than some causes acting individually. Let c_1 and c_2 be two causes that produce an effect e.*

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

For example, let the effect be recovery from lung cancer, and the causes be two forms of treatments, which are known to inhibit each other. Individually each treatment may heal the lung cancer, but together, they counteract, reducing the likelihood of a person recovering from lung cancer.

Likewise, undermining interactions need not occur only between individual variables. The causal interaction can also occur between sets of variables. Two or more sets of variables undermine each other if they satisfy success conjunction and success independence.

Success Conjunction *[21] Let $R_1, R_2, \ldots, R_m$ be a disjoint set of causes that satisfy* success conjunction *iff*

$$(e^+ \leftarrow \underline{r_1^+}, \ldots, \underline{r_m^+}) = (e^- \leftarrow \underline{r_1^+}) \wedge \cdots \wedge (e^+ \leftarrow \underline{r_m^+}).$$

This specifies that the success of the group of sets is determined by each individual set's successes.

Success Independence *[21] Let $R_1, R_2, \ldots, R_m$ be a disjoint set of causes that satisfy* success independence *iff*

$$P(e^+ \leftarrow \underline{r_1^+}, \ldots, \underline{r_m^+}) = P(e^+ \leftarrow \underline{r_1^+}) \times \cdots \times P(e^+ \leftarrow \underline{r_m^+}).$$

This specifies that the probability of all active causes is the joint probability of each individual active cause.

The notion of reinforcing interactions and undermining interactions occurring at a set-level allows for a recursive mixture of interactions. For instance, two causal events $e^+ \leftarrow c_1^+$ and $e^+ \leftarrow c_2^+$ may reinforce each other, but together $e^+ \leftarrow c_1^+, c_2^+$, they may undermine a third causal event $e^+ \leftarrow c_3^+$.

2.7.1 Noisy-OR

Noisy-OR [12] is a causal independence model that encodes reinforcing interactions. It is restricted to binary variables but was later generalized to a *Noisy-MAX* model which can encode multi-valued variables [8]. The models reduces the complexity of a tabular CPT to linear on the number of causes.

Formally, let $C = \{c_1, \ldots, c_k\}$ be a set of uncertain causes that produces an effect e. Let D be a subset of the causes in C where each cause in D actively produces e. The subset D may or may not be equal to C. A Noisy-OR model represents the causal interactions among C by the following property:

$$P(e^+ \leftarrow c_1^+, \ldots, c_k^+) = \begin{cases} 1 - (P(e^+ \not\leftarrow c_1^+) \times \cdots \times P(e^+ \not\leftarrow c_k^+)) & \text{if } D \neq \emptyset \\ 0 & \text{if } D = \emptyset \end{cases}$$

This property can be explained by decomposing it into two possible cases: no causes are active ($D = \emptyset$), or at least one cause is active ($D \neq \emptyset$). If no causes are active, then the effect must be inactive. This satisfies the failure conjunction property. If at least one cause is active, then the effect is inactive if *all* active causes fail to produce the effect; otherwise, the effect is active. The resultant causal event has a probability equal to the product of the causal failures. This satisfies the failure independence property.

However, neither the Noisy-OR nor the Noisy-MAX are able to model undermining interactions. This is a critical limitation that precludes the use of these models in this thesis.

2.7.2 Noisy-AND

The *Noisy-AND* is a causal independence model that encodes reinforcing interactions over binary variables.

Formally, let $C = \{c_1, \ldots, c_k\}$ be a set of uncertain causes that produces an effect e. Let D be a subset of the causes in C where each cause in D actively produces e. The subset D may or may not be equal to C. A Noisy-AND model represents the causal interactions among C by the following property:

$$P(e^+ \leftarrow c_1^+, \ldots, c_k^+) = \begin{cases} P(e^+ \leftarrow c_1^+) \times \cdots \times P(e^+ \leftarrow c_k^+) & \text{if } D = C \\ 0 & \text{if } D \subset C \end{cases}$$

In plain language, if all causes are true, then the probability of the active effect is the product of all active causes. If any cause is inactive, then the probability of the active effect is zero. The behaviour of the Noisy-AND gate is said to be *impeding* since a single inactive cause prohibits an active effect. This prevents the Noisy-AND model from encoding undermining interactions.

2.7.3 Non-Impeding Noisy-AND Trees

The previously discussed causal independence models encode reinforcing interactions and reduce the number of parameters to linear on the number of causes. However, each of these models have a significant limitation: they cannot encode undermining interactions.

Non-impeding Noisy-AND Tree (NIN-AND Tree, or NAT) is a causal independence model that can represent both reinforcement and undermining relationships. The model represents the interactions graphically, using a recursive mixture of two types of NIN-AND gates, *dual NIN-AND gates* and *direct NIN-AND gates.* Both gate types accept causal events as inputs and produce an output causal event, with a probability specified by the product of input causes' probabilities.

Dual Gate

The dual gate operates similarly to a noisy-OR gate by modelling reinforcing relationships between causes. The inputs and outputs of a dual gate are causal failure events. The dual gate satisfies failure conjunction and failure independence properties. The failure conjunction is expressed graphically by the AND gate and the failure independence is expressed by the lack of connection between the causal

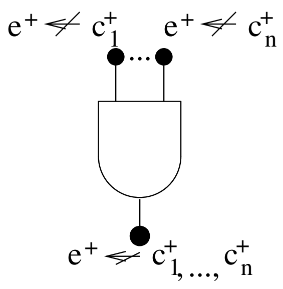

$e^+ \not\leftarrow c_1^+$
$e^+ \not\leftarrow c_n^+$
$e^+ \not\leftarrow c_1^+, \ldots, c_n^+$

Figure 2.6: An example dual gate with n input causal failure events.

failure events. For example, the dual gate pictured in Figure 2.6 accepts n causal failure events as inputs: $e^+ \not\leftarrow c_1^+, \ldots, e^+ \not\leftarrow c_k^+$ and produces the output causal event $e^+ \not\leftarrow c_1^+, \ldots c_n^+$.

Direct Gate

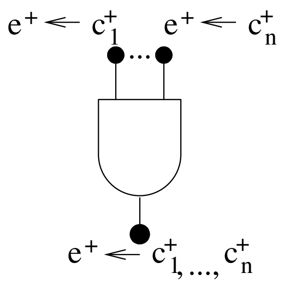


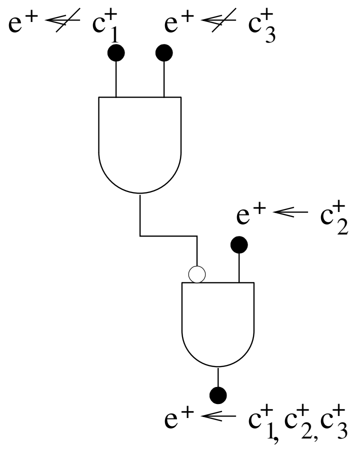
Figure 2.7: An example direct gate with n input causal success events.

The direct gate operates similarly to a noisy-AND gate, with two key differences. First, the direct gate models undermining interactions whereas the Noisy-AND models reinforcement. More specifically, the direct gate operates on causal success events and implements the success conjunction and success independence properties. The success conjunction is expressed graphically by the AND gate and the success independence is expressed by the lack of connection between the causal success input events. Second, the direct gate is non-impeding, such that the presence of an inactive cause does not force the output event of the direct gate to be inactive as well. As an example of a direct gate, consider Figure 2.7, which has n causal successes as inputs: $e^+ \leftarrow c_1^+$,

$e^+ \leftarrow \ldots, c_n^+$ and produces the output causal event $e^+ \leftarrow c_1^+, \ldots, c_n^+$.

Recursive Combinations of NAT Gates

A recursive mixture of dual and direct gate can express complex relationships between causes that produce an effect. This is achieved by connecting the output of one gate as the input of another gate. A connection between gates may be *negated*, denoted by a white dot, to negate the output of the upper gate before it is passed as input to the lower gate.



$e^+ \not\leftarrow c_1^+$
$e^+ \not\leftarrow c_3^+$
$e^+ \leftarrow c_2^+$
$e^+ \leftarrow c_1^+, c_2^+, c_3^+$

Figure 2.8: An example NAT with one dual gate (upper) and one direct gate (lower).

A complex NIN-AND is shown in Figure 2.8. The NAT consists of one dual gate and one direct gate. The upper dual gate accepts two single causals as inputs ($e^+ \not\leftarrow c_1^+$ and $e^+ \not\leftarrow c_3^+$) and outputs the causal event $e^+ \not\leftarrow c_1, c_3$. The lower direct gate accepts inputs of the negation of the causal event from the above dual gate ($e^+ \leftarrow c_1, c_3$) along with the single causal ($e \leftarrow c_2^+$). The lower direct gate outputs the causal event $e^+ \leftarrow c_1^+, c_2^+, c_3^+$.

To demonstrate how the probability of a causal event is retrieved from a NIN-AND tree, consider the following single-causal probabilities specified for the example NAT topology in Figure 2.8:

$$P(e^+ \leftarrow c_1^+) = 0.8 \qquad P(e^+ \leftarrow c_2^+) = 0.4 \qquad P(e^+ \leftarrow c_3^+) = 0.6$$

We can compute the causal event produced by the upper dual gate as the product of the negated single-causals ($e^{\leftarrow}c_1^+$ and $e^+ \leftarrow c_3^+$):

$$\begin{aligned}
P(e^+ \not\leftarrow c_1^+, c_3^+) &= P(e^+ \not\leftarrow c_1^+) \times P(e^+ \not\leftarrow c_3^+) \\
&= (1 - P(e^+ \leftarrow c_1^+)) \times (1 - P(e^+ \leftarrow c_3^+)) \\
&= (1 - 0.8) \times (1 - 0.6) \\
&= 0.08
\end{aligned}$$

We can then compute the causal event produced by the lower direct gate as the product of the negated output from the dual event and the single-causal

$$\begin{aligned}
P(e^+ \leftarrow c_1^+, c_2^+, c_3^+) &= P(e^+ \leftarrow c_1^+, c_3^+) \times P(e^+ \leftarrow c_2^+) \\
&= (1 - P(e^+ \not\leftarrow c_1^+, c_3^+)) \times P(e^+ \leftarrow c_2^+) \\
&= (1 - 0.08) \times 0.4 \\
&= 0.368
\end{aligned}$$

NAT-Modelled Bayesian Networks

A *NAT-modelled Bayesian network* is a Bayesian network where all local distributions are modelled by NATs instead of tabular CPTs. By contrast, a *normal Bayesian network* is a Bayesian network where all local distributions are modelled by tabular CPTs.

Inference methods designed for normal Bayesian networks cannot be applied to NAT-modelled Bayesian networks without normalization. *Normalization* is the process by which each NAT model is expanded into a tabular CPT. This results in a local structure with a size exponential on the family size, thereby discarding all space savings of the NAT model.

2.7.4 Inference by Multiplicative Factorization

Multiplicative factorization is an alternative inference method that preserves some of the savings yielded by the NAT model. It was originally designed by Takawa and D'Ambrosio [17] for the Noisy-OR and Noisy-MAX causal independence models before being extended to NIN-AND by Xiang and Jin [24]. The method works by factorizing a NIN-AND gate in a NAT model (Figure 2.9 (a)) into a new *hybrid graph* (Figure 2.9 (b)) with both directed and undirected edges.

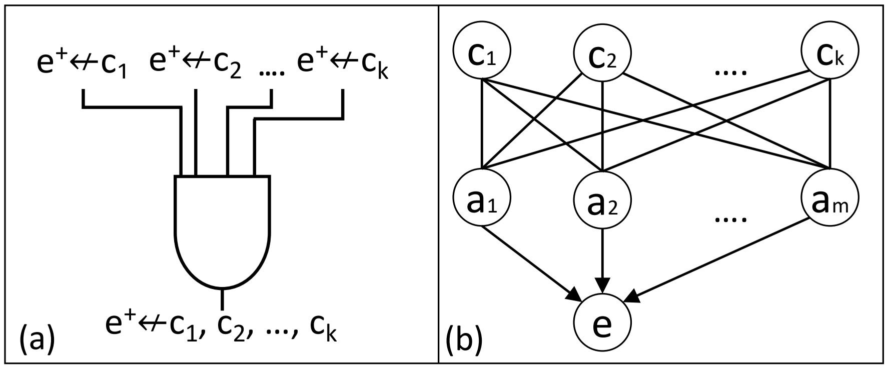


Figure 2.9: (a) Dual gate with k causes. (b) Multiplicative factorization of (a).

The graph consists of one node c_i per cause in the NAT, a node e for the effect e, and one auxillary node a_j for each active value in the effect domain $dom(e)$. The graph is connected as follows: Each cause node c_i is connected to each auxillary node a_i by an undirected edge, and each auxillary node is connected to the effect node e by a directed edge. Each undirected edge is assigned a potential $f(a_j, c_i)$, and the effect node e is assigned a potential $f(e, a_1, \ldots, a_m)$. The hybrid graph can subsequently be compiled into a (lazy) junction tree for inference. Refer to [24] for further details on multiplicative factorization of NAT models.

Multiplicative factorization improves on normalization by preserving some of the causal independencies. It has been shown to improve inference time by up to two orders of magnitude in certain networks [24]. However, if the effect domain becomes large, the fully-connected nature of the auxiliary variables will result in an exponential explosion, thereby limiting multiplicative factorization to smaller effect domains.

2.7.5 Inference by De-causalization

De-causalization is an alternative NAT inference method with the aim of avoiding multiplicative factorization's exponential explosion. The method involves replacing each NAT gate (Figure 2.10 (a)) with a BN segment (Figure 2.10 (b)) quantified by tabular local models that encodes an equivalent CPT. The resulting BN segment consists of ordinary events while maintaining some of the NAT's computational savings. The generated BN segment consists of a node c_i for each cause in the NAT gate, one auxillary node d_j for each cause in the NAT, and an effect node e. The auxillary nodes are connected in a one-in-one-out fashion where their parent is a cause node and their child is the effect node. A tabular local model is then assigned to each node. Some are probabilistic, while others are deterministic. This process is repeated for each NAT gate in a NAT-modelled BN. For further information on the conversion process, refer to [25].

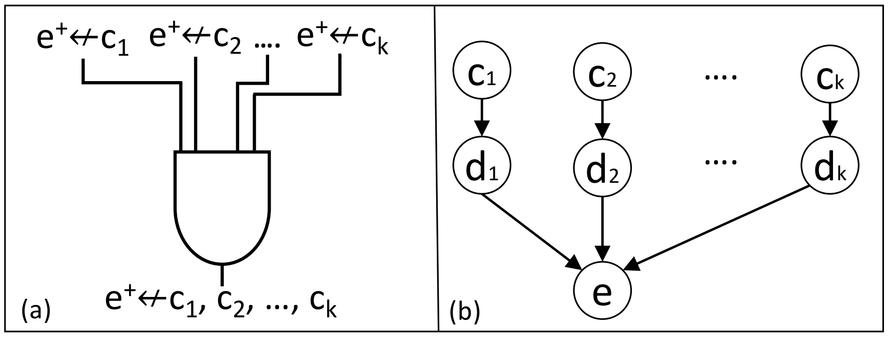


Figure 2.10: (a) Dual gate with k causes. (b) De-causalization of (a).

De-causalization improves on normalization by preserving some of the causal independencies and multiplicative factorization by eliminating the exponential explosion. It has been shown to improve inference efficiency by up to two orders of magnitude in sparse networks [25].

2.7.6 Existence of NAT Models in Real-World BNs

It is necessary to confirm the existence of NAT models in real-world BNs. If the existence is not proven, then NAT models would be restricted to synthetic or expert-specified BNs. Thus, it is critical we establish that NAT models exist in real-world BNs, which were not originally designed to be NAT modelled. The previous work of Xiang and Baird [22] positively identified the existence of NAT models by experiment.

In this section, we summarize their approach and findings. Their experiment began by sourcing 8 real-world BNs from a popular BN repository, *bnlearn*. On each BN, they applied NAT compression to convert certain tabular CPTs into NAT models. Tabular CPTs with less than 2 parents or where the node's CPT is deterministic were not compressed. The resultant NAT modelled BN is denoted a NMBN.

The BNs selected are summarized in Table 2.1. Column 1 indicates the network name. Column 2 indicates the number of nodes in the BN. Columns 3 and 4 indicate the number and percentage of nodes that were NAT modelled. Column 5 indicates the number of edges in the BN. Column 6 indicates the density of the BN: % of links added to a singly connected network.

Both the source BN and the NMBN were compiled for inference using the lazy junction tree approach. Ten rounds of inference with different sets of evidence were performed. The NAT's modelling of real-world BNs compared the KL and ED distances of the posterior marginals computed by the source BN against the NMBN. The results are presented in Figure 2.11. The previous work observed that the distances are reasonably small and that the posterior marginals are reasonably accurate, given that $30-50\%$ of the families are NAT modelled. Notably, they also discovered that the posterior error was smaller than the NAT compression error indicating that compression errors are weakened, rather than exacerbated, by the inference.

Network	# of Nodes	# NAT CPTs	% NAT CPTs	# of Links	w
Alarm	37	16	43.2	46	28
Andes^{-}	220	106	48.2	338	54
Barley	48	28	58.3	84	79
Child	20	6	30	25	32
Hepar2	70	33	47.1	123	78
Insurance	27	8	29.6	52	100
Win95pts	76	8	10.5	112	49
Munin	1041	11	1.1	1397	34

Table 2.1: NAT modelling was confirmed by evaluating its existence on these BNs from the *bnlearn* repository. The Andes^{-} BN is the Andes BN with 3 isolated nodes removed. Table reprinted from [22].

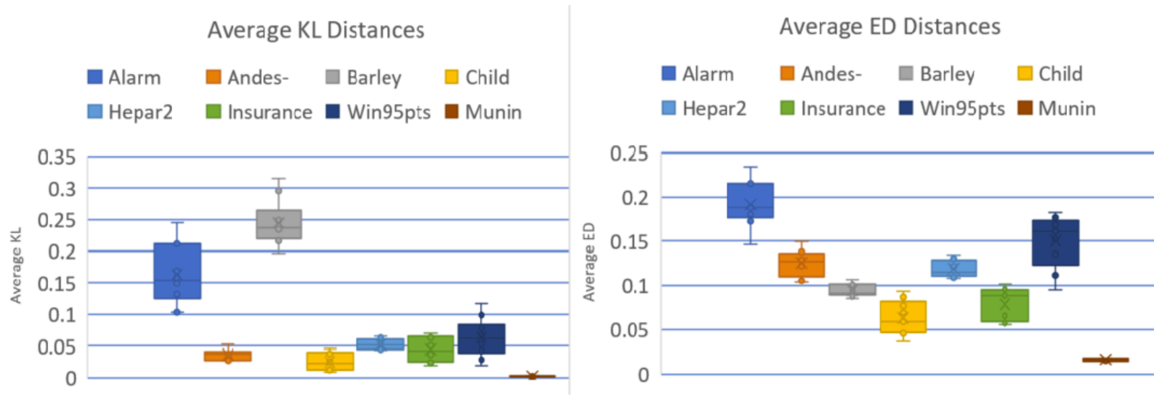


Figure 2.11: KL and ED Distances from comparison of posterior marginals of 8 real-world BNs. Figure reprinted from [22].

2.8 Context-Specific Independence

Causal independence models are not the only way to address the exponential complexity of tabular CPTs. An alternative approach is *context-specific independence* (CSI), which exploit relationships between the child variable and some of its parent variables.

Prior to introducing CSI, it is first necessary to introduce the notion of a *context*.

Context *Let n be a generic node in a Bayesian network. A* context *is an assignment of value(s) to a subset of n's parents $\pi(n)$.*

Consider the variable z in Figure 2.3. Valid contexts include (but are not limited to) $\{w = \{w_0\}\}$ and $\{w = \{w_0\}, y = \{y_0, y_1\}\}$. We can now introduce context-specific independence.

Context-specific Independence *For disjoint sets of variables A, B, Z, and a non-empty context Cxt, A and B are said to be* contextually independent *given Z and context $Cxt = cxt$, denoted $I_c(A; B|Z, Cxt = cxt)$, iff*

$$P(A|B, Z, cxt) = P(A|Z, cxt) \text{ whenever } P(B, Z, cxt) > 0.$$

For example, two CSI interactions exist in the CPT $P(z|y, w, x)$ shown in Figure 2.3. The first occurs when $(y = y_0, w = w_0)$, the probabilities of $P(z|y, w, x)$ are the same, regardless of the value of x. This interaction can be denoted $I_c(z; x|y = y_0, w = w_0)$. The second interaction occurs when $(y = y_1)$, the probabilities of $P(z|y, w, x)$ are the same, regardless of the values of w and x. This interaction can be denoted $I_c(z; w, x|y = y_1)$.

Another way to think of CSI is to consider a real-world example. Suppose we are modelling an individual's recovery from surgery, depending on whether or not they use physiotherapy and the skill of the physiotherapist. If the patient completes physiotherapy, then the likelihood of recovery increases. The magnitude of the increase is dependent on the skill of the physiotherapist. A highly skilled physiotherapist will result in a significant increase to the likelihood of recovery, whereas a less skilled physiotherapist will result in a smaller increase to the likelihood of recovery. On the other hand, if the patient declines physiotherapy, then the likelihood of recovery decreases.

Assuming each variable is binary, a tabular CPT would represent this environment with $2^2 = 4$ instantiations: one for each possible instantiation of values. However, it is observed that when the patient declines physiotherapy, the likelihood of recovery decreases, regardless of the skill of the physiotherapist. Thus, the model exhibits the CSI interaction where the patient's recovery is contextually independent of the skill of the physiotherapist when the patient declines physiotherapy. A CSI model can exploit this interaction to reduce the number of parameters needed.

In the following sections, we discuss the various representations of CSI and a compatible method to support inference for each.

2.8.1 Default Tables & Normalization

Default tables are a CSI model that improve on tabular CPTs by only explicitly representing a subset of the instantiations on the parent variables. Formally, a default table [6] is specified by a a triplet (M, P, Csi) where M is a set of variables, P is the set of all CPDs over the variables M, and Csi is a single CSI interaction over the variables M. The CSI interaction Csi partitions P into two segments. The first segment contains all CPDs whose instantiations satisfy Csi. The second segment contains all remaining CPDs in P. The default table encodes this partitioning of P as follows: Each CPD in the first segment is consolidated into a single row (denoted by an $*$), while the CPDs in the second partition are expressed with no change.

			$P(z\|y,w,x)$	
y	w	x	z_0	z_1
y_0	w_0	x_0	0.9	0.1
y_0	w_0	x_1	0.9	0.1
y_0	w_1	x_0	0.2	0.8
y_0	w_1	x_1	0.1	0.9
	$*$		0.4	0.6

Figure 2.12: Default table for $P(z|y, w, x)$

In Figure 2.12, the CSI interaction $I_c(z; w, x|y = y_1)$ is encoded by explicitly representing all instantiations that satisfy the interaction by the default row (denoted by $*$), while all other rows remain unchanged.

However, default tables suffer from two critical limitations. First, a default table can only express one CSI interaction. Any other CSI interactions must be explicitly represented. For instance, the CSI interaction $I_c(z; x|y = y_0, w = w_0)$ in Figure 2.12 cannot be exploited and must be explicitly encoded. Second, there does not appear to be any efficient inference methods for default tables. To perform inference on a default table, one must expand the default table into a tabular CPT. The tabular CPT is then compatible with any common BN inference algorithm. This is an insufficient as all representational savings from the default table will be discarded by the expansion. Hence, the limitations preclude the use of default tables in this thesis.

2.8.2 Rule-based Representation & Variable Elimination

A rule-based CPT is a set of rules of the form $\alpha \mid Cxt : P(\alpha \mid Cxt)$ where α is a variable, Cxt is context and $P(\alpha \mid Cxt)$ is the CPD specified by its parameters[1]. The context is encoded as a logical sentence where each assignment in the context is conjuncted together. Each instantiation must be encoded by a rule but many instantiations can be encoded by the same rule.

CSI can be exploited by applying operations on the sentences, which simplify the rules. One such operation is the *combination operation*, which consolidates rules that correspond to identical CPDs. Let $\alpha \mid Cxt_1 : P(\alpha \mid Cxt_1)$ and $\alpha \mid Cxt_2 : P(\alpha \mid Cxt_2)$ be two rules in the same rule base such that $P(\alpha \mid Cxt_1) = P(\alpha \mid Cxt_2)$. The combination operation would replace the two rules with a new rule holding the intersection of Cxt_1 and Cxt_2:

$$\alpha \mid Cxt_1 \cap Cxt_2 : P(\alpha \mid Cxt_1)$$

Additional operations include the *split* operation, and an extension of a potential's *product* and *marginalization* operations to rules. The details of these rules are omitted due to space considerations. Refer to [13] for further details.

[1]The rule-based representation's notation has been modified from [13] to distinguish it from the causal event notation.

$$z \mid y = y_0 \wedge w = w_0 : 0.9$$
$$z \mid y = y_0 \wedge w = w_1 \wedge x = x_1 : 0.2$$
$$z \mid y = y_0 \wedge w = w_1 \wedge x = x_1 : 0.1$$
$$z \mid y = y_1 : 0.4$$

Figure 2.13: Rule-based representation of the CPT $P(g|f, d, e)$ in Figure 2.3.

The CPT $P(z \mid y, w, x)$ is expressed in a rule-based form in Figure 2.13. The combination operation was performed on each CSI interaction. For instance, four rules with matching CPDs that all contain the context $(y = y_1)$ were combined into one rule $z \mid y = y_1 : 0.4$.

The marginalization and product rules allows for variable elimination to be extended to a set of rules [13]. Unfortunately, variable elimination appears to be the only inference method defined for the rule-based representation. As stated in the introduction to variable elimination on CPTs (Section 2.5.2), this thesis avoids its use due to its inability to incorporate new evidence at runtime without re-compilation.

2.8.3 CPT-tree & Network Transformation

A *CPT-tree* is a CSI model introduced by Boutillier et al. [3], which uses a tree structure to specify the CPT for a variable x conditioned on its parents $\pi(x)$. The tree is directed from the root (node with no parents) to the leaves (node with no children). The tree structure restricts each node to have at most one parent resulting in a single path from the root node to each node in the tree.

Each non-leaf node is a variable in $\pi(x)$. Each outgoing edge from a node is labelled by a value the variable holds[2]. The *path* from the root node to a leaf node encodes a context. Each leaf node encodes a CPD of x, given the context. We note

[2]In Section 4.1, we generalize the labelling of edges to allow an edge to encode multiple values.

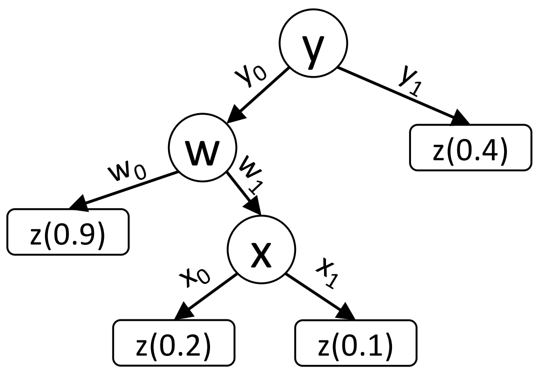


Figure 2.14: CPT-tree for $P(g|f, d, e)$.

that a CPT-tree *is not* a decision tree. While the semantics and visual appearance are similar; a CPT-tree exists in an uncertain environment whereas a decision tree exists in a deterministic environment.

The CPT-tree for the tabular CPT $P(z|y, w, x)$ is shown in Figure 2.14. The CSI interactions are observable when paths do not include all variables in $\pi(x)$. Specifically, the CSI interaction $I_c(z; w, x|y = y_1)$ is encoded by the right-most branch from the root node f. The other CSI interaction $I_c(z; x|y = y_0, w = w_0)$ is encoded by the left-most branch. Each CPD can be retrieved from the label of the leaf node. For instance, the CPD $P(z|y = y_0, w = w_1, x = x_1)$ can be retrieved by following the left path from the root node ($y = y_0$), then the right node ($w = w_1$), and finally, its right child ($x = x_1$). Hence, a CPT-tree can be *normalized* to an exponentially sized tabular CPT by iterating through each instantiation of the parent variables in the CPT-tree and retrieving the appropriate CPD.

We refer to BNs where some families are modelled by CPT-trees as *CPT-tree-modelled BNs.* There are various inference methods that support inference with CPT-tree-modelled BNs, including network transformation and clustering [3], cutset conditioning [3], and variable elimination [13]. In this thesis, we will focus on network transformation and clustering.

Network transformation and clustering consists of replacing each CPT-tree mod-

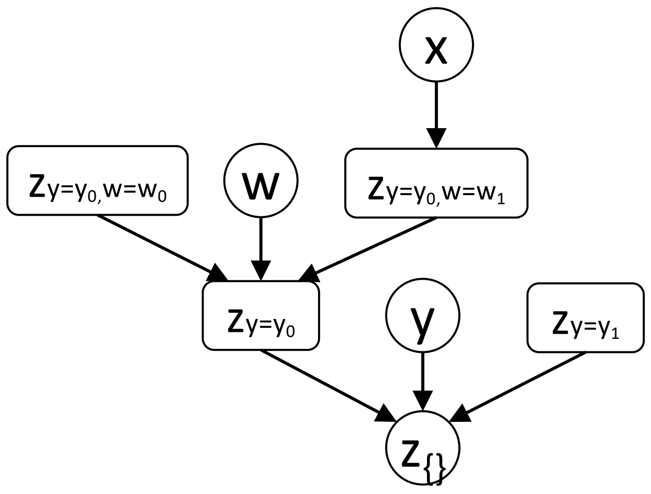


Figure 2.15: BN segment generated from network transformation of the CPT-tree in Figure 2.14.

elled family (Figure 2.14) with a BN segment (Figure 2.15) quantified by tabular local models that encodes an equivalent CPT while preserving some of the context-specific independencies. The resulting BN is then compatible with any standard BN inference algorithm.

The initial idea of this algorithm was proposed by [3] but they did not present a formal algorithm. A contribution of this thesis, presented in Chapter 4, is to formalize an algorithm suite for this method. We omit further discussion on this method until that section.

Chapter 3

Orthogonality of NAT & CSI Models

This chapter confirms the necessity of the novel contribution introduced in our work. The structure of the chapter is as follows. Section 3.1 introduces the premise and motivation of testing for orthogonality. Section 3.2 discuss how a CSI model can be approximated by a NAT model. Section 3.3 outlines how a NAT model can be approximated by a CSI model. We test the orthogonality of NAT and CSI model in two parts. Section 3.4 investigates if NAT models are able to encode CSI CPTs, while Section 3.5 explores if CSI models are able to encode NAT CPTs.

3.1 General Information on Orthogonality

Two models are *orthogonal* if neither model is able to efficiently and exactly encode the other. Otherwise, the models are said to be *not orthogonal.* To illustrate the concept of orthogonality, consider the following two examples.

i) Rational and irrational numbers *are orthogonal* representations of real numbers. The reason for this is a rational number cannot represent an irrational number, and conversely, an irrational number cannot represent a rational number. Consider the irrational number π; there does not exist any rational number composed of two integers a and b such that $\frac{a}{b} = \pi$. Likewise, there does not exist any irrational number that can represent the rational number $\frac{3}{2}$. Hence, the models are orthogonal since neither representation can exactly encode the other.

ii) The Noisy-OR and NIN-AND models *are not orthogonal*, since the Noisy-OR model is a special case of the NIN-AND model. The rationale for this can be demonstrated in two parts. It is first necessary to show that there exists a NIN-AND model that cannot be encoded by a Noisy-OR. This can be proven by the fundamentals: A NIN-AND model encodes both reinforcing and undermining interactions, while a Noisy-OR model only encodes reinforcing interactions. Hence, any NIN-AND model containing an undermining interaction cannot be expressed by a Noisy-OR. Likewise, we must demonstrate that a NIN-AND model can efficiently and exactly encode every Noisy-OR model. While a formal justification for this exists in [23], for simplicity purposes, we present a sufficiently general example. Consider a BN family with a child variable e that is dependent on three parents c_1, c_2 and c_3. All variables are binary since the Noisy-OR is restricted to binary variables. All causes reinforce each other. In Figure 3.1, we show that the CPT generated by the Noisy-OR is equivalent to the CPT specified by a dual NIN-AND gate. Hence, the models are not orthogonal since there exist NIN-AND models that cannot be encoded by the Noisy-OR models and every Noisy-OR model is a special case of the dual NIN-AND gate.

$P(e^+ \leftarrow c_1^+) = 0.9$ $\quad P(e^+ \leftarrow c_2^+) = 0.8$ $\quad P(e^+ \leftarrow c_3^+) = 0.7$

			$P(e^+\|c_1, c_2, c_3)$	
c_1	c_2	c_3	by Noisy-OR	by NIN-AND
T	F	F	0.90 (given)	0.90 (given)
F	T	F	0.80 (given)	0.80 (given)
F	F	T	0.70 (given)	0.70 (given)
T	T	T	0.994	0.994
T	T	F	0.98	0.98
T	F	T	0.97	0.97
F	T	T	0.94	0.94
F	F	F	0	0

Figure 3.1: Binary-valued CPT $P(e^+|c_1^+, c_2^+, c_3^+)$ where all causes reinforce each other demonstrating that a Noisy-OR can be encoded by the NIN-AND gate.

To resolve any potential ambiguities, the notion of orthogonality presented in this work differs from the geometric interpretation of orthogonality. In geometry, two vectors are orthogonal if they are perpendicular at their point of intersection. When two vectors $\overrightarrow{q}, \overrightarrow{r}$ are orthogonal, their dot product is 0 ($\overrightarrow{q} \cdot \overrightarrow{r} = 0$), and thus the length of the vector projection of $\overrightarrow{q}$ onto $\overrightarrow{r}$ is 0, and equivalently, $\overrightarrow{r}$ projected onto $\overrightarrow{q}$ is 0. It can be interpreted that neither of the orthogonal vectors $\overrightarrow{q}, \overrightarrow{r}$ can provide any approximation of the other. By contrast, in this work, the concept of orthogonality only requires that neither model can efficiently and exactly represent the other. It does not require that the models be unable to approximate each other.

Moreover, it is of critical importance to the necessity of this research that we demonstrate the NAT and CSI models are orthogonal. If they are *not orthogonal*, then the necessity of this research is nullified since an inference method designed for the more general model could be applied to both classes of local models. But, if the models *are orthogonal*, then a novel method is needed to support inference on a BN modelled with both NAT and CSI local models.

Before testing for the orthogonality of the NAT and CSI models, it is first necessary to demonstrate how one model may be converted into the other. In this chapter, we make use of the CPT-tree CSI representation; however, the conversion can be trivially applied on any CSI representation.

The following two sections demonstrate the conversion in both directions: Section 3.2 shows the conversion from a CSI model to a NAT model. Section 3.3 shows the conversion from a NAT model to a CSI model.

3.2 Converting from a CSI Model to a NAT Model

We demonstrate the conversion from a CSI model to a NAT model by example. Since there does not exist a direct conversion method between NAT and CSI, we must use an intermediary representation to facilitate the conversion. The intermediary representation must support conversion from a CPT-tree to the intermediary representation, and conversion from the intermediate representation to a NAT model.

Thus, a tabular CPT is selected as the intermediary representation since it satisfies both properties.

Suppose, we have the CPT-tree in Figure 3.2 specifying the CPT of a family with the child variable z and three parents w, x, y. All variables are binary. The CPT-tree admits 1 CSI interaction: $I_C(z; w|y, x = x_1)$, which states that when $x = x_1$, z is contextually independent of w for each y.

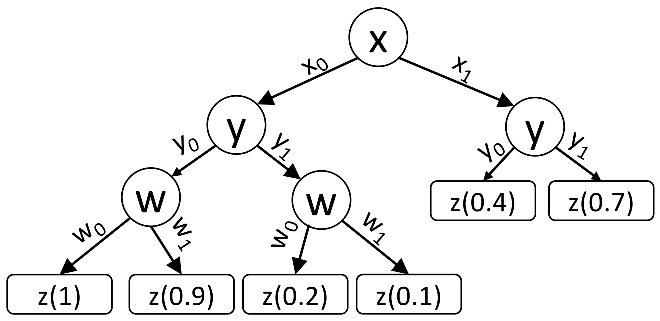


Figure 3.2: Binary CPT-tree encoding the CSI interaction $I_C(z; w|y, x = x_1)$.

3.2.1 Normalization of a CPT-tree to a Tabular CPT

The next step is to normalize the CPT-tree to a tabular CPT. The resultant tabular CPT is presented in Figure 3.3. We note the values encoded in the resultant CPT are exactly identical to the values expressed by the CPT-tree.

3.2.2 Compression of a CSI CPT to a NAT Model

We can now apply compression on the tabular CPT to convert the tabular CPT into a NAT model. Let the 0^{th} index of a variable's value be the inactive state (e.g., $w_0 = w^-$) and the 1^{st} index of a variable's value (e.g., $w_1 = w^+$) be the active state. The resultant NAT model is shown in Figure 3.4 with the following single-causals:

Overall, this process allows for the conversion of an existing CPT-tree into a

			$P(z \mid x, y, w)$	
x	y	w	z_0	z_1
x_0	y_0	w_0	1	0
x_0	y_0	w_1	0.9	0.1
x_0	y_1	w_0	0.2	0.8
x_0	y_1	w_1	0.1	0.9
x_1	y_0	w_0	0.4	0.6
x_1	y_0	w_1	0.4	0.6
x_1	y_1	w_0	0.7	0.3
x_1	y_1	w_1	0.7	0.3

Figure 3.3: Tabular CPT from the CPT-tree in Figure 3.2.

$P(z^+ \leftarrow x^+) = 0.6$ $\quad$ $P(z^+ \leftarrow y^+) = 0.8$ $\quad$ $P(z^+ \leftarrow w^+) = 0.1$

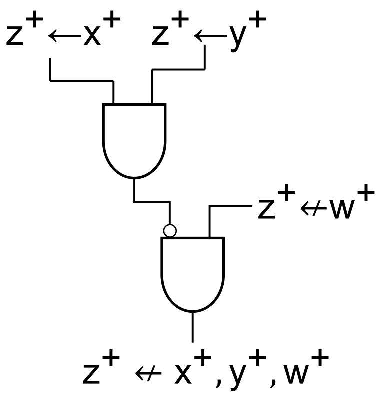


Figure 3.4: NAT model from the tabular CPT in Figure 3.3

NAT model. We omit a comparison of the resultant NAT model against the source CPT-tree since a more comprehensive evaluation is presented in Section 3.4. In the next section, we discuss the inverse of converting a NAT model to a CSI model.

3.3 Converting from a NAT Model to a CSI Model

In this section, we outline a method that estimates the number of parameters needed to define a CSI model. While this suffices for our experiments, we leave the learning of a CPT-tree from a tabular CPT as future work.

By definition, a CPT exhibits CSI if there are duplicated CPDs. If there are no duplicated CPDs, then a CSI representation's space complexity will degrade to a size exponential on the number of parents. Since a NAT is capable of representing CPTs without duplicated CPDs efficiently, it is necessary to weaken the criterion of CSI to similar probabilities. In other words, if two probabilities are very similar, a CSI model can encode both probabilities as a single value with a small error.

3.3.1 Normalization of a NAT Model to a Tabular CPT

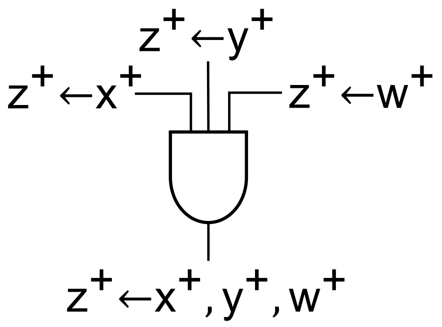
Suppose, we would like to convert the NAT in Figure 3.5 (a) to a CSI model. The NAT model consists of 3 binary causes x, y, w that produce an effect e. All three causes undermine each other. Let the 0^{th} index of a variable's value be the inactive state (e.g., $w_0 = w^-$) and the 1^{st} index of a variable's value (e.g., $w_1 = w^+$) be the active state.

Since there does not exist a direct conversion method, we will use a tabular CPT as an intermediary representation. Thus, the first step is normalize the NAT model into a tabular CPT. The resultant CPT is represented in Figure 3.5 (b). We note the values encoded in the resultant CPT are exactly identical to the values expressed by the NAT model.

3.3.2 Clustering a CPT to Estimate Number of CPT-Tree Parameters

Clustering is the process of grouping similar objects together such that the objects in the same cluster (group) have similar properties. We can apply clustering on a CPT's probabilities to estimate the number of parameters a CSI model would need to approximately encode a NAT CPT. More specifically, if we apply clustering on a binary NAT CPT, each cluster would represent a node in a CPT-tree, and the total number of clusters would represent the total number of parameters required to specify the CPT. Thus, if the clustering results in tight clusters (small distance

$P(z^+ \leftarrow x^+) = 0.39$ $P(z^+ \leftarrow y^+) = 0.40$ $P(z^+ \leftarrow w^+) = 0.41$



(a)

			$P(z\|x,y,w)$	
x	y	w	z_0	z_1
x_0	y_0	w_0	1	0
x_0	y_0	w_1	0.59	0.41
x_0	y_1	w_0	0.6	0.4
x_0	y_1	w_1	0.836	0.164
x_1	y_0	w_0	0.61	0.39
x_1	y_0	w_1	0.84	0.16
x_1	y_1	w_0	0.844	0.156
x_1	y_1	w_1	0.936	0.064

(b)

Figure 3.5: (a) NAT over all binary variables to be converted to a CSI model. (b) CPT obtained by normalizing NAT model in (a).

between member values), and the total number of clusters is significantly less than the number of NAT parameters, then it is worthwhile to express the NAT CPT as a CSI model. Tight clusters indicate smaller approximation errors and a fewer number of clusters indicates the CSI model saves space.

Based on this idea, the algorithm `Cluster` takes a binary-valued NAT CPT T and a distance bound δ. The algorithm groups values in the CPT into a set of clusters Ψ, such that the following conditions hold:

1. For each cluster $Q \in \Psi$ and each pair of values $p, q \in Q, |p - q| \leq \delta$. This condition specifies the inner cluster distance (distance between the minimum and maximum values) within each cluster is upper bounded by δ.

2. For each two clusters, $Q, R \in \Psi$, let $\min_Q, \max_Q, \min_R, \max_R$ be extreme values in Q and R, respectively. Either $\max_Q < \min_R$ or $\max_R < \min_Q$. This condition specifies that the clusters are ordered in ascending order, such that a cluster with smaller member values is positioned before a cluster with larger member values.

3. For clusters $Q, R \in \Psi$ where $\max_Q < \min_R$, we have $\min_R - \max_Q > \delta$. This condition specifies that the distance between any two clusters must be greater than δ.

The algorithm `Cluster` satisfies these conditions.

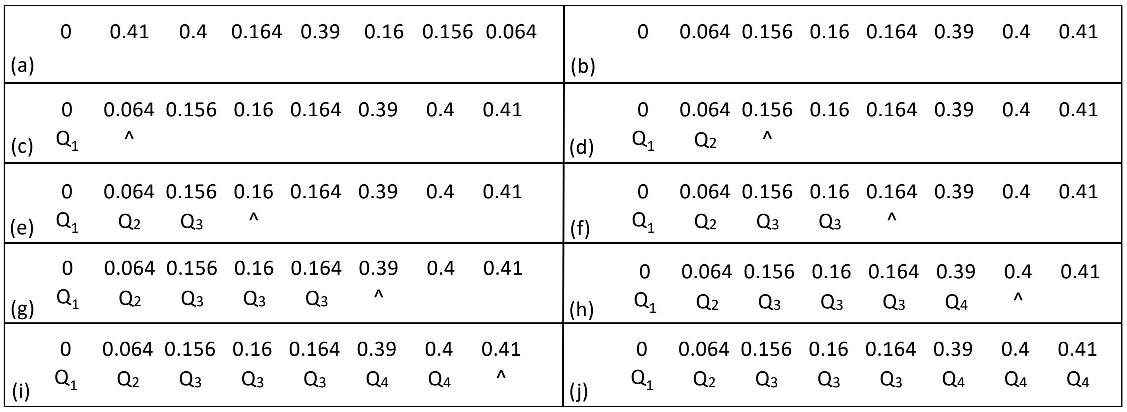
Algorithm 3.1: Cluster(T, δ)

```
1  sort values of T in ascending order;
2  $\Psi$ = an empty set of clusters [];
3  initialize Q to the cluster $\{t_0\}$;
4  for each prob $t_i$ of T[$t_1, ..., t_k$],
5      if ($t_i$ − first value of Q) $\geq \delta$,
6          add Q to $\Psi$;
7          Q = an empty cluster {};
8      }
9      add $t_i$ to Q;
10 }
11 if Q is not empty, add Q to $\Psi$;
12 return $\Psi$;
```

To demonstrate the clustering algorithm, consider the binary-valued CPT that specifies $P(z \mid x, y, z)$ shown in Figure 3.5 (b). The clustering algorithm's parameter T is specified as $P(z = z_1 \mid x, y, w)$ where z_1 is arbitrarily selected $\in dom(z)$. The values are shown in Figure 3.6 (panel a). The distance bound δ is specified to be 0.02.

The algorithm begins by sorting T in ascending order (panel b). We then initialize a set of empty clusters Ψ and create a new cluster Q containing the first element $t_0 = 0$ (panel c). The algorithm then iterates from the second probability to the last probability in the sorted order.

The 1st iteration $t_i = 0.064$, checks on line 5 if t_i is in the previous cluster, or if it is necessary to create a new cluster. This consists of computing t_i minus the first



(a)	0	0.41	0.4	0.164	0.39	0.16	0.156	0.064
(b)	0	0.064	0.156	0.16	0.164	0.39	0.4	0.41
(c)	0	0.064	0.156	0.16	0.164	0.39	0.4	0.41
	Q_1	^						
(d)	0	0.064	0.156	0.16	0.164	0.39	0.4	0.41
	Q_1	Q_2	^					
(e)	0	0.064	0.156	0.16	0.164	0.39	0.4	0.41
	Q_1	Q_2	Q_3	^				
(f)	0	0.064	0.156	0.16	0.164	0.39	0.4	0.41
	Q_1	Q_2	Q_3	Q_3	^			
(g)	0	0.064	0.156	0.16	0.164	0.39	0.4	0.41
	Q_1	Q_2	Q_3	Q_3	Q_3	^		
(h)	0	0.064	0.156	0.16	0.164	0.39	0.4	0.41
	Q_1	Q_2	Q_3	Q_3	Q_3	Q_4	^	
(i)	0	0.064	0.156	0.16	0.164	0.39	0.4	0.41
	Q_1	Q_2	Q_3	Q_3	Q_3	Q_4	Q_4	^
(j)	0	0.064	0.156	0.16	0.164	0.39	0.4	0.41
	Q_1	Q_2	Q_3	Q_3	Q_3	Q_4	Q_4	Q_4

Figure 3.6: Clustering of CPT for $P(z = z_1|x, y, w)$ where all variables are binary and the distance bound $\delta = 0.02$.

value in the cluster (0). Since the $0.064 - 0 = 0.064 > \delta$, the condition is met and it is necessary to create a new cluster. This process consists of adding the cluster Q to the set of clusters Ψ and resetting Q to an empty cluster (panel d). We then add t_i to newly reset cluster and proceed to the next iteration.

The 2nd iteration $t_i = 0.156$, repeats the check on line 5 to determine if t_i should be grouped in the previous cluster, or if it is necessary to create a new cluster. Since t_i minus the first value in the cluster (0.064) is greater than δ, it is necessary to create a new cluster. We add the cluster Q to the set of clusters Ψ and reset Q to an empty cluster (panel e). We then add t_i to the newly reset cluster and proceed to the next iteration.

The 3rd iteration $t_i = 0.16$ tests if it should be grouped in the previous cluster, or if it is necessary to create a new cluster. Since t_i minus the first value in the cluster is $\leq \delta$, we need not create a new cluster. The t_i value is subsequently added to the cluster Q (panel f). The iterations continue for each of the remaining values in the CPT (panel g through j). Once all values have been clustered, it is possible that the last cluster is non-empty and has not been added to Ψ. Hence, once all iterations are complete, we add Q to Ψ on line 11. Figure 3.7 shows the resulting clustered CPT.

Clust.	x	y	w	$P(z\|x,y,w)$ z_0	z_1
Q_1	x_0	y_0	w_0	1	0
Q_2	x_1	y_1	w_1	0.936	0.064
Q_3	x_0	y_1	w_1	0.844	0.156
Q_3	x_1	y_0	w_1	0.84	0.16
Q_3	x_1	y_1	w_0	0.836	0.164
Q_4	x_0	y_0	w_1	0.61	0.39
Q_4	x_0	y_1	w_0	0.6	0.4
Q_4	x_1	y_0	w_0	0.59	0.41

Figure 3.7: CPT of $P(z|x, y, w)$ with initial clusters specified.

While this clustering algorithm groups the values according to a distance bound, it does not guarantee that each value in the cluster corresponds to instantiations that are exploitable by CSI.

Two instantiations in a CPT α_1, α_2 are *exploitable* by CSI iff their intersection (the context) is non-empty $\alpha_1 \cap \alpha_2 \neq \emptyset$. An exploitable cluster is expressible by a single CPT-tree leaf. Conversely, if the intersection of the instantiations is empty,then the cluster is not expressible by a single CPT-tree leaf since the conditions for context-specific independence are not met as it requires a non-empty context.

Consider, the clusters Q_3 in the CPT shown in Figure 3.7. It groups 3 CPDs corresponding to the instantiations: $(x = x_0,\ y = y_1,\ w = w_1)$, $(x = x_1,\ y = y_0,\ w = w_1)$, and $(x = x_1,\ y = y_1,\ w = w_0)$. It can be observed that the intersection of these instantiations is an empty set.

$$x_0 \cap x_1 \cap x_1 \quad \cup \quad y_1 \cap y_0 \cap y_1 \quad \cup \quad w_1 \cap w_1 \cap w_0 \quad = \quad \emptyset$$

Thus, the cluster would not be exploitable by CSI. The number of clusters obtained by the clustering algorithm is consequently a lower-bound on the number of parameters when the NAT CPT is approximated by a CSI model. We resolve this issue by splitting such clusters into the largest compatible sub-clusters possible.

3.3.3 Splitting Clusters to Ensure Exploitability

The algorithm `Split` takes a set of clusters Ψ that may or may not be exploitable, and returns a set of exploitable clusters Ψ'. The set of input clusters Ψ is typically obtained from the `Cluster` algorithm.

Algorithm 3.2: Split(Ψ)

```
1   $\lambda$ = a queue initialized to $\Psi$;
2   $\Psi'$ = empty set of clusters [];
3
4   while $\lambda$ is non−empty,
5       $Q_i$ = pop front of queue;
6       if $Q_i$ is of size 1, add $Q_i$ to $\Psi'$ and continue;
7       if intersection of all instantiations in $Q_i \neq \emptyset$,
8           add $Q_i$ to $\Psi'$ and continue;
9       $v_i$ = most frequent variable value held by instantiations in $Q_i$;
10      $R$ = all instantiations in $Q_i$ where $v_i$ is assigned;
11      $S$ = all instantiations in $Q_i$ where $v_i$ is not assigned;
12      add $R$ to $\Psi'$ and append $S$ to $\lambda$;
13  }
14  return $\Psi'$;
```

The `Split` algorithm works as follows. On line 1, we initialize a queue λ holding all clusters in Ψ. The order of clusters inserted into the queue is immaterial. On line 2, we initialize an empty set of clusters Ψ' to hold all exploitable clusters.

On line 4, we enter the main loop that continues looping until the queue is empty. On line 5, we pop the front of the queue to obtain the current cluster Q_i that will be tested. On lines 6 and 7, we check if a splitting is unnecessary. This consists of two tests: (i) if the cluster Q_i contains 1 instantiation, then it is said to be exploitable. (ii) If all instantiations in the cluster Q_i are exploitable such that their intersection is non-empty. If either of these tests are positive, then we add the cluster to the set of exploitable clusters Ψ' and continue to the next iteration. If both tests fail, we proceed to the splitting on lines 9 through 12.

The splitting first consists of identifying the most frequent value of all variables across the instantiations in the cluster. Let the most frequent value across all variables be the value v_i assigned to variable v. We then partition the instantiations in the cluster Q_i into 2 segments: R and S. The segment R contains all instantiations where $v = v_i$ and S contains all instantiations where $v \neq v_i$. We then add R to the set of exploitable clusters Ψ' and append S to the queue λ. The algorithm continues on to the next cluster in the queue.

We demonstrate the algorithm on the clustered CPT shown in Figure 3.7. We begin by initializing λ to the set of clusters:

$$\begin{aligned}
\{ \\
&\{(x = x_0, y = y_0, w = w_0)\}, \\
&\{(x = x_1, y = y_1, w = w_1)\}, \\
&\{(x = x_0, y = y_1, w = w_1), (x = x_1, y = y_0, w = w_1), (x = x_1, y = y_1, w = w_0)\}, \\
&\{(x = x_0, y = y_0, w = w_1), (x = x_0, y = y_1, w = w_0), (x = x_1, y = y_0, w = w_0)\} \\
\}
\end{aligned}$$

We initialize the exploitable clusters set Ψ' to an empty set. For the 1st iteration, we pop the following cluster:

$$\{(x = x_0, y = y_0, w = w_0)\}$$

We then test if the cluster contains 1 instantiation. Since this test passes, we add Q_i to the set of exploitable clusters Ψ' and proceed to the next iteration.

The 2nd iteration pops the following cluster from the queue:

$$\{(x = x_1, y = y_1, w = w_1)\}$$

Since it also contains 1 instantiation, it is added to the set of exploitable clusters Ψ'.

The 3rd iteration pops the following cluster from the queue:

$$\{(x = x_0, y = y_1, w = w_1), (x = x_1, y = y_0, w = w_1), (x = x_1, y = y_1, w = w_0)\}$$

We then test if the cluster contains 1 instantiation. This test fails since the cluster contains 3 instantiations. We then test if the entire cluster is exploitable.

$$x_0 \cap x_1 \cap x_1 \quad \cup \quad y_1 \cap y_0 \cap y_1 \quad \cup \quad w_1 \cap w_1 \cap w_0 \quad = \quad \emptyset$$

The intersection of all assignments in Q_i is an empty set. Hence, the conditional on line 7 fails as the cluster is not exploitable and must be split. We then identify the most frequent value held by instantiations. Table 3.1 shows the result of recording the frequency of each value in Q_i.

v_i	Count of v_i in Q_i
x_1	2
y_1	2
w_1	2
x_0	1
y_0	1
w_0	1

Table 3.1: Frequency of values in 3rd iteration's cluster.

In this case, there is a 3-way tie for the most frequent value of 2. We settle the tie by arbitrarily selecting one value — x_1. We note that it is critical to select one maximum frequency value, not multiple. Selecting multiple values will not maximize the size of splits and will result in sub-optimal clusters. For example, selecting x_1 by itself has 2 matching instantiations but selecting $x_1 \wedge w_1$ has only 1 matching instantiation.

Next, we partition the cluster into two segments R and S. The segment R contains all instantiations where $x = x_1$:

$$\{(x = x_1, y = y_0, w = w_1), (x = x_1, y = y_1, w = w_0)\}$$

The segment S contains all instantiations where $x \neq x_1$:

$$\{(x = x_0, y = y_1, w = w_1)\}$$

The segment R is now exploitable and can be added to the set of exploitable clusters Ψ', while the segment S may not be exploitable and is appended to the queue λ in order for further testing and splitting if needed.

The 4th iteration pops the following cluster from the queue:

$\{(x = x_0, y = y_0, w = w_1), (x = x_0, y = y_1, w = w_0), (x = x_1, y = y_0, w = w_0)\}$

This cluster fails the tests on lines 5 and 6 as the cluster is not a single instantiation nor is the cluster's intersection an empty set:

$$x_0 \cap x_0 \cap x_1 \quad \cup \quad y_0 \cap y_1 \cap y_0 \quad \cup \quad w_1 \cap w_0 \cap w_0 \quad = \quad \emptyset$$

We then proceed to the identification of the most frequent value as shown in Table 3.2.

v_i	Count of v_i in Q_i
x_0	2
y_0	2
w_0	2
x_1	1
y_1	1
w_1	1

Table 3.2: Frequency of values in 4th iteration's cluster.

In this case, there is a 3-way tie for the most frequent value of 2. We settle the tie by arbitrarily selecting one value — x_0. Next, we partition the cluster into two segments R and S. The segment R contains all instantiations where $x = x_0$:

$$\{(x = x_0, y = y_0, w = w_1), (x = x_0, y = y_1, w = w_0)\}$$

And, the segment S contains all instantiations where $x \neq x_0$:

$$\{(x = x_1, y = y_0, w = w_0)\}$$

The segment R is now exploitable and can be added to the set of exploitable clusters Ψ', while the set S may not be exploitable and is appended to the queue λ in order for further testing and splitting if needed.

The 5th iteration pops the following cluster from the queue:

$$\{(x = x_0, y = y_1, w = w_1)\}$$

This cluster is of 1 instantiation and thus is added directly to set of exploitable clusters. The 6th iteration also contains one instantiation and is added to the set of exploitable clusters Ψ'. The 5th and 6th iterations popped clusters that were appended to the queue by the R/S partitioning performed in the 3rd and 4th iterations, respectively.

The CPT shown in Figure 3.8 shows the final clustered CPT such that each cluster is exploitable. The CPT contains 6 instantiations, which is an increase of 2 instantiations (printed in red) relative to the initial clustering shown in Figure 3.7.

3.3.4 Extending Clustering to Multi-valued Variables

The clustering algorithm outlined by the `Cluster` algorithm is restricted to binary variables. In this section, we discuss how the clustering approach may be extended to multi-valued variables. This is achieved by building the test up from binary variables to ternary, and finally to the general variables. While we did not perform

				$P(z\|x,y,w)$	
Clust.	x	y	w	z_0	z_1
Q_1	x_0	y_0	w_0	1	0
Q_2	x_1	y_1	w_1	0.936	0.064
Q_3	x_1	y_0	w_1	0.844	0.156
Q_3	x_1	y_1	w_0	0.84	0.16
Q_4	x_0	y_0	w_1	0.61	0.39
Q_4	x_0	y_1	w_0	0.6	0.4
Q_5	x_0	y_1	w_1	0.836	0.164
Q_6	x_1	y_0	w_0	0.59	0.41

Figure 3.8: CPT of $P(z|x,y,w)$ and its clusters after splitting all unexploitable clusters. The newly added clusters that were introduced by the splitting in order to make all clusters exploitable are printed in red.

experiments with this extension due to time constraints, the general case was implemented in Java to verify correctnesss.

Binary Case

Consider the CPT $P(x|y,z)$ with variables x, y, z over the domains $dom(x) = \{x_0, x_1\}$, $dom(y) = \{y_0, y_1\}$, $dom(z) = \{z_0, z_1\}$, respectively. Suppose, we would like to know if the CPT encodes the CSI interaction $I_C(x; z|y = y_0)$. This can be determined by performing the following test:

$$P(x_0|y_0, z_0) = P(x_0|y_0, z_1)$$

In plain language, we must test that $P(x_0|y_0, z)$ is the same for both possible values of z: $z = z_0$ and $z = z_1$.

Figure 3.9 presents an example CPT specifying $P(x|y,z)$ where x, y, z are all binary variables. The CPT exhibits the CSI interaction as $P(x = x_0|y = y_0, z = z_i) = 0.3$ where $i = 0, 1$. It is also observed that testing for $P(x = x_1|y = y_0, z = z_i) = 0.7$ is redundant, given that we have established x_0 and that the CPDs must sum to 1.

y	z	$x = x_0$	$x = x_1$
y_0	z_0	0.3	0.7
y_0	z_1	0.3	0.7
y_1	z_0	0.1	0.9
y_1	z_1	0.2	0.8

Figure 3.9: Example CPT with three binary variables

Ternary Case

Now, suppose we extend the domains of all variables to ternary: $dom(x) = \{x_0, x_1, x_2\}, dom(y) = \{y_0, y_1, y_2\}$ and $dom(z) = \{z_0, z_1, z_2\}$. To determine if the same CSI interaction $I_C(x; z|y = y_0)$ is exhibited by the CPT, we now must perform the following tests:

$$P(x = x_0|y = y_0, z = z_0) = P(x = x_0|y = y_0, z = z_1) = P(x = x_0|y = y_0, z = z_2)$$

$$and$$

$$P(x = x_1|y = y_0, z = z_0) = P(x = x_1|y = y_0, z = z_1) = P(x = x_1|y = y_0, z = z_2)$$

Hence, the number of tests required to establish CSI has increased, making the conditions for CSI more stringent. In other words, testing for CSI with binary variables requires only 1 condition to be met, but testing for CSI with ternary variables requires 6 conditions to be met. Note, we do not have to test $P(x = x_2|y = y_0, z = z_i)$ since all CPDs sum to 1. To demonstrate the ternary case, consider the example CPT shown in Figure 3.10.

In the above CPT, we can see that CSI interaction holds as $P(x = x_0|y = y_0, z = z_i) = 0.3$ and $P(x = x_1|y = y_0, z = z_i) = 0.5$ where $i = 0, 1$. It is also observed that testing for $P(x = x_2|y = y_0, z = z_i) = 0.2$ is redundant, given that we have already established x_0 and x_1 and know that the CPDs must sum to 1.

y	z	$x = x_0$	$x = x_1$	$x = x_2$
y_0	z_0	0.3	0.5	0.2
y_0	z_1	0.3	0.5	0.2
y_0	z_2	0.3	0.5	0.2
y_1	z_0	0.1	0.2	0.7
y_1	z_1	0.2	0.6	0.2
y_1	z_2	0.3	0.4	0.3
y_2	z_0	0.4	0.5	0.1
y_2	z_1	0.6	0.1	0.3
y_2	z_2	0.7	0.1	0.2

Figure 3.10: Example CPT with three ternary variables

General Case

To generalize the test for context-specific independence, let the domain of x be of size m, and let the domain of z be of size n. We then will have $m \times n$ tests of the form:

$$P(x = x_0|y = y_0, z = z_0) = \cdots = P(x = x_0|y = y_0, z = z_{n-1})$$
$$P(x = x_1|y = y_0, z = z_0) = \cdots = P(x = x_1|y = y_0, z = z_{n-1})$$
$$\cdots$$
$$P(x = x_{m-1}|y = y_0, z = z_0) = \cdots = P(x = x_{m-1}|y = y_0, z = z_{n-1})$$

In order for CSI to hold, the above expression must hold for all values of x, z from 0 to $m - 1$ and 0 to $n - 1$, respectively.

3.4 Evaluating Expression of CSI Models as NAT Models

Theoretically, if the NAT model were able to encode a CSI CPT, then it should be able to exactly model every CSI CPT and require the same or fewer parameters than a CSI representation. Hence, in this section, we conduct an experiment to find counterexamples to demonstrate by contradiction that there exists CSI models, which cannot be encoded by the NAT model exactly.

3.4.1 Pre-processing: Adding CSI to an Existing CPT

In this section, we outline the method used to randomly generate CPT-trees that exhibit the same CSI interaction but encode different CPTs over the same BN family topology. This method is needed because we wish to identify repeatable counterexamples of CSI models that cannot be encoded by a NAT model. Hence, it is necessary to demonstrate that some CSI interaction(s) are not expressible by a NAT model over a sufficiently wide range of CPTs.

Unfortunately, naively randomly generating CPT-trees has no guarantee to encode the same CSI interaction. This can be mitigated by adding CSI interactions to a randomly generated CPT, denoted P^*, over the same variables. Initially, P^* would not encode any CSI interactions. Our method modifies the CPDs in P^* affected by the CSI interaction, while leaving the non-affected CPDs unchanged. This results in a randomly generated CPT exhibiting the same CSI interaction, but with different values.

Consider, the randomly generated binary-valued CPT $P(z|x, y, w)$ in the left pane of Figure 3.11 and the CSI interaction $I_C(z; w|y, x = x_1)$ from Section 3.2. To generate a new CPT P^* from the CPT P and the CSI interaction I, we must assign the same probability for each distinct combination of (y, z) when $x = x_0$.

We begin by duplicating P as P^*. For each distinct (y, z) combination denoted (y', z'), we retrieve a probability ρ from P such that $P(z = z'|x = x_1, y = y', w)$ where w is arbitrarily assigned. Then, for each instantiation in P^* that satisfies the combination $(y = y', z = z')$, we replace the probability corresponding to that instantiation with ρ. This ensures that all instantiations of a (y', z') combination have the same value.

x	y	w	$P(z\|x,y,w)$ z_0	z_1
x_0	y_0	w_0	0.3	0.7
x_0	y_0	w_1	0.5	0.5
x_0	y_1	w_0	0.8	0.2
x_0	y_1	w_1	0.1	0.9
x_1	y_0	w_0	0.6	0.4
x_1	y_0	w_1	0.4	0.6
x_1	y_1	w_0	0.7	0.3
x_1	y_1	w_1	0.2	0.8

x	y	w	$P(z\|x,y,w)$ z_0	z_1
x_0	y_0	w_0	0.3	0.7
x_0	y_0	w_1	0.5	0.5
x_0	y_1	w_0	0.8	0.2
x_0	y_1	w_1	0.1	0.9
x_1	y_0	w_0	0.6	0.4
x_1	y_0	w_1	0.6	0.4
x_1	y_1	w_0	0.7	0.3
x_1	y_1	w_1	0.7	0.3

Figure 3.11: Left: Initial randomly generated CPT for $P(z|x,y,w)$ where all variables are binary. Right: CPT $P^*(z|x,y,w)$ after incorporating $I_C(z;w|y,x=x_1)$. Each distinct combination of (y,z) when $x=x_1$ is printed in a different colour.

The right pane of Figure 3.11 shows the resultant CPT. Each combination is printed in a different colour. For the $(y=y_0, z=z_0)$ combination that is printed in red, we assigned a value of 0.6. For the $(y=y_0, z=z_1)$ combination that is printed in green, we assigned a value of 0.4. For the $(y=y_1, z=z_0)$ combination that is printed in purple, we assigned a value of 0.7. Lastly, for the $(y=y_1, z=z_1)$ combination that is printed in orange, we assigned a value of 0.3.

3.4.2 Experimental Setup

The objective of this experiment is to identify CSI models that cannot be encoded by NAT models. We simulated a batch of 100 randomly CPTs. Each CPT had a child variable v dependent on 5 parent variables q, r, s, t, u. All variables have a domain size of 5.

We specified 3 CSI interactions to evaluate. Each CSI interaction imparts a varying amount of duplication in a CPT. The CSI interactions are as follows:

1. $I_c(v; t, u \mid q = q_1, r = r_2, s \in \{s_3, s_4\})$

   v is contextually independent of t and u when $q = q_1, r = r_2, s \in \{s_3, s_4\}$.

2. $I_c(v; r, s, t, u \mid q = q_1)$
   v is contextually independent of r, s, t, u when $q = q_1$.

3. $I_c(v; r, s, t, u \mid q \in \{q_1, q_2, q_3, q_4\})$
   v is contextually independent of r, s, t, u when $q \in \{q_1, q_2, q_3, q_4\}$.

For each CPT P in the batch, we created three copies P_1, P_2, P_3. Using the preprocessing outlined in Section 3.4.1, CSI interaction 1 is added to P_1, CSI interaction 2 is added to P_2, and CSI interaction 3 is added to P_3. Thus, we create $4 \times 100 = 400$ total source CPTs. We then compress each CPT into a NAT model. The accuracy of the compressions were evaluated and compared per CSI interaction. Results are presented in the next section.

3.4.3 Experimental Results

The results of this experiment are presented in Table 3.3. The CSI interactions (column 1) are ordered from the least amount of space reduction to the greatest amount of space reduction. The number of parameters for the CSI model and NAT model are presented in columns 2 and 3 respectively. The average Kullback-Leibler and Euclidean distances between the NAT and source CPTs are shown in columns 4 and 5.

CSI Statement	#Src Param	#NAT Param	KL	ED
No CSI	12,500	80	0.738	0.219
$I_c(v; t, u \mid q = q_1, r = r_2, s \in \{s_3, s_4\})$	12,304	80	0.710	0.214
$I_c(v; r, s, t, u \mid q = q_1)$	10,004	80	0.692	0.210
$I_c(v; r, s, t, u \mid q \in \{q_1, q_2, q_3, q_4\})$	2,504	80	0.501	0.176

Table 3.3: Summary of experiments when representing CSI CPTs as NAT models.

It is observed that CSI CPTs take more than 30 times the space of the resultant NAT models. These representational savings are significant, but come at the price of an approximation error. While the approximation error decreases when the number of CSI parameters decreases, these results suggest that NAT models generally cannot encode CSI CPTs exactly.

3.5 Evaluating Expression of NAT CPTs as CSI Models

Conversely, we now empirically demonstrate that NAT models cannot be efficiently and exactly encoded as CSI models. The approach of this section will be similar to the above as we will conduct an experiment to identify NAT CPT counterexamples that cannot be encoded by a CSI CPT.

3.5.1 Experimental Setup

The experiment is conducted on 100 generated NAT CPTs, each over a CPT of 5 parent variables. All variables are binary, with 32 parameters per CPT. Each NAT CPT is clustered with a distance bound of $\delta = 0.02$ and split as needed.

3.5.2 Experimental Results

The results are plotted in Figure 3.12. Each bar counts the number of NAT CPTs that produced a particular number of clusters. It can be observed that all CPT-trees need at least 17 parameters, while the NAT CPT only requires 5 parameters.

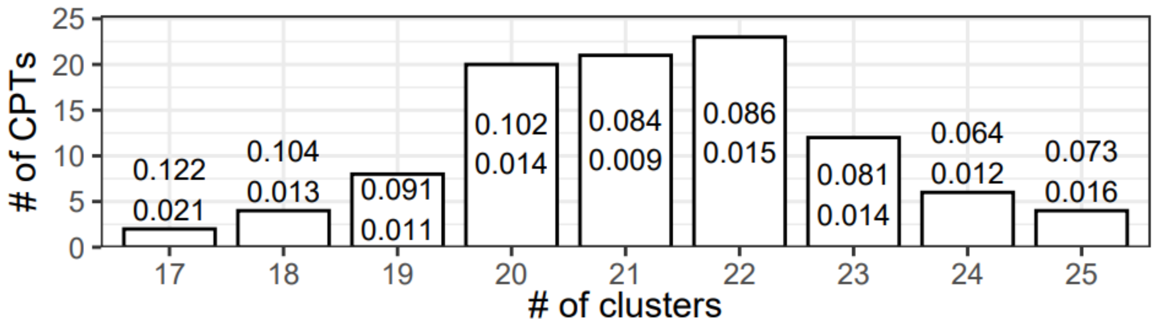


Figure 3.12: Experiment results on representing NAT CPTs as CSI models.

Since we weakened the requirements for a CPT to exhibit CSI, there is now a modelling error associated with the CSI representation. Each cluster can be identified by a *centroid*, which is computed as the mean of the cluster's member values. To evaluate the error, we expand the clusters into a tabular CPT where each instantiation is specified by the centroid containing the instantiation. We then calculate the Euclidean distance between the source CPT and the CPT generated by expanding the clustering results. In Figure 3.12, each bar is labelled by its average modelling error on the top and the standard deviation below it.

Overall, the CSI representation is not able to encode the NAT CPT exactly and efficiently. If exactness is required, we demonstrated by definition that a CSI representation is unable to encode a NAT CPT with no duplicated probabilities efficiently. If exactness is not required, we applied clustering to determine the number of parameters required to express the NAT CPT as a CPT-tree. The results of clustering demonstrated that the CSI representation not only introduces an error but also requires a greater number of parameters to encode the CPT. These findings in combination with Section 3.4 suggest that the NAT and CSI representations are orthogonal.

Chapter 4

Formalizing CPT-tree Transformation

In this chapter, a CPT-tree transformation algorithm is designed. The idea for CSI transformation was first introduced in Boutillier et al. [3] through a simple binary example. However, to the best of our knowledge, no general algorithm has been formalized. We aim to fill this gap by generalizing network transformation to multi-valued variables and formalizing the process through a suite of algorithms.

The chapter is laid out as follows: Section 4.1 extends CPT-tree arcs to be set-valued. Section 4.2 describes the general algorithm by pseudocode and a running example. Section 4.3 demonstrates the exactness of a source family's CPT compared to a transformed family's CPT. Lastly, Section 4.4 discusses a property of CPT-trees that affects the efficiency of inference.

4.1 Set-valued CPT-tree Edges

In this section, we extend the CPT-tree representation to support set-valued edges. Recall that, in a CPT-tree as specified by [3], each outgoing edge in a CPT-tree is labelled by a path that the variable holds, and that a path from the root to each leaf labels a single context in the CPT. For example, in Figure 4.1, the edge from u to the leftmost v node is labelled by the value u_0. The leftmost leaf node with the label $z(0.8)$ is reached by the context $(u = u_0, v = v_0, w = w_0)$. While this idea is suitable for binary trees, it does not extend to multi-valued trees.

Figure 4.1 specifies $P(z|u, v, w)$ where u is ternary and all other variables are binary. In this example, it can be observed that the left sub-tree rooted at v and

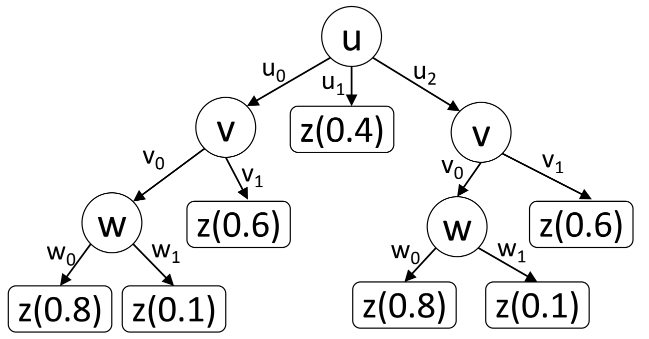


Figure 4.1: CPT-tree with single-value arcs specifying $P(z|u, v, w)$ where u is ternary and all other variables are binary.

the right sub-tree rooted at v are identical. Yet, the CPT-tree does not exploit this interaction due to fact that each arc can only be labelled by a single value. The importance of this limitation grows as the number of variables grow resulting in the possibility of a greater number of duplicate sub-trees.

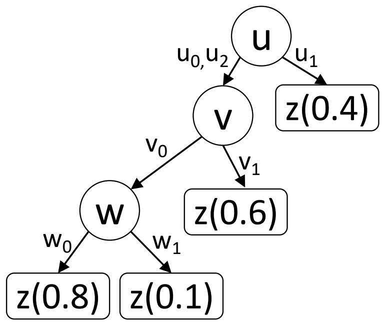


Figure 4.2: CPT-tree with set-value arcs specifying $P(z|u, v, w)$ where u is ternary and all other variables are binary.

In this work, we generalize arcs of CPT-trees to support set-valued edges. A set-valued notation allows for each outgoing from a node n to be labelled by a subset of the values in $dom(n)$. For example, the left-most edge connecting u to v in Figure 4.2 is labeled by a set of values $\{u_0, u_2\}$ in $dom(u)$. This allows for the 2 identical sub-trees to be consolidated into a single sub-tree.

A byproduct of introducing the set-valued notation is that a path from the root to a leaf no longer encodes a single context. Instead, the the path from the root to a leaf encodes a set of contexts. In Figure 4.2, all paths from the root node to a leaf node with the set-valued edge specify 2 contexts. For example, the leftmost leaf node labelled as $z(0.8)$ is reached by 2 contexts: $(u = u_0, v = v_0, w = w_0)$ and $(u = u_2, v = v_0, w = w_0)$.

It is noted that the single-value notation is a special case of the set-valued notation. The set-value notation expresses single values by a singleton. For instance, the single-value notation would label the rightmost edge connecting u to v as u_1. By contrast, the set-valued notation would express the same edge label as $\{u_1\}$. We omit all braces in figures with no ambiguity for readability.

4.2 Algorithm Suite

Network transformation is a method to transfer the structure of the CPT-tree to BN that preserves the context-specifies independencies encoded in the CPT-tree. This is achieved by swapping the child variable x of a BN family with a new structure, composed entirely of auxiliary variables, with the exception of x and its parents $\pi(x)$.

In this section, we introduce the suite of algorithms for CSI transformation. The suite of algorithms consists of three algorithms: generation of the BN segment to encode the CSI interaction, assignment of CPTs, and generation of switch CPTs. Each algorithm is discussed individually in Sections 4.2.1, 4.2.2, and 4.2.3, respectively.

4.2.1 Generate BN Segment to Encode CSI Interaction

The first algorithm `SetDagSeg` accepts a CPT-tree T over a variable x and parents $\pi(x)$ and generates a BN segment with a single leaf x that encodes the CSI interaction.

The transformation is applied from top-to-bottom of the CPT-tree. Let each node of the CPT-tree be assigned a level, corresponding to the depth of the node from the root. For example, the root is level 0, the children of the root are level 1, the grandchildren of the root are level 2 and so forth. Thus, we apply the transformation is driven in the CPT-tree from level 0 onwards.

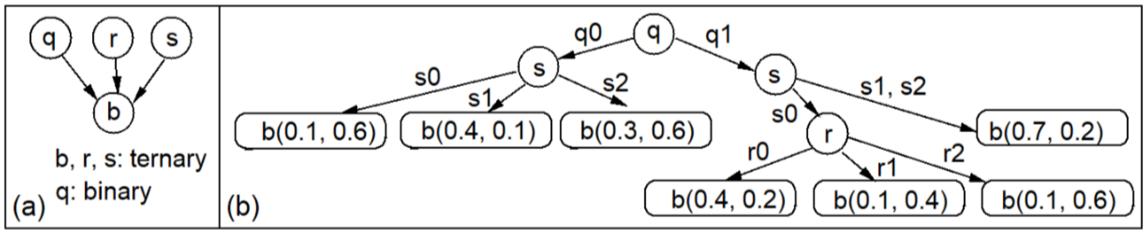


Figure 4.3: (a) A BN family. (b) CPT-tree for the family.

Let each node n in the CPT-tree have a path. The path, $path(n)$ indicates the branches taken from the root node to the node n. Each branch is indicated by a variable-value pair: the label of the node indicates the variable and the label of the arc indicates the value$s$ assigned to the variable. For instance, in Figure 4.3, the left-most node in the CPT-tree which is labelled $b(0.1, 0.6)$ can be reached by the path $q = q_0, s = s_0$.

We demonstrate `SetDagSeg` through an example using the BN and CPT-tree pictured in Figure 4.3. Variables b,r,s are ternary while the variable q is binary. The algorithm begins with an empty graph G consisting of the nodes x and $\pi(x)$. There are no connections between any of the nodes. This is shown in Figure 4.4. The current path is initialized to $\{\}$ and the child node b is labelled as $b_{\{\}}$.

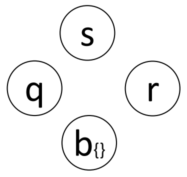


Figure 4.4: Initialization of Algorithm 4.1

Algorithm 4.1: SetDagSeg(T)

```
1  initialize empty graph G with nodes $\{x\} \cup \pi(x)\}$;
2  denote r the root of T by p;
3  set path(p) = {};
4  label x in G as $x_{path(p)}$;
5
6  for level $L = 0$ to max level in T,
7    for each node t in T at level L with path(t),
8      find node v in G that is labelled $x_{path(t)}$;
9      add arc $t \rightarrow v$ in G
10     if each child of t in T is a leaf, continue;
11     refer to t as a multiplexer;
12     denote partition of dom(t) by arcs outgoing from t as $\{sd_1, ..., sd_m\}$;
13     for i=1 to u,
14       add new node y to G with domain dom(x) and label it $x_{\{path(t), t \in sd_i\}}$;
15       add arc $y \rightarrow v$ in G;
16 return G;
```

For the first level $L = 0$ in the CPT-tree T, the only node at that level is the root node q (t in the algorithm). The algorithm identifies a node labelled $b_{\{\}}$ (v in the algorithm). An arc is then added from node q to the node $b_{\{\}}$. We then check if each child of q is a leaf. Since this test is negative, we continue processing the iteration. We now can identify q as a *multiplexer node*, since it has children and not all children are leaves. We then partition the domain of q into two segments, since the node q in the CPT-tree has two children. The first segment is $\{q_0\}$ while the second segment is $\{q_1\}$. For each segments, we create a new auxiliary node (y in the algorithm) labelled $b_{q=q_0}$ and $b_{q=q_1}$, respectively. Both auxiliary nodes are added to the segment as parents of $b_{\{\}}$ (or equivalently, siblings of q). The first iteration, shown in Figure 4.5, is now complete and we can move onto discussing the second level.

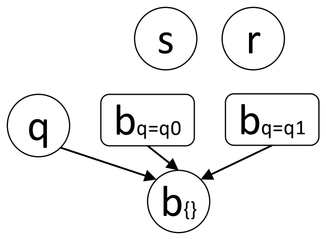


Figure 4.5: BN segment after transforming the root node of the CPT-tree.

The second level $L = 1$ in the CPT-tree T consists of two nodes. We will discuss each node individually.

The left-most node in the second level $L = 1$ is a special node because *all of its children are leaves* (ACAL). Decomposing the node will not yield any savings as there are no further CSI interactions. The iteration for the left-most node s begins by identifying node $b_{q=q_0}$ as node v and adds an arc from s to $b_{q=q_0}$. We then check if each child of s in the CPT-tree is a leaf. Since this test is positive, the algorithm will skip the decomposition by continuing to the next node. This is shown in Figure 4.6.

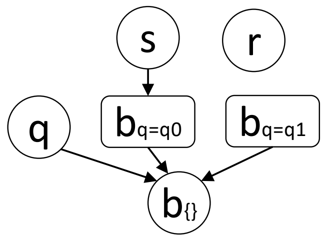


Figure 4.6: BN segment after skipping the decomposition of the left-most node in the second level of the CPT-tree, $b_{q=q0}, s$ as all of $b_{q=q0}$'s children are leaves.

The right-most node in the second level $L = 1$ of the CPT-tree is also node s. This iteration begins by identifying its parent in the CPT-tree (v) as $b_{q=q_1}$. It then adds an arc from s to $b_{q=q_1}$. Since node s in the CPT-tree has one non-leaf child, it fails the test on line 10 and we continue processing the iteration. We then partition the ternary domain of s into two segments $\{s_0\}$ and $\{s_1, s_2\}$, by the values assigned to the arcs of the outgoing edges from s. We then create two new nodes in the BN segment with the labels $b_{q=q_1,s=s_0}$ and $b_{q=q_1,s\in\{s_1,s_2\}}$, respectively. Each newly introduced node is added as a parent of of $b_{q=q_1}$. This is shown in Figure 4.7.

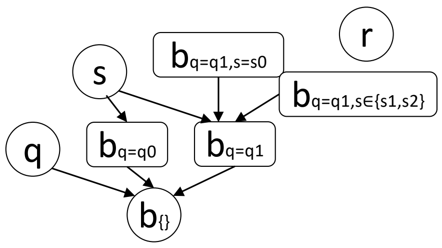
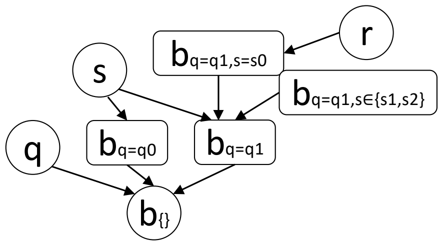


Figure 4.7: BN segment after the decomposition of the right-most node in the second level, $b_{q=q_1}$ as all of $b_{q=q_0}$'s children are leaves.

Lastly, the third level $L = 2$ of the CPT-tree consists of r. Similar to the left-most node in the CPT-tree, this node consists of all leaf children rendering a decomposition redundant. The algorithm adds an arc from r to $b_{q=q1,s=s_0}$ This concludes the generation of the BN segment.



In summary, we can classify the auxiliary nodes introduced by the `SetDagSeg` algorithm into the following three types:

- *ACAL Nodes*: These nodes are created when the CPT-tree node has all leaf children. They are added in line 14 and never processed after by the test on line 7. Hence, they remain roots (e.g., $b_{q=q_1;s_2\in\{s_1;s_2\}}$).
- *Multiplexer Nodes*: These nodes are created when the CPT-tree has at least

one non-leaf child. They are processed on line 7 as v and by the *for* loop on line 13 (e.g., $b_{\{\}}$ and $b_{q=q_1}$).

- *Outer Nodes*: These nodes are created when the CPT-tree node is a leaf. They are the remaining nodes that are processed in line 7 as v, passed the test in line 7, and skipped the for loop in line 13 (e.g., $b_{q=q_0}$ and $b_{q=q_1;s=s_0}$).

Each type of node has a different method to assign the CPT. We discuss each of these in detail next.

4.2.2 Assignment of CPTs to Generated BN Segment

Given a child variable x, its parents $\pi(x)$, a CPT-tree T and a transformed BN segment, the second algorithm `AssignCPT` iterates through each non-parent node in G and assigns the appropriate CPT based on the type of node.

Algorithm 4.2: AssignCpt(x, $\pi(x)$, T, G)

```
1   for each node v in G,
2     if v ∈ π(x), continue;
3     if v is Outer Node with path(v),
4       traverse path(v) in T to leaf t;
5       retrieve CPD parameters at t and assign the CPD to v;
6     else if v is ACAL with path(v),
7       traverse path(v) in T to node t;
8       for each child z of t in T, retrieve CPD parameters at z;
9       assemble the CPDs into CPT and assign to v;
10    else, // v is Multiplexer
11      denote the unique parent of v from π(x) by y;
12      call SetSwitchCpt(v, y, T, G) and assign the CPT returned to v;
13    return G;
```

We demonstrate the `AssignCpt` algorithm by applying it to the BN segment pictured in Figure 4.7

For Outer Nodes, the CPT is retrieved directly from the CPT-tree. For instance, in Figure 4.7, the node $b_{q=q1,s\in s1,s2}$'s CPT is assigned by following the path $(b_{q=q_1}, s \in \{s_1, s_2\})$ in the CPT-tree to the leaf node. The label of the leaf node represents a CPD encoded by its parameters. Figure 4.8 presents the CPD for the node $b_{q=q1,s\in s1,s2}$.

$b = b_0$	$b = b_1$	$b = b_2$
0.7	0.2	0.1

Figure 4.8: CPT for Outer Node $b_{q=q1,s\in s1,s2}$

For ACAL nodes, the CPT is assembled from the CPT-tree node's children. Consider, the node $b_{q=q0}$ in Figure 4.7. Follow the path $q = q0$ in the CPT-tree Figure 4.7 to a node t. Each child of t is a leaf that specifies a CPD for when $s = s_i$ where $i \in \{0, 1, 2\}$. Assembling all CPDs together results in the CPT presented in Figure 4.9.

s	$b = b_0$	$b = b_1$	$b = b_2$
s_0	0.1	0.6	0.3
s_1	0.4	0.1	0.5
s_2	0.3	0.6	0.1

Figure 4.9: CPT for ACAL Node $b_{q=q0}$

For Multiplexer nodes, the first parent y is from $\pi(x)$ and is identified on line 11 of `AssignCpt`. The other parents are all auxiliary. The CPT for node v is deterministic. In the next section, we discuss how the CPT is generated.

4.2.3 Generate Multiplexer CPT

The last algorithm generates the multiplexer CPT. Given a a multiplexer, child variable v, a parent y that v is switching on, a CPT-tree T and a transformed BN segment G, the SetSwitchCpt algorithm assigns a deterministic CPT that switches based on the value of v, according to a specified deterministic distribution.

We first demonstrate the deterministic distribution by way of an example. Consider the CPT for the multiplexer node $v = b_{\{\}}$ with its parent $y = q$. The CPT for $P(b_{\{\}} = b_i \mid q, b_{q=q0}, b_{q=q1})$ where $i = 0..2$ is determined as follows:

$$b_i = \begin{cases} 1 & \text{if } q = q_0 \wedge b_{q=q_0} = b_i \\ 1 & \text{if } q = q_1 \wedge b_{q=q_1} = b_i \\ 0 & otherwise \end{cases}$$

In plain language, for a given configuration, $(q, b_{q=q0}, b_{q=q1})$, we match the value q_i assigned to q with the auxiliary variable whose path includes the assignment $q = q_i$. We take the value of that auxiliary variable as the observed value (value of 1), with the other states in b being unobserved (value of 0). Algorithm 4.3 shown below formalizes the above for all configurations of a generic multiplexer node v and parent y.

Algorithm 4.3: SetSwitchCpt(v,y,T,G)

```
1  initialize CPT $P(v|y, u_1, ..., u_k)$ where $\{y, u_1, ..., u_k\}$ is parent set of $v$ in $G$;
2  for each assignment $(v = v', y = y', u_1 = u'_1, ..., u_k = u'_k)$,
3    find $u_i$ in $\{u_1, .., u_k\}$ whose path label $= (path(v), y \in sd_i)$ and $y' \in sdi$;
4    if $v' = u'_i$, $P(v'|y', u'_1, ..., u'_k) = 1$;
5    else, $P(v'|y', u'_1, ..., u'_k) = 0$;
6  return $P(v|y, u_1, ..., u_k)$;
```

The resulting multiplexer node for $b_{\{\}}$ is shown in Figure 4.10.

q	$b_{q=q_0}$	$b_{q=q_1}$	b_0	b_1	b_2
q_0	b_0	b_0	1	0	0
q_0	b_0	b_1	1	0	0
q_0	b_0	b_2	1	0	0
q_0	b_1	b_0	0	1	0
q_0	b_1	b_1	0	1	0
q_0	b_1	b_2	0	1	0
q_0	b_2	b_0	0	0	1
q_0	b_2	b_1	0	0	1
q_0	b_2	b_2	0	0	1
q_1	b_0	b_0	1	0	0
q_1	b_0	b_1	0	1	0
q_1	b_0	b_2	0	0	1
q_1	b_1	b_0	1	0	0
q_1	b_1	b_1	0	1	0
q_1	b_1	b_2	0	0	1
q_1	b_2	b_0	1	0	0
q_1	b_2	b_1	0	1	0
q_1	b_2	b_2	0	0	1

Figure 4.10: CPT for Multiplexer node $b_{\{\}}$

4.3 Demonstration of Exactness

In this section, we demonstrate the exactness of the network transformation by demonstrating the resulting marginals of a transformed segment and the original family are exact. To simplify presentation, we introduce a new example, consisting of three binary variables: u, v, and z. In Figure 4.11, panel (a) presents the BN, panel (b) shows the CPT-tree and panel (c) is the BN segment generated from the CPT-tree. Panels (d) through (f) show the CPTs for each auxiliary node introduced by the transformation.

We aim to demonstrate that $P(z|u, v)$ is equivalent to the transformed BN segment $P(z_{\{\}}, z_{u=u_0}, z_{u=u_1}|u, v)$.

$$P(z|u, v) = \sum_{z_{u=u_0}, z_{u=u_1}} P(z_{\{\}}, z_{u=u_0}, z_{u=u_1}|u, v) \tag{4.1}$$

The chain rule for two random probabilistic events i and j states that $P(i, j) = P(j|i) \times P(i)$. We can apply this rule to the above equation to separate the product into multiple parts.

$$= \sum_{z_{u=u_0}, z_{u=u_1}} P(z_{\{\}}|z_{u=u_0}, z_{u=u_1}, u, v) \times P(z_{u=u_0}|z_{u=u_1}, u, v) \times P(z_{u=u_1}|u, v)$$

Using contextual independence, we know that $z_{u=u_0}$ is only dependent on v and $z_{u=u_1}$ is contextually independent of u and v. We also know that the variable u cannot hold both $u = u_0$ and $u = u_1$ at once. Thus, their auxiliary variables $z_{u=u_0}$ and $z_{u=u_1}$ are independent of each other. This allows us to omit the terms have no effect on the resulting probabilities.

$$= \sum_{z_{u=u_0}, z_{u=u_1}} P(z_{\{\}}|z_{u=u_0}, z_{u=u_1}, u) \times P(z_{u=u_0}|v) \times P(z_{u=u_1})$$

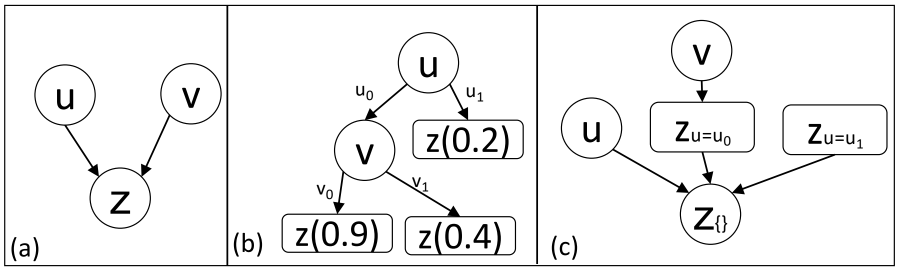
Given a configuration of $(z_{\{\}}, u, v)$, we can now compute the probability of that configuration by expanding the sum and then substituting the values in to each variable in the sum.

$$\begin{aligned}
&= P(z_{\{\}}|z_{u=u_0} = \mathbf{z_0}, z_{u=u_1} = \mathbf{z_0}, u) \times P(z_{u=u_0} = \mathbf{z_0}|v) \times P(z_{u=u_1} = \mathbf{z_0}) \\
&+ P(z_{\{\}}|z_{u=u_0} = \mathbf{z_0}, z_{u=u_1} = \mathbf{z_1}, u) \times P(z_{u=u_0} = \mathbf{z_0}|v) \times P(z_{u=u_1} = \mathbf{z_1}) \\
&+ P(z_{\{\}}|z_{u=u_0} = \mathbf{z_1}, z_{u=u_1} = \mathbf{z_0}, u) \times P(z_{u=u_0} = \mathbf{z_1}|v) \times P(z_{u=u_1} = \mathbf{z_0}) \\
&+ P(z_{\{\}}|z_{u=u_0} = \mathbf{z_1}, z_{u=u_1} = \mathbf{z_1}, u) \times P(z_{u=u_0} = \mathbf{z_1}|v) \times P(z_{u=u_1} = \mathbf{z_1})
\end{aligned}$$

By example, consider the configuration $(z_{\{\}} = z_0, u = u_1, v = v_0)$. We can substitute $z_{\{\}}$ with z_0, u with u_1 and v with v_0 in the above equation and then replace each term with the values from the CPTs shown in Figure 4.11 panels d through f. Note, that the multiplexer CPT will zero-out the entries that include $z_{u=u_0} = z_0$.

$$\begin{aligned}
P(z_{\{\}} = z_0|u = u_1, v = v_0) &= 1 \times 0.9 \times 0.2 \\
&+ 1 \times 0.9 \times 0.8 \\
&+ 0 \times 0.1 \times 0.2 \\
&+ 0 \times 0.1 \times 0.8 \\
&= 0.9
\end{aligned}$$

Repeating this process for each $(z_{\{\}}, u, v)$ configuration yields the same CPT over the original BN. Hence, the resulting marginals are identical.



(d)

$P(z_{u=u_1} = z_0)$	$P(z_{u=u_1} = z_1)$
0.2	0.8

(e)

	$P(z_{u=u_0}\|v)$	
v	z_0	z_1
v_0	0.9	0.1
v_1	0.4	0.6

(f)

			$P(z_{\{\}}\|u, z_{u=u_0}, z_{u=u_1})$	
u	$z_{u=u_0}$	$z_{u=u_1}$	$z = z_0$	$z = z_1$
u_0	z_0	z_0	1	0
u_0	z_0	z_1	1	0
u_0	z_1	z_0	0	1
u_0	z_1	z_1	0	1
u_1	z_0	z_0	1	0
u_1	z_0	z_1	0	1
u_1	z_1	z_0	1	0
u_1	z_1	z_1	0	1

(g)

		$P(z\|u, v)$	
u	v	$z = z_0$	$z = z_1$
u_0	v_0	0.9	0.1
u_0	v_1	0.4	0.6
u_1	v_0	0.2	0.8
u_1	v_1	0.2	0.8

Figure 4.11: (a) BN family with three binary variables u, v, z. (b) CPT-tree with one CSI interaction: z is contextually independent of v when $u = u_1$. (c) Transformed network incorporating the CSI of (b). (d-f) CPTs for each auxiliary node in (c). (g) CPT obtained by normalizing CPT-tree.

4.4 Dependence on Variable Duplications

4.4.1 General Information on Variable Duplications

The efficiency of the CSI transformation algorithm is dependent on the number of variable duplications that occur in the CPT-tree. Before demonstrating the dependence, it is first necessary to specify the *number of variable duplications* on a CPT-tree.

Number of Duplicate Variables *Given a CPT-tree for a child node z and the set of parents $\pi(z)$, we specify* the number of duplicated variables *as the number of non-leaf nodes in the CPT-tree minus the number of parents $|\pi(z)|$.*

In Figure 4.12, we present 3 possible CPT-trees for the same BN family $P(z|x, w, y)$. All variables are binary. Each CPT-tree has a different number of variable duplications: Panel (a) has 0 duplications, panel (b) has 1 duplication, and panel (c) has 3 duplications.

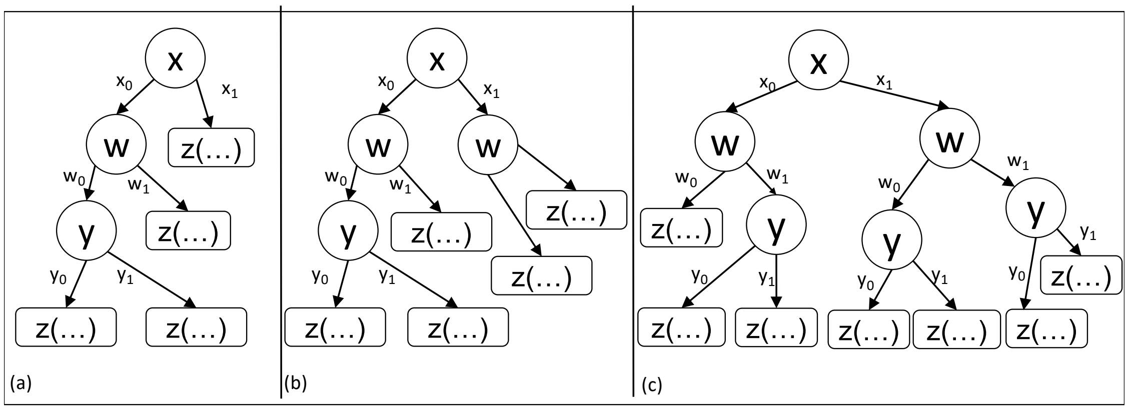


Figure 4.12: Three possible CPT-trees for the same BN family $P(z|x, w, y)$. (a) CPT-tree with 0 duplicated variables. (b) CPT-tree with 1 duplicated variable. (c) CPT-tree with 3 duplications.

Applying network transformation on a CPT-tree with duplicated variables induce loops in the resultant BN segment, which increases the treewidth of the junction tree.

A *loop* is a cycle in the directed acyclic graph of a BN if the directions of edges are ignored. A junction tree with a larger tree width decreases inference efficiency. By contrast, applying network transformation on a CPT-tree with no duplicated variables results in no loops in the the resultant BN segment.

4.4.2 Loop Demonstration

To demonstrate the impact of duplicated variables, we apply the transformation on the CPT-trees shown in panels (a) and (c), and then compile the resultant BN segments into junction trees.

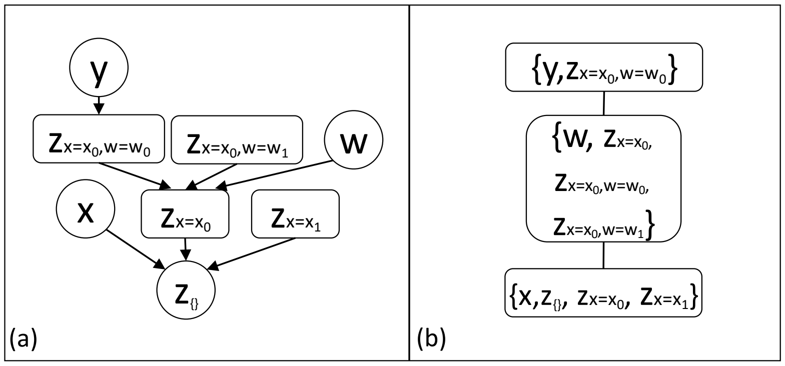
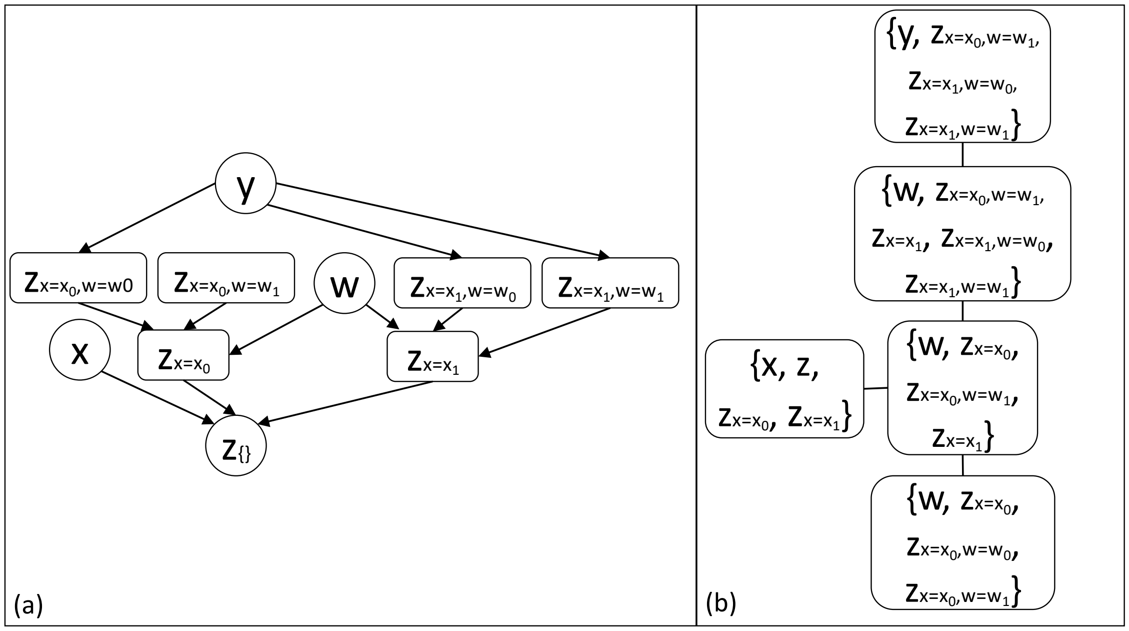


Figure 4.13: (a) Transformed BN segment from CPT-tree with 0 loops in Figure 4.12 (a). (b) Junction tree for BN in panel (a). Separator labels are omitted for readability.

For the CPT-tree with 1 duplicated variable in Figure 4.12, we show the transformed BN segment in panel (a) and the resulting junction tree in panel (b) of Figure 4.13. We observe there are no loops in the BN segment. With no loops, the resulting junction tree has a treewidth of size 3.

For the CPT-tree with 3 duplicated variables in Figure 4.12, we show the transformed BN segment in panel (a) and the resulting junction tree in panel (b) of Figure 4.14. The duplicated variables result in several loops in the transformed segment, such as $\{z_{\{\}}, z_{x=x_1}, w, z_{x=x_0}\}$, leading to a junction tree with a treewidth of size 4.



y
$Z_{x=x_0,w=w0}$
$Z_{x=x_0,w=w_1}$
w
$Z_{x=x_1,w=w_0}$
$Z_{x=x_1,w=w_1}$
x
$Z_{x=x_0}$
$Z_{x=x_1}$
$Z_{\{\}}$
(a)
$\{y, Z_{x=x_0,w=w_1}, Z_{x=x_1,w=w_0}, Z_{x=x_1,w=w_1}\}$
$\{w, Z_{x=x_0,w=w_1}, Z_{x=x_1}, Z_{x=x_1,w=w_0}, Z_{x=x_1,w=w_1}\}$
$\{x, z, Z_{x=x_0}, Z_{x=x_1}\}$
$\{w, Z_{x=x_0}, Z_{x=x_0,w=w_1}, Z_{x=x_1}\}$
$\{w, Z_{x=x_0}, Z_{x=x_0,w=w_0}, Z_{x=x_0,w=w_1}\}$
(b)

Figure 4.14: (a) Transformed BN segment from CPT-tree with 3 loops in Figure 4.12 (c). (b) Junction tree for BN in panel (a). Separator labels are omitted for readability.

4.4.3 Frequency of Loops

The frequency of variable duplications occurring depend on the underlying CPT and the structure of the CPT-tree. A CPT that contains a fewer number of duplicated CPDs will typically result in a greater number of duplications. This can be observed by noting that a CPT with fewer duplicated CPDs will require a greater number of parameters to encode the CPT, which generally results in a more complete CPT-tree, ultimately leading to a greater number of variable duplications in the CPT-tree. Conversely, a CPT with a greater number of duplicated CPDs will typically result in a fewer number of duplications.

Additionally, there is not a unique CPT-tree for each CPT. There exist many possible CPT-tree structures for a given CPT. Some CPT-tree structures may result in a fewer number of variable duplications than others. Suppose, a CPT-tree is testing each value of a ternary variable x individually. A CPT-tree may model this test in various ways, including a (1) multi-valued tree with singleton edges, and (2) binary tree with set-valued edges. A CPT-tree that uses a multi-valued approach with singleton edges, segment shown in Figure 4.15 (a), will have 3 outgoing edges (one edge for each $x_i \in dom(x)$) and 0 variable duplications. In comparison, a binary CPT-tree, segment shown in Figure 4.15 (b), with set-valued edges will have 1 variable duplication. This variable duplication occurs as it tests x twice: Once to partition $\{x_0, x_1, x_2\}$ into $\{\{x_0, x_2\}, \{x_1\}\}$, and again to partition $\{x_0, x_2\}$ into $\{\{x_0\}, \{x_2\}\}$. Hence, it is possible for a CPT-tree to express the same CPT with different amounts of variable duplication. We evaluate the impact of varying levels of duplication on inference runtime in Chapter 6.

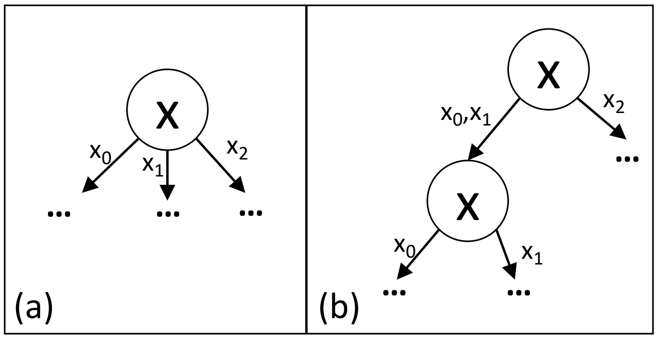


Figure 4.15: CPT-tree segments over ternary variable x. (a) Multi-valued tree with 0 duplicated variables. (b) Binary tree with multi-valued edges and 1 duplication.

Chapter 5

Mixed NAT-CSI Bayesian Networks

This chapter outlines a BN representation modelled with both NAT and CSI models, and an accompanying compilation method to support efficient inference. The chapter is structured as follows: Section 5.1 details the BN representation composed of both local models. Section 5.2 discusses the method to support efficient inference.

5.1 Representation

Causal independence models and CSI models can each be exploited to improve space and inference efficiency in BNs. To our knowledge, no prior study has considered inference on BNs that take advantage of both simultaneously. Combining the models results in several issues, which we address below:

First, we note that causal independence models and CSI models both apply to individual families in BNs. Thus, it is plausible that the models can both coexist in the same environment, and consequently the same BN. For example, a patient's recovery from surgery may be dependent on whether or not they use physiotherapy, the skill of the physiotherapist, and their use of medicine. The patient's use of medicine may in-turn be dependent on three medicines, which may counteract. The recovery from surgery variable may be modelled by a CSI model while the variable indicating the use of medicine may be modelled by a causal independence model. Hence, it is reasonable that the CSI modelled variable (recovery from surgery) is dependent on a NAT modelled variable (use of medicine). The coexistence may also occur in reverse (i.e., NAT modelled variable is dependent on a CSI modelled variable), or as two conditionally independent variables in an environment.

Second, a suitable representation is needed for each type of local model. In this thesis, we adopt NAT models as our causal independence model and CPT-trees as our CSI model. The NAT model was selected due to its ability to encode both reinforcing and undermining interactions. The CPT-tree model was selected due to its wider support of inference methods. To avoid digressing, refer to Chapter 2 for details. We define a BN modelled with both NAT models and CPT-trees as a *mixed NAT-CSI BN*.

Mixed NAT-CSI Bayesian Network (MNCBN) *A MNCBN is a BN* (M, G, P)*, specified in terms of the following:*

- M *is a set of variables.*
- G *is a directed acyclic graph whose nodes correspond one-to-one to members of* M*. Each variable in the graph is conditionally independent of its non-descendants given its parents.*
- P *is a set of CPTs partitioned into the triplet* (TC, NM, CT) *where* TC *is a set of tabular CPTs, NM is a set of NAT models, and CT is a set of CPT-trees.*

An example MNCBN is shown in Figure 5.1. The directed acyclic graph is shown in panel (a). All variables are ternary where each variable has the domain $\{s_0, s_1, s_2\}$. The MNCBN consists of 18 nodes, of which all nodes whose labels are prefixed by v (16 in total) are modelled by tabular CPTs. The remaining 2 nodes h and g are CPT-tree and NAT modelled, respectively. The CPT-tree is shown in panel (b) while the NAT model is shown in panel (c).

Third, a MNCBN does not support the use of typical BN inference algorithms due to the presence of the non-tabular local models, nor does it support the use of an alternative processing designed for one local model on the other due to the orthogonality of the local models. Thus, it is necessary to identify a novel approach that supports inference on MNCBNs. We introduce a novel inference framework designed for MNCBNs below.

5.2 Inference Framework

In this section, we outline an framework that prepares MNCBNs for inference, select an inference method, and demonstrate the framework on an example MNCBN.

5.2.1 Framework Outline

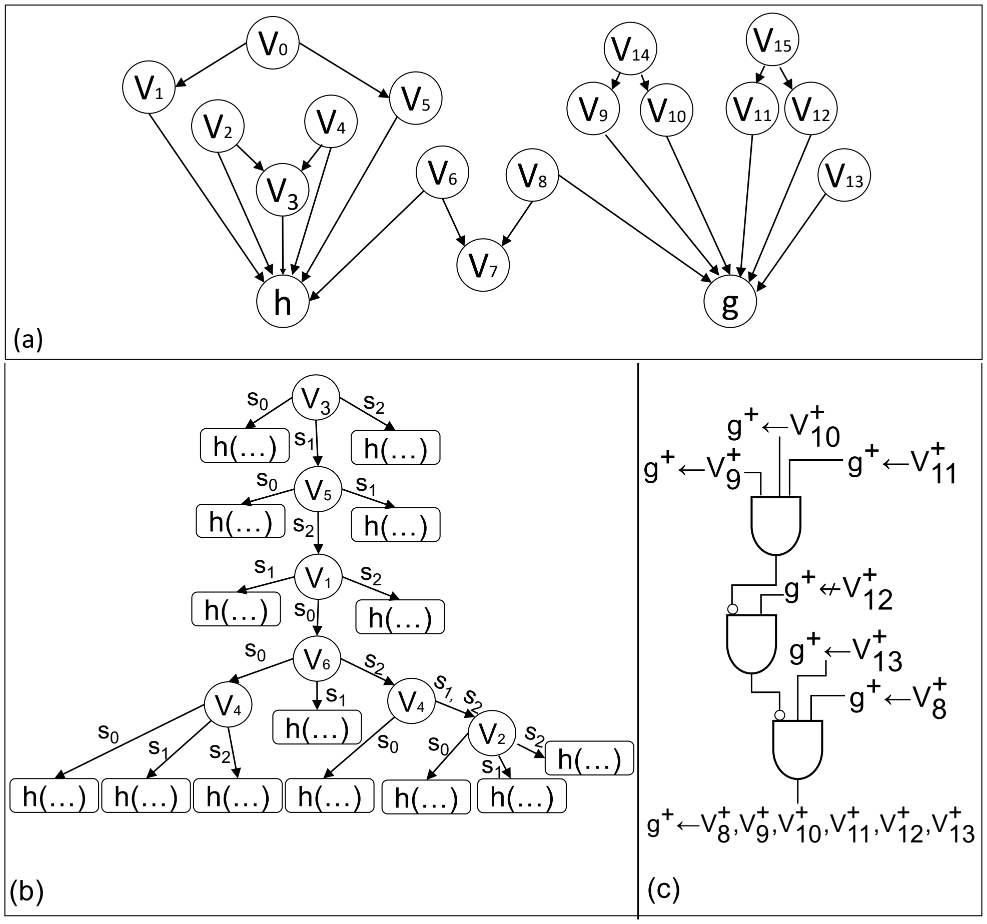


Figure 5.1: (a) MNCBN DAG. (b) NAT model for family of g. (c) CPT-tree over family of h.

The inference framework for MNCBN applies an alternative processing method that supports inference on NAT modelled families and a separate alternative processing method on CSI modelled families. However, the choice of method is critical:

Both methods must not conflict with each other, such that the usage of one method precludes the usage of the other.

In this thesis, we have selected to use de-causalization for the NAT models and network transformation for the CPT-trees. These methods do not conflict since they both convert their respective local model into BN segments. Both methods make no assumptions on the evidence, allowing for the evidence to be changed without re-compilation. Both methods also support the computation of posterior probabilities of all variables. By comparison, applying multiplicative factorization for NAT models and a rule-based representation for CSI would conflict since neither method supports compiling into a common structure.

The MNCBN can be converted to a standard BN: Each NAT model is de-causalized into a BN segment that maintains computational savings. Each CSI model is similarly network transformed into a BN segment that maintains computational savings. Each segment is connected into the BN. To connect a BN segment into the BN, we replace the child BN node and all parent BN nodes of the family with the segment. We then reconnect the edges to maintain exactness: All incoming edges to the parent nodes and all outgoing edges from the child node should be restored.

After de-causalizing, transforming, and connecting all segments, the result is a standard BN where each local model is of tabular form, while preserving the computational savings of the local models. We refer to this result as a *de-causalized and transformed BN* (DTBN). The resulting DTBN is subsequently compatible with many standard BN inference algorithms.

One thing to note is that evidence can only be specified on the observed variables. The auxillary variables introduced by both de-causalization and network transformation are said to be unobservable. Hence, observations may only be entered on the nodes that were in the original MNCBN, prior to the conversion into a standard BN.

5.2.2 Producing Lazy Junction Trees from DTBNs

In this work, we make use of the Lazy Propagation algorithm due to its ability to efficiently compute posterior probabilities of all observable variables at inference run-time. Lazy propagation can be directly applied to a DTBN with no further changes. Refer to Sections 2.5.4 and 2.5.5 for further details on the lazy propagation algorithm.

5.2.3 Framework Demonstration

Consider the MNCBN shown in Figure 5.1 with the DAG shown in panel (a). All variables are ternary where each variable v_i has the domain $dom(v_i) = \{s_0, s_1, s_2\}$. The MNCBN has one CPT-tree modelled node h with its local model shown in panel (b). The MNCBN has one NAT modelled node g with its local model shown in panel (c).

We convert the MNCBN given in Figure 5.1 into a DTBN by de-causalizing node g and network transforming node h. The resulting DTBN is shown in Figure 5.2. Nodes prefixed with x, y, or q are auxillary nodes introduced by the de-causalization. Nodes prefixed with h (except the child node itself $h_{\{\}}$) are auxillary nodes introduced by the network transformation.

At first glance, the DTBN Figure 5.2 may appear to be more complex than the MNCBN in Figure 5.1. The number of nodes has significantly increased — from 18 nodes in the MNCBN to 42 nodes in the DTBN. This is a result of the de-causalization and transformation's process, which introduce auxillary variables in order to encode the causal independence models in a BN segment.

However, the increase in the number of nodes is offset by a decrease in the size of the largest CPT. Since the size of the largest CPT enforces a lower bound on the inference efficiency, it follows that a smaller maximum CPT size increases the efficiency of inference. In the running example, if all CPTs are tabular, the MNCBN in Figure 5.1 has a total CPT size of 4506 probabilities, where the largest CPT has a size of 2187 probabilities. By comparison, the DTBN in Figure 5.2 has a total CPT size of 1603 probabilities, where the largest CPT has a size of 243. This amounts to a 64% decrease in the total number of probabilities necessary to specify the BN.

Once the MNCBN is converted into a DTBN, the framework compiles the DTBN in Figure 5.2 into a lazy junction tree in order to perform lazy propagation inference. Figure 5.3 shows the lazy junction tree obtained by compiling the MNCBN if all CPTs are tabular. The junction tree maintains a treewidth of 6. By comparison, Figure 5.4 shows the lazy junction tree obtained by compiling the DTBN. The junction tree maintains a treewidth of 4. We evaluate the efficiency of this framework in Chapter 6.

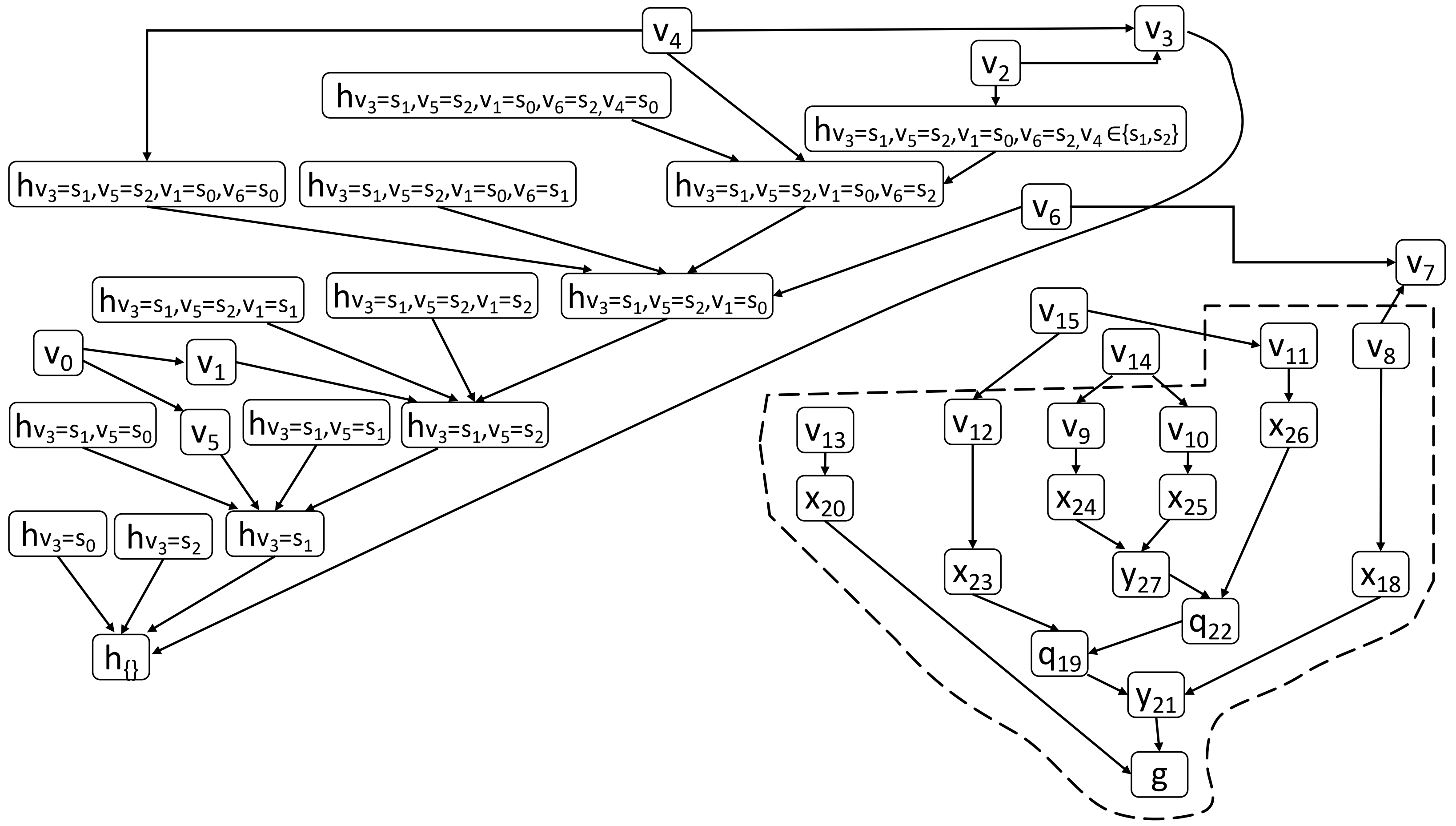


Figure 5.2: DTBN of the MNCBN in Figure 5.1

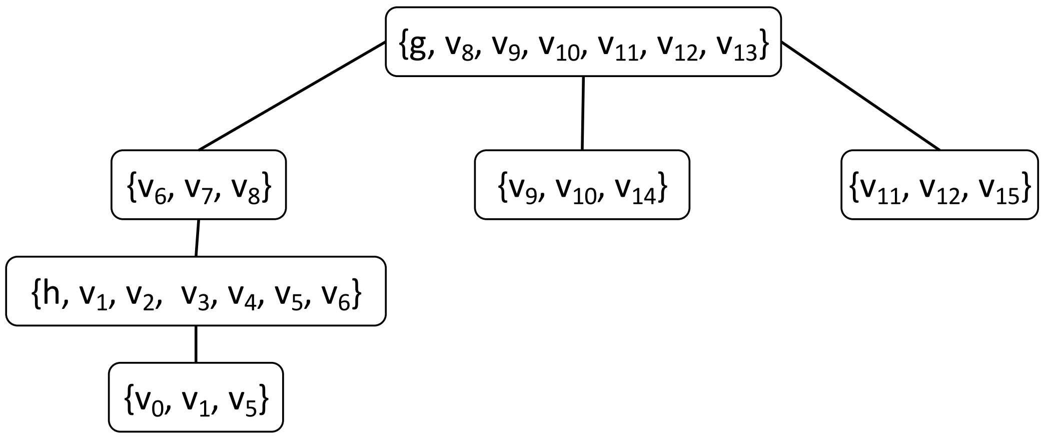
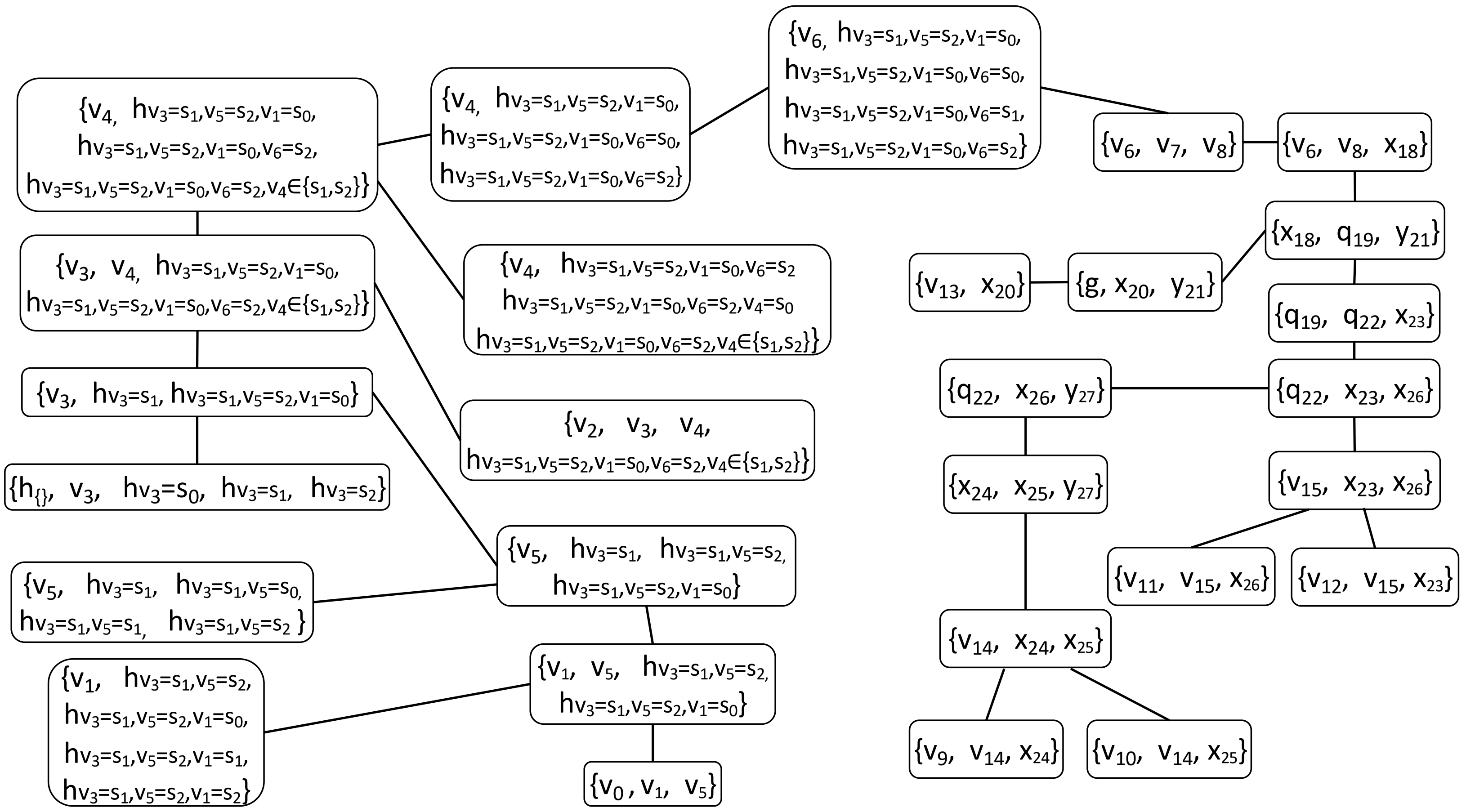

{g, v_8, v_9, v_{10}, v_{11}, v_{12}, v_{13}}
{v_6, v_7, v_8}
{v_9, v_{10}, v_{14}}
{v_{11}, v_{12}, v_{15}}
{h, v_1, v_2, v_3, v_4, v_5, v_6}
{v_0, v_1, v_5}

Figure 5.3: Lazy junction tree obtained by compiling MNCBN in Figure 5.1 (a) if all CPTs are tabular.



$\{v_4, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0,v_6{=}s_2, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0,v_6{=}s_2,v_4{\in}\{s_1,s_2\}\}$
$\{v_4, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0,v_6{=}s_0, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0,v_6{=}s_2\}$
$\{v_6, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0,v_6{=}s_0, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0,v_6{=}s_1, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0,v_6{=}s_2\}$
$\{v_6, v_7, v_8\}$
$\{v_6, v_8, x_{18}\}$
$\{x_{18}, q_{19}, y_{21}\}$
$\{v_{13}, x_{20}\}$
$\{g, x_{20}, y_{21}\}$
$\{q_{19}, q_{22}, x_{23}\}$
$\{v_3, v_4, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0,v_6{=}s_2,v_4{\in}\{s_1,s_2\}\}$
$\{v_4, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0,v_6{=}s_2\ hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0,v_6{=}s_2,v_4{=}s_0\ hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0,v_6{=}s_2,v_4{\in}\{s_1,s_2\}\}$
$\{v_3, hv_{3}{=}s_1, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0\}$
$\{q_{22}, x_{26}, y_{27}\}$
$\{q_{22}, x_{23}, x_{26}\}$
$\{h_{\{\}}, v_3, hv_{3}{=}s_0, hv_{3}{=}s_1, hv_{3}{=}s_2\}$
$\{v_2, v_3, v_4, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0,v_6{=}s_2,v_4{\in}\{s_1,s_2\}\}$
$\{x_{24}, x_{25}, y_{27}\}$
$\{v_{15}, x_{23}, x_{26}\}$
$\{v_5, hv_{3}{=}s_1, hv_{3}{=}s_1,v_5{=}s_2, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0\}$
$\{v_5, hv_{3}{=}s_1, hv_{3}{=}s_1,v_5{=}s_0, hv_{3}{=}s_1,v_5{=}s_1, hv_{3}{=}s_1,v_5{=}s_2\}$
$\{v_{11}, v_{15}, x_{26}\}$
$\{v_{12}, v_{15}, x_{23}\}$
$\{v_{14}, x_{24}, x_{25}\}$
$\{v_1, hv_{3}{=}s_1,v_5{=}s_2, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_1, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_2\}$
$\{v_1, v_5, hv_{3}{=}s_1,v_5{=}s_2, hv_{3}{=}s_1,v_5{=}s_2,v_1{=}s_0\}$
$\{v_0, v_1, v_5\}$
$\{v_9, v_{14}, x_{24}\}$
$\{v_{10}, v_{14}, x_{25}\}$

Figure 5.4: Lazy junction tree obtained by compiling DTBN in Figure 5.2

Chapter 6

Experimental Evaluation

Three experiments were conducted. The first, presented in Section 6.1, confirmed the coexistence of NAT and CSI models in real-world BNs. The second, presented in Section 6.2, evaluated the computational gain achieved by mixing NAT and CSI local models. The third, presented in Section 6.3, compares effectiveness of NAT and CSI models. The second and third experiments used the WebWeavr software toolkit [19].

6.1 Coexistence of NAT & CSI Models in Real-World BNs

The first experiment aimed to confirm the coexistence of NAT and CSI local models in real-world BNs. If one (or both) of the models do not exist in the real-world, then the usefulness of this research is limited to synthetic MNCBNs. If both models exist, then the research can be applied in practice. The existence of NAT models in 8 real-world BNs was positively identified with reasonable inference errors by [22], which we summarized in Section 2.7.6. In this section, we demonstrate the existence of CSI models in real-world BNs.

6.1.1 Experimental Setup

This was achieved by testing the existence of CSI in 2 of the 8 real-world BNs from the NAT modelling study. The BNs selected were *Andes*, which models a physics tutoring system, and *Win95pts*, which models a printer diagnostics system [16]. The BNs were selected because they are entirely binary-valued; the other 6 BNs from the

NAT modelling study were multi-valued. The Andes BN was modified to remove 3 isolated nodes (nodes with no parents nor children). We denote the modified BN as Andes$^-$.

We apply the clustering approach discussed Section 3.3 on all families in each BN, which has 2 or more parents. Recall, the clustering algorithm groups probabilities into clusters based on a distance bound δ. Each cluster has a maximum inner-cluster cluster distance of δ. Each cluster has a minimum inter-cluster distance of δ. For this experiment, we use a distance bound of $\delta = 0.02$.

6.1.2 Experimental Results

The clustering results are presented in Table 6.1. Column 1 indicates the BN tested. Column 2 indicates the total number of nodes in the BN. Column 3 indicates the total number of families processed: The number of nodes with 2 or more parents. Column 4 indicates the maximum number of parameters per CPT over the BN. Column 5 indicates the maximum number of clusters of all CPTs. Column 6 indicates the total Euclidean distance of each clustered CPT from its source CPT.

BN	#Node	#Fam Proced	Max #Par/CPT	Max #Clu/CPT	Eu Dist
Andes$^-$	223	50	64	3	0
Win95pts	76	24	128	6	0.041

Table 6.1: Summary of results from clustering Andes$^-$ and Win95pts BNs.

The results confirm the existence of CSI. The Andes$^-$ BN expressed a significant amount of CSI: With only 3 parameters, the CPT-tree representation can exactly encode (no error) a CPT of 64 parameters. This is due to every cluster of each CPT containing identical member values. Similarly, the Win95pts BN expressed a significant amount of CSI with an error of 0.041. This is due to 22 of the 24 CPTs having no error, with the remaining 2 having distances of 0.001 and 0.04.

Other studies [2, 7, 18] have also identified CSI in real-world environments, such as: Biological datasets, other BNs in the *bnlearn* repository and machine learning datasets from the UCI repository.

Hence, the NAT and CSI models have each been identified in practice: The NAT modelling study identified NAT models in practice. The clustering results and the other referenced studies suggest the existence of CSI models in practice. These independent findings have applied the same local modelling on all suitable families in a given BN. It follows that if these models exist on all suitable families, then they also exist on a subset of all suitable families. It is also noted that a particular BN family's NAT and CSI modelling error can be compared. Thus, the set of suitable families in a BN can be partitioned into 2 disjoint subsets: The first contains all families best modelled by a NAT model, and the second contains all families best modelled by a CSI model. This suggests that real-world BNs may be modelled by a mixture of NAT and CSI models.

6.2 Computational Gain of Mixing NAT & CSI

6.2.1 Experimental Setup

The objective of the second experiment was to evaluate the space complexity by mixing NAT and CSI models. We generated mixed NAT-CSI Bayesian networks (MNCBN) over 100 binary or ternary variables. Of the families of 2 parents or more, 50% are NAT modelled while the remaining 50% are CSI modelled. The maximum number of parents per node is 12, with at least 2 families having exactly 12 parents. Of the families with 12 parents, at least 1 is NAT modelled and at least 1 is CPT-tree modelled. We generated 300 distinct MNCBNs, one for each combination of the 3 parameters:

- Number of variable duplications (k): 0, 2, 4, 7, 10
- Density beyond being singly connected (d): 5%, 10%

- BN topology: 30 randomly generated

Each MNCBN is then converted into 4 standard BNs (encoding the same JPD) by the following methods:

D+T De-causalizing NAT models and transforming CPT-trees

D+N De-causalizing NAT models and normalizing CPT-trees

N+T Normalizing NAT models and transforming CPT-trees

N+N Normalizing NAT models and normalizing CPT-trees

Each resultant BN is compiled for inference by lazy propagation. Each BN has 10 inference runs, each with different observations over 10 randomly selected variables. Inference runs by BNs from the same MNCBN resulted in the same posterior marginals.

6.2.2 Experimental Results

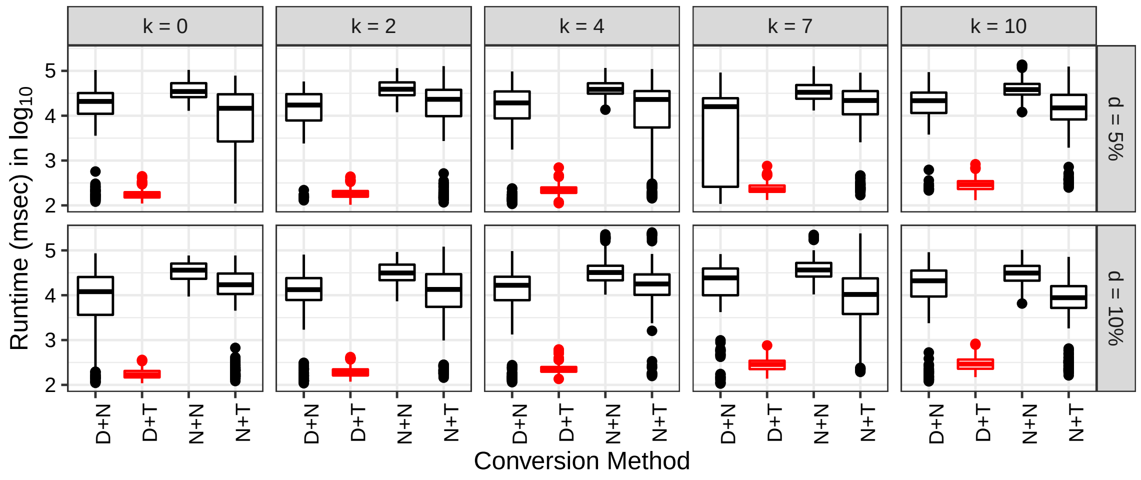


Figure 6.1: Comparison of inference runtimes of 4 conversion methods

Figure 6.1 has 10 panels: 1 panel for each (k, d) combination. Each panel has 4 boxes, 1 for each conversion method (D+T, D+N, N+T, N+N). Each box represents 300 inference runs (30 topologies $\times 1$ conversion method $\times 10$ inference runs).

The D+T approach introduced by this thesis is highlighted in red. Runtimes were obtained using a desktop with 2.9GHz clock speed.

It can be observed that N+N is the slowest in all (k, d) combinations, and that both D+N and N+T improve on the N+N approach. However, the relative performance between D+N and N+T is indiscernible. In 4 of the 10 (k, d) combinations: (0, 5%), (7, 10%), (10, 5%), (10, 10%), D+N has a greater mean inference runtime than N+T. In the remaining 6 combinations, N+T has a greater mean inference runtime than D+N. This could be partly due to the presence of normalization in both conversion methods.

Both de-causalization and network transformation tend to result in compiled structures with a smaller maximum number of parents and treewidth. Applying normalization on local models results in compiled structures with a larger maximum number of parents and treewidth. Since the inference efficiency is bounded by the maximum number of parents and treewidth, it is possible that the de-causalized or transformed families admit some efficiency savings but the remaining inefficient normalized families set the bounds on the inference efficiency. It follows that evaluating the relative performance between the D+T and N+T conversion methods may be comparing the normalization, rather than the de-causalization and transformation approaches. The third experiment eliminates the confounding variable of normalization to investigate the relative gain from alternative models.

Moreover, D+T is on average two orders of magnitude faster than alternatives, which clearly demonstrates the computational advantage obtained when exploiting both NAT and CSI in MNCBNs. Based on the same logic as above, the speedup of the D+T conversion method suggests that the removal of normalization reduces the bounds of inference efficiency.

6.3 Performance of De-causalization vs. Transformation

6.3.1 Experimental Setup

The objective of the third experiment was to directly compare de-causalization and network transformation without the added noise of normalization. In this experiment, we generated BNs in two steps. First, we generated only DAGS with 200 variables each (binary or ternary). The largest number of parents per node is 12, and each DAG has at least 4 such families. We generated 300 distinct DAGs, one for each combination of the three parameters:

- Number of variable duplications (k): 0, 2, 4, 7, 10
- Density beyond being singly connected (d): 5%, 10%
- BN topology: 30 randomly generated

Second, a pair of Bayesian networks are created from each DAG: a NAT-modelled Bayesian network (NMBN) and a CPT-tree modelled Bayesian network (CMBN). The NMBN is generated by modelling all families with 2 or more parents with NAT models. The CMBN is generated by modelling all families with 2 or more parents with CPT-trees. Families with less than 2 parents are left as tabular CPTs. Hence, the pair of Bayesian networks have the same DAG, but differ in JPDs.

Each NMBN is de-causalized and each CMBN is network transformed. Each resultant BN is compiled for inference by lazy propagation. Ten inference runs are performed on each BN with random observations over 20 randomly selected variables.

6.3.2 Experimental Results

Figure 6.2 contains two panels that compare the $\log_{10}$ inference runtimes of the NMBNs against the CMBNs. The left panel shows the inference runtime with a density beyond singly connected of 5% while the right panel shows a density beyond

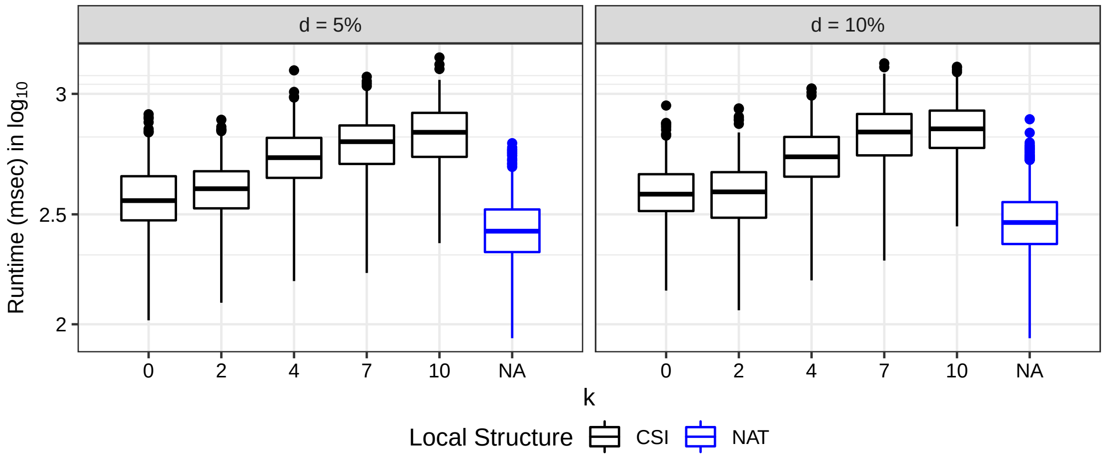


Figure 6.2: Summary of inference runtimes by NMBNs and CMBNs

singly connected of 10%. Each panel contains 6 boxes: 5 black boxes and 1 blue box. Each black box shows the inference runtime distribution of the 30 CMBNs produced from a single (k, d) combination. Each blue box shows the inference runtime distribution of the 30 NMBNs produced for a given d value. The black boxes are arranged in ascending order of number of duplicated variables, such that the boxes furthest left have the most efficient CPT-trees. The runtimes were recorded on a desktop with a 2.9GHz clock speed.

In both panels ($d = 5\%$ and $d = 10\%$), inference runtime of NMBNs are the least, even compared to the most efficient CMBNs that consist of CPT-trees with no duplicated variables ($k = 0$). This is observed in Figure 6.2 as the NMBN (blue box) is below the leftmost CMBN (black box) in both panels. We note the difference in efficiency between CMBN and NMBNs may be explained by two limitations of the network transformation approach.

First, the number of parents of a transformation's multiplexer node is always greater than the number of parents of a de-causalized node. Consider a CPT-tree transformed BN segment with one multiplexer node m that switches on the parent node n. Assuming all arcs in the source CPT-tree are singly-valued (i.e., one parent value per arc), the multiplexer node m will have $dom(n) + 1$ parents. That is, one parent to encode the original variable n, plus one parent to encode each state of $dom(n)$. In comparison, every node in a de-causalized BN segment is guaranteed to have at most 2 parents [25]. A greater maximum number of parents generally decreases inference efficiency.

Second, a de-causalized BN segment is guaranteed to be loop free. By contrast, a CPT-tree with duplicated variables will induce loops, which raises the treewidth of the transformed structure (Section 4.4). A higher treewidth generally decreases inference efficiency. Hence, the greater maximum number of parents and the larger treewidth suggest that NAT modelled Bayesian networks are generally more efficient than CPT-tree modelled networks.

Furthermore, we note that larger k values in Figure 6.2 correspond to longer inference runtimes. This confirms the expected results discussed in Section 4.4. While a larger d value corresponds to longer inference as well, the impacts of increasing density from 5 to 10% beyond singly connected appear to have relatively less of an impact than increasing the k value.

In summary, we confirmed the coexistence of NAT and CSI models in real-world BNs. Next, we demonstrated the D+T conversion method that exploits both local models is two orders of magnitude faster than all other conversion methods. A possible explanation for the speedup is that the D+N and N+T conversion methods, which exploit one local model are subject to normalization on the other local model. It is plausible that the exploitation of one local model admits some efficiency savings but the normalized families enforce the lower bound of inference efficiency. Lastly, we evaluated the relative performance between the de-causalization and normalization approaches where we observed inference runtime of NAT-modelled BNs to be least, even relative to the most efficient CPT-trees with no duplicated variables.

Chapter 7

Conclusion

In this chapter, we summarize the key contributions of this thesis and offer some areas for future research.

7.1 Summary of Contributions

This thesis has introduced MNCBNs, a new representation for exploiting both CSI and causal independence in the same environment. The representation makes use of NAT models as its causal independence model and CPT-trees as its CSI model. An inference framework designed for this representation facilitates efficient inference by combining de-causalization on the NAT models and network transformation on the CPT-tree models in the MNCBN. This avoids the previous requirement normalizing one model to an exponentially sized tabular CPT. The inference framework has been shown to be both exact and efficient, resulting in a 2 times order of magnitude speed up for inference tasks on low density networks.

We demonstrated the necessity of this research by confirming that neither model was able to efficiently and exactly encode the other model. This indicates that an inference approach designed for one model cannot be applied to the other while maintaining exactness and efficiency.

We extended the work of Boutillier et al. [3] by formalizing the network transformation algorithm suite. This facilitates the conversion of a CPT-tree into a tabular modelled BN segment that preserves context-specific independencies. This BN segment is then compatible with many standard BN algorithms.

We have explored the coexistence of NAT and CSI models in real-world BNs. This experiment combined previous work that demonstrated the existence of NAT models on 8 real-world BNs, with our study of identifying CSI on 2 real world BNs to positively confirm the coexistence in real-world BNs. Our study applied the clustering algorithm on the real-world BNs to identify how many groups of values would be needed to express the CPT. We found that both CPTs expressed a significant amount of CSI allowing for a concise expression by a CSI representation. Other work has also identified CSI in real-world datasets. Collectively, these studies confirm the coexistence of NAT and CSI in real-world BNs.

Some material forming the core of this thesis has been published and presented at the 2020 Canadian Artificial Intelligence conference [15].

7.2 Future Work

The first area of future work could explore the compression of tabular CPTs into CPT-trees. In this work, we make use of a clustering algorithm to estimate the number of parameters needed to specify a CPT-tree. Future work can extend these findings by learning a CPT-tree topology from a tabular CPT. It is likely the topology could be identified by extending common decision tree learning algorithms to CPT-trees. It is noted that some work [18] has been conducted in this area but its learning is restricted to binary trees with multi-valued arcs. It was demonstrated in Section 4.4 that if a CPT-tree tests each attribute individually, this restriction results in a greater number of variable duplications. Hence, a CPT-tree learning algorithm supporting multi-valued trees is desired.

Building off the first area, the second area of future work could explore learning MNCBNs from raw data. This would likely require first identifying the dependence structure (the directed acyclic graph) then identifying the optimal local model for each family. One possible way to identify the optimal local model for each family would be to compress each tabular CPT into both a NAT model and a CSI model. The hypothetical algorithm would then evaluate the number of parameters and approximation error of the NAT and CSI models against the tabular CPT, and assign the model which best fits the family.

The final area of future work could be to evaluate the ability of CSI to model multi-valued NAT CPTs. In this work, we conducted experiments over binary NAT CPTs and outlined an approach to extend clustering to larger domains. It is predicted that clustering multi-valued NAT CPTs will identify less CSI due to the stricter conditions that must be met for CSI to exist, relative to the clustering of binary NAT CPTs. Further experimental evaluation over multi-valued NAT CPTs would substantiate the prediction.

Bibliography

[1] S. Andersen, K. Olesen, F. Jensen, and F. Jensen. Hugin-a shell for building Bayesian belief universes for expert systems. In *IJCAI*, volume 89, pages 1080–1085, 1989.

[2] Y. Barash and N. Friedman. Context-specific Bayesian clustering for gene expression data. *Journal of Computational Biology*, 9:169–191, 2002.

[3] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in Bayesian networks. In *Proc. 12th Conf. on Uncertainty in Artificial Intelligence*, pages 115–123, 1996.

[4] M. Chavira and A. Darwiche. Compiling Bayesian networks using variable elimination. In *Proc. 20th International Joint Conf. on Artificial Intelligence*, pages 2443–2449, 2007.

[5] F. Cozman. Generalizing variable elimination in Bayesian networks. In *Workshop on probabilistic reasoning in artificial intelligence*, pages 27–32, 2000.

[6] N. Friedman and M. Goldszmidt. Learning Bayesian networks with local structure. In *Proc. of the 12th Conf. on Uncertainty in Artificial Intelligence*, pages 252–262, 1996.

[7] N. Friedman and Z. Yakhini. On the sample complexity of learning Bayesian networks. In *Proc. 12th Conf. on UAI*, pages 274–282. Morgan Kaufmann, 1996.

[8] M. Henrion. Practical issues in constructing a Bayes' belief network. In *Proc. 3rd Conf. on Uncertainty in Artificial Intelligence*, pages 132–139, 1987.

[9] S. Kullback and R. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1):79–86, 1951.

[10] P. Maaskant and M. Druzdzel. An independence of causal interactions model for opposing influences. In *Proc. 4th European Workshop on Probabilistic Graphical Models*, pages 185–192, 2008.

[11] A. Madsen and F. Jensen. Lazy propagation: a junction tree inference algorithm based on lazy evaluation. *Artificial Intelligence*, 113(1):203–245, 1999.

[12] J. Pearl. *Probabilistic reasoning in intelligent systems: networks of plausible inference.* Morgan Kaufmann Publishers Inc., 1988.

[13] D. Poole. Probabilistic partial evaluation: exploiting rule structure in probabilistic inference. In *IJCAI*, volume 97, pages 1284–1291, 1997.

[14] D. Poole. Context-specific approximation in probabilistic inference. In *Proc. of the 14th Conf. on Uncertainty in Artificial Intelligence*, pages 447–454. Morgan Kaufmann Publishers Inc., 1998.

[15] M. Roher and Y. Xiang. Mixing ICI and CSI for more efficient probabilistic inference. In C. Goutte and X. Zhu, editors, *Advances in Artificial Intelligence*, LNAI 12109, pages 451–463. Springer, 2020.

[16] M. Scutari. Bayesian network repository, 2007.

[17] M. Takikawa and B. D'Ambrosio. Multiplicative factorization of noisy-MAX. *Computing Research Repository*, 2013.

[18] T. Talvitie, R. Eggeling, and M. Koivisto. Learning Bayesian networks with local structure, mixed variables, and exact algorithms. *International Journal of Approximate Reasoning*, 115:69–95, 2019.

[19] Y. Xiang. WebWeavr-IV Research Toolkit. www.cis.uoguelph.ca/˜yxiang/.

[20] Y. Xiang. *Probabilistic reasoning in multiagent systems: a graphical models approach.* 2002.

[21] Y. Xiang. Non-impeding Noisy-AND tree causal models over multi-valued variables. *International Journal of Approximate Reasoning*, 53(7):988–1002, 2012.

[22] Y. Xiang and B. Baird. Compressing Bayesian networks: swarm-based descent, efficiency, and posterior accuracy. In E. Bagheri and J. Cheung, editors, *Canadian Conference on AI*, LNAI 10832, pages 3–16, 2018.

[23] Y. Xiang and N. Jia. Modeling causal reinforcement and undermining for efficient CPT elicitation. *IEEE Trans. Knowledge and Data Engineering*, 19(12):1708–1718, 2007.

[24] Y. Xiang and Y. Jin. Multiplicative factorization of multi-valued NIN-AND tree models. In *Proc. 29th International FLAIRS Conference*, pages 680–685, 2016.

[25] Y. Xiang and D. Loker. De-causalizing NAT-Modeled Bayesian networks for inference efficiency. In E. Bagheri and J. Cheung, editors, *Advances in Artificial Intelligence*, LNAI 10832, pages 17–30. Springer, 2018.