ON THE USABILITY OF CONTINUOUS TIME BAYESIAN NETWORKS: IMPROVING SCALABILITY AND EXPRESSIVENESS

by

Logan Jared Perreault

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

March, 2017

DEDICATION

To Samantha, my loving wife

ACKNOWLEDGEMENTS

I wish to thank my advisor, Dr. John Sheppard for his essential role in my journey through the PhD program. His passion for machine learning convinced me to join the graduate program, and his continued support enabled me to see it through to the end. I also thank Dr. Brittany Fasy, Dr. Brendan Mumey, and Dr. Qing Yang for their help throughout my education at MSU.

I thank the past and present members of the Numerical Intelligent Systems Laboratory for making research an enjoyable experience. I especially thank Monica Thornton, who acted as my co-author and friend for the vast majority of my time spent researching at MSU. I also wish to thank Shane Strasser, who inspired me to be a better researcher, developer, and person. This dissertation would not be what it is today without the lengthy and unsolicited LaTeX lectures that Shane provided.

Thank you Samantha for your endless patience during my time in the graduate program. I recognize the burden that I placed on our family by pursuing my degree, and I understand the sacrifices you made to give me the opportunity. Thank you Mom and Dad for your support during my long education, both financially and emotionally. I know you have never fully understood the desire to pursue a PhD, which is why I appreciate your unwavering support all the more.

Funding Acknowledgment

This work was supported in part under the NASA contract number NNX14CJ45C, QSI Job number SR1429.

TABLE OF CONTENTS

TABLE OF CONTENTS - CONTINUED

TABLE OF CONTENTS - CONTINUED

LIST OF TABLES

LIST OF FIGURES

Figure Page

LIST OF FIGURES - CONTINUED

Figure Page

LIST OF FIGURES - CONTINUED

ABSTRACT

The Continuous Time Bayesian Network (CTBN) is a model capable of compactly representing the behavior of discrete state systems that evolve in continuous time. This is achieved by factoring a Continuous Time Markov Process using the structure of a directed graph. Although CTBNs have proven themselves useful in a variety of applications, adoption of the model for use in real-world problems has been limited. We believe this is due in part to limitations relating to scalability as well as representational power and ease of use. This dissertation attempts to address these issues.

First, we improve the expressiveness of CTBNs by providing procedures that support the representation of non-exponential parametric distributions. This allows for the specification of any positive and continuous parametric distribution, which increases the number of processes that can be described using CTBNs. We also propose the Continuous Time Decision Network (CTDN) as a framework for representing decision problems using CTBNs. This new model supports optimization of a utility value as a function of a set of possible decisions. The CTDN further improves expressiveness of CTBNs, thereby extending the set of problems that can be addressed using this model.

Next, we address the issue of scalability by providing two distinct methods for compactly representing CTBNs by taking advantage of similarities in the model parameters. These compact representations are able to mitigate the exponential growth in parameters that CTBNs exhibit, allowing for the representation of more complex processes. We then introduce another approach to managing CTBN model complexity by introducing the concept of disjunctive interaction for CTBNs. Disjunctive interaction has been used in Bayesian networks to provide significant reductions in the number of parameters, and we have adapted this concept to provide the same benefits within the CTBN framework.

Finally, we demonstrate how CTBNs can be applied to the real-world task of system prognostics and diagnostics. We show how models can be built and parameterized directly using information that is readily available for diagnostic models. We then apply these model construction techniques to build a CTBN describing a vehicle system. The vehicle model makes use of some of the newly introduced algorithms and techniques, including the CTDN framework and disjunctive interaction. This extended application not only demonstrates the utility of the novel contributions presented in this work, but also serves as a template for applying CTBNs to other real-world problems.

CHAPTER ONE

INTRODUCTION

In a technology-driven society, there are many complex tasks that must be performed routinely to support daily operations. Consider a fleet of multi-purpose all-terrain vehicles designed to serve a military operation. The vehicles may perform a variety of tasks, including reconnaissance, transportation of supplies, and redistribution of personnel. These services are crucial for proper logistical support, and in some cases may even be necessary for mission success. Given the importance of these vehicles, it is imperative that at least a portion of the fleet remain operational. In pursuit of this goal, it is first desirable to understand the behavior of the components that make up the vehicles, and how they interact. These interactions can be very complex, and to make matters more difficult, the state of the components may change over time. Although reasoning about a system of dynamic, interdependent components is a challenging task, it is one worth pursuing.

The need to understand and reason about complex and changing systems extends to a variety of domains. Long-distance travel often relies on highly complex aircraft, and even automobiles are becoming more advanced in their capabilities. Military tools and weapon systems are evolving constantly to improve safety. Lab equipment for medical or production environments is also improved continually to reflect the current state of the art. As the complexity of systems and our dependence on them increases, ensuring their reliability becomes critical.

One approach to reasoning about complex systems is to employ a modeling technique. When working with systems that change over time, temporal models are used to capture the dynamic aspect of the components. Although there exist many types of temporal models, some are better suited for specific tasks than others. The correct choice of model often depends on whether or not the states of the system are discrete or continuous. For instance, the bendix in the starter of a vehicle is typically in either a working or failed state, and modeling the component as a continuous variable would not make much sense. Additionally, temporal models handle the notion of time differently. Time may be represented either implicitly or explicitly and can be treated as a discrete or continuous variable. In this dissertation, we focus on the subset of models where time is represented continuously, and states are represented discretely.

One temporal model that meets the discrete-state continuous-time criteria is the Continuous Time Markov Process (CTMP) [87]. This model works by describing the initial probability distribution over a set of discrete states, along with transition rates that describe how the process moves between these states as a function of time. CTMPs are representationally powerful models that have been applied successfully to the task of temporal modeling. Unfortunately, these models do not scale well. The that space of the process increases exponentially with the number of variables that are being modeled, making it difficult or impossible to describe large systems.

To manage the issue of exponential complexity, a CTMP can be factored by taking advantage of structural independencies. One such factored representation is Kronecker Algebra, which represents the transition rates for a CTMP compactly as a sum of Kronecker products [11]. Another possibility is to use a decision diagram, which is a model capable of encoding generic functions using a directed graph, and can be used to encode a CTMP by treating each potential transition rate as the domain of the function [43]. Although these are both valid compact representations, there is relatively little research available in the literature describing how to reason efficiently over these models. Instead, our focus lies in the Continuous Time Bayesian Network (CTBN), which factors a CTMP compactly using a directed graph structure [84]. The nodes in the graph represent variables, each of which are parameterized individually to describe transition behavior. CTBNs have received more attention in the literature, and a number of efficient inference algorithms and useful extensions have been developed as a result.

1.1 Motivation

The potential of many technological advancements remained initially unappreciated for one reason or another. One historically recurring issue is a simple failure to communicate and disseminate. The mere development of new technology is not sufficient to ensure its adoption; the development must be made known to the public before it is to achieve acclaim. Fortunately, the introduction of printed text, technical conferences, and web technologies have greatly reduced the difficulties associated with relaying new discoveries. Assuming proper notoriety in the literature, other limiters with respect to adoption of a theoretical technology often involve a shortcoming that prevents practical usage. This certainly holds true for many algorithmic advancements in computer science, especially in the field of machine learning, which often targets problems traditionally considered computationally intractable.

As an example, consider the artificial neural network. The concept of neural networks has existed since the early days of computing but was viewed as a theoretical novelty for the first few decades. In the late 1980s, the computational limitations of traditional computing techniques were exposed as new problems were attempted, and focus shifted back to neural networks as a potential solution [5]. Substantial funding was offered for novel work in the area, largely in the form of a proposal request from the Defense Advanced Research Projects Agency (DARPA) [69]. As a result, a variety of training algorithms were developed to aid in creating neural networks. Of particular note is the backpropagation algorithm, which was discovered and rediscovered several times [90, 143], until finally being popularized by Rumelhart, Hinton and Willaims in the mid-1980s [112]. This training algorithm provided a tractable means of training neural networks, which paved the way for many practical applications, especially over the next ten years [144, 145]. The rediscovery of the backpropagation algorithm demonstrates the need for the continued exchange of ideas. Furthermore, the delay in neural network adoption outside of a theoretical context shows that progress can be impeded when gaps in the literature exist that make for excessive complexity or ineffectiveness of an algorithm.

The convolutional neural network is another area of study in machine learning whose progress was forestalled after its initial proposal [68]. More recently however, a breakthrough was made in the literature, and convolutional neural networks were shown to substantially outperform prior state-of-the-art classification accuracies [64]. In this case, the breakthrough can be attributed to a more efficient implementation that distributed training computations over multiple GPUs. Prior to this type of implementation, the complexity of convolutional neural networks was a limiting factor and prevented wide-spread adoption of the model. Using parallel computing, convolutional neural networks are now the current state of the art for image classification tasks, and have been widely adopted by organizations like Google. Since the burden of complexity has been alleviated and practical usage has been demonstrated, a substantial number of works have been published in an effort to further improve convolutional neural networks [47, 48, 122]. This observed publishing behavior shows that not only can research enable practical application, but a demonstration of a real-world application can also spur additional research as well.

The CTBN is a probabilistic temporal model capable of representing complex systems. Inference algorithms have been developed that can answer sophisticated queries within a relatively short period of time [25, 38, 83]. Additionally, a variety of extensions have been proposed that improve or supplement representational capabilities, query retrieval, and evidence types [82, 127, 132, 136]. Despite this, there have been relatively few large-scale, real-world applications of CTBNs documented in the literature since its introduction in 2002. Why then does interest in CTBNs remain so largely theoretical?

One reason may be that despite the compact representation, CTBN complexity can still be unmanageable when applied to large systems. Although CTBNs achieve their compact representation by parameterizing each variable separately, parameterization of even a single variable can be a highly complex task. The issue is that the number of parameters required to specify a variable X is exponential in the number of variables that X directly depends on. In the worst case, a variable might depend on all other variables, resulting in complexity that is equivalent to an unstructured Markov process for the system. In general, for nodes with a large number of dependencies, the factored representation is less effective, and complexity of the model may make storing and reasoning over the model intractable.

Another potential inhibitor to the wide-spread adoption of CTBNs is restrictive representational capabilities. CTBNs naturally represent transition events as exponentially distributed random variables. While this may be sufficient for some applications, many real-world systems exhibit transitions that follow non-exponential distributions. The literature provides extensions to the CTBN framework that allow more complex distributions to be learned from data, but little has been done to show

how one might represent other parametric distributions and incorporate them into an existing CTBN.

The motivating factors that drive the work presented in this dissertation are to address the issues of scalability and representational power. The intent is to provide more compact CTBN representations that can be used when working with large, inter-dependent, real-world systems that may follow non-exponential parametric distributions. We hope that these contributions, along with a demonstration of how they might aid in modeling complex systems, will help to promote more wide-spread adoption of CTBNs for use in practical applications.

1.2 Contributions

Each contribution of this dissertation marks an advancement toward either the representational capabilities of CTBNs or the scalability of the model representation. These advancements share the common goal to support CTBN adoption in various domains. The most important contributions are listed below with citations to any of our existing published work in the area.

- We formalize the embedding process that inserts phase-type distributions into a CTBN model [99]. This provides the community with a mathematically rigorous procedure for representing non-exponential distributions within the CTBN framework.

- We present a method for learning phase-type distributions that closely approximate known non-exponential parametric distributions and empirically evaluate the effect of several factors on the approximation quality [97]. We then argue for a semantic interpretation of CTBN transition parameters that best describes the process in the presence of phase-type embedding. The result is a well-defined process for manually parameterizing and understanding CTBNs with non-exponential transition distributions.

- We introduce the continuous time decision network (CTDN) as a framework for optimal decision making in CTBNs [131]. Optimization is performed over a function that maps a set of discrete inputs to a value obtained by running inference over the model. The entire function, including all inputs, is represented directly using standard CTBN semantics.

- We derive distance metrics for quantifying the similarity between variable transition behavior. These metrics are used as the fundamental operators in a hierarchical clustering algorithm that identifies which parameters are exactly or approximately equivalent to one another for the purpose of producing compact representations.

- We develop a compact CTBN representation scheme based on discrete functions or compositions of discrete functions that map a large parameter space into a smaller space. We then show how these function mappings can be achieved using standard CTBN semantics.

- We develop a compact CTBN representation scheme based on decision trees that is able to capture context independence. We then show how the tree representation can be converted to use standard CTBN parameterization.

- We introduce the concept of disjunctive interaction for CTBNs, both for the binary and general case [96, 101]. This allows for a compact CTBN representation in cases where a set of variables act disjunctively on a dependent variable.

- We demonstrate how these contributions may be used when applying CTBNs to real-world problems and introduce several techniques for developing models that represent physical systems for the purpose of prognostics and health management (PHM) [98, 100, 102].

1.3 Organization

This section describes the organization of the remainder of this dissertation and provides a brief description of the content in each chapter.

Chapter 2 provides the background information necessary to understand the work presented in this dissertation. We begin by discussing Bayesian Networks and Dynamic Bayesian Networks, which are related probabilistic graphical models. We then describe the Continuous Time Markov Process, which is the model upon which CTBNs are founded. Next we provide an overview of CTBNs: the model used throughout the research presented in this dissertation.

Chapter 3 starts by providing background information on parametric distributions that are used frequently to describe transitions in temporal systems. We then discuss phase-type distributions, which have previously been used to approximate non-exponential distributions in CTBNs. We provide a formal description of how to embed these phase-type distributions into the model and demonstrate how non-parametric distributions can be learned. This allows non-exponential parametric distributions to be modeled within the CTBN framework.

Continuous Time Decision Networks and optimization are presented in Chapter 4. This builds on work done in the area of CTBN performance functions. The notion of decision nodes is introduced, and a demonstration of how to use these nodes for optimization is presented.

Chapter 5 starts by deriving distance measures and a clustering algorithm for the parameters in a CTBN. The chapter continues by introducing two compact representation schemes. The first is based on discrete functions or convolutions of discrete functions that map parent state combinations to a smaller domain. Next, a compact representation scheme based on decision trees is presented, providing an alternative to the discrete function representation. The chapter concludes by showing that there are CTBN parameterizations that can be represented fully using either discrete functions, decision trees, both, or neither.

In Chapter 6, we introduce disjunctive interaction for CTBNs. This allows for a potentially significant reduction in the number of parameters that need to be specified for a given CTBN. We then show how existing inference algorithms can be adapted to use disjunctive interaction by computing transition rates on the fly. Disjunctive interaction simplifies the modeling process and can reduce the space-complexity of the model, as well as the expected runtimes for the corresponding inference procedures, which may enable the representation of more complex systems.

Chapter 7 describes how CTBNs can be used in the context of PHM. We start by providing background on diagnostic models referred to as D-Matrices. We then show how a D-Matrix can be used to derive the structure of a prognostic CTBN automatically. We then show how the resulting CTBN can be parameterized using failure and repair rates, along with false alarm and non-detect values. Next, background is provided on fault trees, and it is shown how to derive CTBNs automatically from these models. The process for combining the two types of derived CTBN models is then described. Finally, an example CTBN is introduced that makes use of the described derivation algorithms, along with the other features discussed throughout this dissertation.

Chapter 8 concludes the work by summarizing previous chapters and the contributions made by each. This chapter continues by discussing how these contributions impact the existing body of literature for CTBNs and how this might improve work in CTBNs going forward. The chapter and this dissertation ends with a discussion on future work that we hope to pursue in the near future.

CHAPTER TWO

BACKGROUND

This chapter provides the background necessary to set the context for our contributions. We start by describing Bayesian Networks and Dynamic Bayesian Networks, which are probabilistic graphical models that share many features with CTBNs. Next we discuss Continuous Time Markov Processes, which form the mathematical foundation for CTBNs. Finally we provide background on CTBNs, the model on which we base our work throughout the remainder of this dissertation.

2.1 Bayesian Networks

The probability distribution over a set of random variables $\mathbf{X}$ is referred to as the joint probability distribution. Although this distribution is rich with information, its size is exponential in the number of variables, making it infeasible to work with the information for most practical applications. The Bayesian Network (BN) is a probabilistic graphical model capable of describing the joint probability distribution over $\mathbf{X}$ compactly [61]. This compact representation is achieved by taking advantage of conditional independence between variables in $\mathbf{X}$. Relationships between variables are encoded using a graph structure $\mathcal{G}$, such that the nodes correspond to variables, while directed edges indicate that a node is conditioned on its parents in the graph. To maintain a valid probability distribution, $\mathcal{G}$ is constrained to be acyclic. To formalize, the notion of independence in random variables must first be provided.

Definition 2.1.1 (Independence)**.** *Two random variables* X_1 *and* X_2 *are* independent *if and only if*

$$P(X_1, X_2) = P(X_1)P(X_2).$$

That is, the joint probability of two independent variables is equal to the product of the probability of the marginals. When two variables are independent of one another, then obtaining evidence for one of the variables will not change the distribution for the other variable. This can be seen by expanding the conditional probability formula, and applying Definition 2.1.1:

$$P(X_1|X_2) = \frac{P(X_1, X_2)}{P(X_2)} = \frac{P(X_1)P(X_2)}{P(X_2)} = P(X_1).$$

Although this would greatly simplify the process of identifying probability distributions in a complex system of variables, it is unlikely that independence between variables will occur. Instead, it is more reasonable to expect a system to exhibit conditional independence, defined as follows.

Definition 2.1.2 (Conditional Independence)**.** *Two random variables* X_1 *and* X_2 *are* conditionally independent *given a set of random variables* $\mathbf{Y}$ *if and only if*

$$P(X_1, X_2|\mathbf{Y}) = P(X_1|\mathbf{Y})P(X_2|\mathbf{Y}).$$

Using the same technique as before, we find that $P(X_1, X_2|\mathbf{Y}) = P(X_1|\mathbf{Y})$. Intuitively, two variables are conditionally independent if they exhibit independence when conditioned on evidence applied to another set of variables. This means that although X_1 may affect X_2 indirectly, this affect is mediated by the set of variables $\mathbf{Y}$. Conditional independence is the underlying concept that enables the BN to factor the joint probability distribution over a set of variables.

Definition 2.1.3 (Bayesian Network)**.** *A* Bayesian Network B *is a factored representation of a joint probability distribution over a set of random variables* $\mathbf{X} = \{X_1, X_2, \cdots, X_n\}$*. The model consists of two parts: a graph structure* $\mathcal{G}$ *and a set of parameters* P*. Graph* $\mathcal{G}$ *is a directed acyclic graph with nodes corresponding to the variables in* $\mathbf{X}$*. Parameterization* P *is a set of conditional probability distributions (CPDs) for each* $X_i \in \mathbf{X}$*, providing the conditional distribution* $P(X_i|\mathbf{Pa}(X_i))$*, where* $\mathbf{Pa}(X_i)$ *is the set of parents for node* X_i *in graph* $\mathcal{G}$*.*

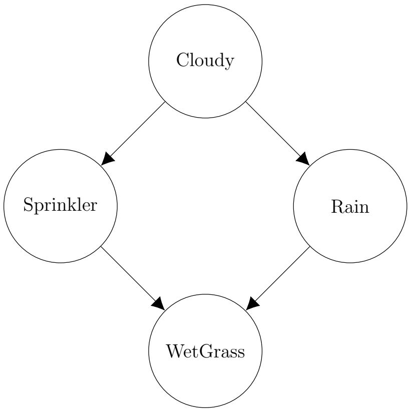
The parameters P in a BN are sufficient to describe the joint probability distribution over the set of variables $\mathbf{X}$. The full joint probability distribution can be obtained by combining the conditional probability distributions using the chain rule:

$$P(\mathbf{X}) = \prod_{i=1}^{n} P(X_i|\mathbf{Pa}(X_i)). \tag{2.1}$$

In other words, the BN factors the joint probability distribution into a product of smaller, more manageable distributions over each variable and its parents in the graph. This compact representation is able to represent the same information as its unstructured counterpart by making use of the graph structure $\mathcal{G}$, which encodes the independencies in the original probability distribution. To demonstrate the BN factorization, consider the following example borrowed from Pearl [92].

Example 2.1.1. *Sprinkler Bayesian Network*

Consider a system of four discrete variables describing the interaction of a sprinkler and the surrounding environment: $\mathbf{X} = \{\text{Cloudy}, \text{Sprinkler}, \text{Rain}, \text{WetGrass}\}$*, where each variable has two states True (T) and False (F). Variables indicate whether or not the sky is cloudy, the sprinkler is on, it is raining, or the grass is currently wet respectively. Variables* Sprinkler *and* Rain *depend directly on* Cloudy*, while variable*



	Cloudy	T	F
Sprinkler	T	0.1	0.5
	F	0.9	0.5

	Cloudy	T	F
Rain	T	0.8	0.2
	F	0.2	0.8

Cloudy	T	0.5
	F	0.5

	Sprinkler	T		F	
	Rain	T	F	T	F
WetGrass	T	0.99	0.9	0.9	0.0
	F	0.01	0.1	0.1	1.0

Figure 2.1: Bayesian network sprinkler example.

WetGrass *depends directly on both* Sprinkler *and* Rain*. The graph structure* $\mathcal{G}$ *and conditional probability distributions* P *are given by Figure 2.1.*

The joint probability distribution could be specified for the sprinkler model by assigning a probability to each unique combination of state instantiations of the four variables, resulting in $2^4 = 16$ parameters. While this may be possible for small models, the exponential increase in parameters makes this unstructured representation intractable for all but the most trivial problems. A BN provides a compact representation by factoring the full joint distribution using the chain rule from Equation 2.1. For the sprinkler example, the full joint distribution is factored as follows, where each variable is represented using its first letter:

$$P(C, S, R, W) = P(C)P(S|C)P(R|C)P(W|S, R).$$

This requires only $2^0 + 2^1 + 2^1 + 2^2 = 9$ free parameters, one for each column in the conditional probability tables from Figure 2.1. Note that there is actually a total of 18 parameters in the sprinkler example, but only half of them are free parameters, with the other half being determined implicitly by the discrete probability distribution.

Some probabilities can be retrieved directly from the BN, such as the likelihood that the grass is wet given that the sprinkler is on and it is not raining: $P(W = T|S = T, R = F) = 0.9$. Other probabilities, such as $P(C = T, S = T|W = F)$, the probability that it is cloudy and the sprinkler is on given that the grass is not wet, are not encoded directly by the network but can be obtained by performing inference over the network.

Despite being a representationally powerful model, the BN is only capable of representing static distributions. When working with systems that change through time, a temporal model must be employed to reason about distributions through time. The remaining sections in this chapter discuss models capable of representing a system's state distribution as a function of time.

2.2 Dynamic Bayesian Networks

The Dynamic Bayesian Network (DBN) is an extension to the BN discussed in the previous section that allows for the representation of a system that changes over time [27, 28]. Essentially, a DBN is a collection of BNs that each represent the state of a system at a different point in time. To encode the expected dynamics of the system, edges are added from the nodes in one BN to the corresponding nodes in a BN representing some time in the future. These edges ensure that the state of a variable depends directly on the state of that variable at a previous point in time.

Definition 2.2.1 (Dynamic Bayesian Network). *Let $\mathbf{X}_i$ be a set of random variables at some time $t = i$, and let $\mathbf{Z} = \{\mathbf{X}_0, \mathbf{X}_1, \mathbf{X}_2, \ldots\}$ be an infinite series of these sets at fixed time intervals. Then a* Dynamic Bayesian Network *is a specialized Bayesian Network that provides a factored representation of a joint probability distribution over each set of random variables $\mathbf{X}_i \in \mathbf{Z}$. The model consists of two parts, a Bayesian*

Network B_0 over variables $\mathbf{X}_0$, and a special Bayesian Network B_k that specifies a network over some set of variables $\mathbf{X}_k$ conditioned on variables in prior networks $\mathbf{X}_j$, where $j < k$. The network B_k may have intra-time-slice arcs that connect variables within $\mathbf{X}_k$, or inter-time-slice arcs that connect variables from some $\mathbf{X}_j$ to $\mathbf{X}_k$.

Note that in general, the nodes in a DBN can depend on any previous timestep. For instance, a variable from timestep k may depend on a variable from timestep $k-3$. As with any recursively defined series, this implies that the first three BNs must be specified before the recursive definition can be employed, or else assumptions must be made about the absence of these dependencies. Furthermore, modeling dependencies between two timesteps that are relatively far away can be computationally expensive, and often unnecessary from a representational standpoint. As a result, DBNs often impose the restriction that a BN B_k can only depend on nodes in the previous timestep B_{k-1}. This is referred to as a first-order DBN, which factors the joint probability distribution as follows:

$$P(\mathbf{X}_0, \ldots, \mathbf{X}_n) = P(\mathbf{X}_0) \prod_{t=0}^{n} P(\mathbf{X}_{t+1} | \mathbf{X}_t).$$

Standard inference algorithms designed for BNs can be tailored to work with DBNs as well. The resulting model is capable of answering queries about the state distribution at any point in time that is explicitly represented by one of the contained BNs. Going forward, the notation $P(X(t))$ is used to indicate the probability distribution over variable X at time t. For instance, it may be necessary to obtain the probability distribution over node X at some time $t = 100$, or $P(X(100))$. Inference can achieve this by first performing inference over the initial variables $\mathbf{X}_0$, and then using this information to perform inference over $\mathbf{X}_1$. This process, referred to as “unrolling”, is repeated 100 times until timestep $\mathbf{X}_{100}$ is reached. It is in this way

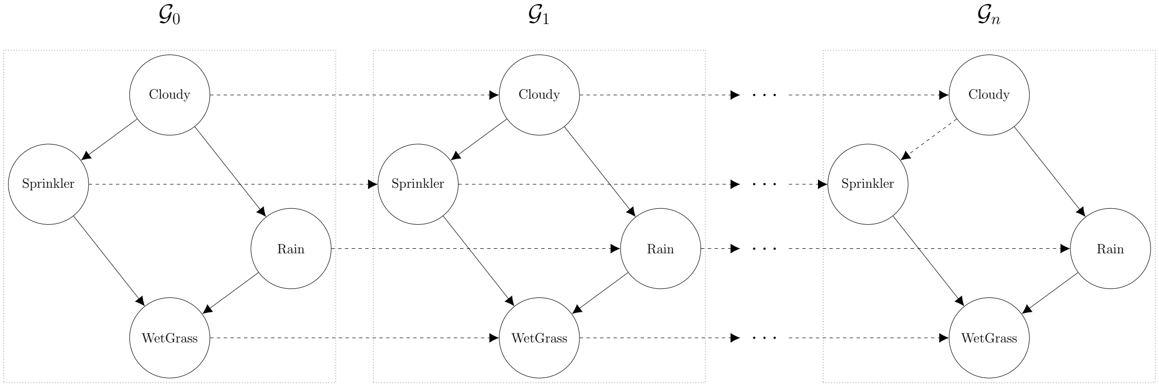


Figure 2.2: First-order dynamic Bayesian network sprinkler example.

that the DBN is able to work with an infinite sequence of nodes, in that nodes are only considered as they are necessary for inference.

Example 2.2.1. *Sprinkler Dynamic Bayesian Network*

Consider a system of four discrete variables describing the interaction of a sprinkler and the surrounding environment: $\mathbf{X} = \{\text{Cloudy}, \text{Sprinkler}, \text{Rain}, \text{WetGrass}\}$*, where each variable has two states True (T) and False (F). Variables indicate whether or not the sky is cloudy, the sprinkler is on, it is raining, or the grass is currently wet respectively. Variables* Sprinkler *and* Rain *depend directly on* Cloudy*, while variable* WetGrass *depends directly on both* Sprinkler *and* Rain*. Furthermore, the state of these variables change in discrete time, such that each variable also depends on its own previous state. The graph structure for the corresponding DBN is shown in Figure 2.2.*

DBNs have been applied to a variety of domains that require temporal reasoning. In prognostics, failure and repair rates are encoded by the model, providing a means to predict the likelihood of failure at some time in the future [12,33,71]. These failure probabilities can then be used to perform proactive maintenance [72,78]. DBNs have also been applied to variety of other fields, including bioinformatics [32,154], medical prognostics [36,89], and computer vision [30,91].

Despite their demonstrated success, there are limitations that make the DBNs impractical for some applications. By nature, DBNs implicitly represent time as a discrete set of timesteps. As such, queries cannot be performed at times “between” these timesteps. Furthermore, unrolling can be an expensive procedure when the desired timestep t is high. For cases where events do not occur at uniform intervals, choosing a time granularity can be a difficult tradeoff between complexity and representational accuracy. In cases like these, it makes more sense to represent time directly as a continuous random variable. The remaining two sections in this chapter discuss continuous time models that avoid the discrete time assumption made by DBNs.

2.3 Markov Processes

An alternative to DBNs that allows for a continuous representation of time is the Continuous Time Markov Process (CTMP) [6]. This model describes the distribution over a continuous time random process, which is a set of random variables $\mathbf{X}$ defined as a function of time. Roughly speaking, $\mathbf{X}$ can be thought of as a discrete state variable in a BN, where each state is now a random variable dependent on time rather than a fixed value.

Definition 2.3.1 (Continuous Time Markov Process). *Let X be a continuous time random process, consisting of a set of variables $\mathbf{X}$ that change as a function of continuous time. A CTMP is a model over X consisting of two parts: an initial distribution $P_X(0)$ and a transition intensity matrix $\mathbf{Q}_X$ defined over the states of X. Each entry $q_{i,j}$ in row i, column j of the matrix $\mathbf{Q}_X$ defines the non-negative intensity* *with which the process will transition from state x_i to state x_j as a function of time. The diagonal entry for some row i and column i is denoted $q_{i,i}$ or simply q_i, and is constrained to be the negative sum of the rest of the row. Formally, $q_{i,i} = -\sum_{j \neq i} q_{i,j}$.*

Note that the constraint on the diagonals ensures that each row sums to 0.0, which is an attribute of the matrix that facilities inference options. If a process X has n states, then the initial distribution P is represented using a normalized n-length vector of probabilities, while the transition intensity matrix is represented using an $n \times n$ matrix of intensities. Such a CTMP is shown below.

$$P_X(0) = \begin{bmatrix} p_1 & p_2 & \cdots & p_n \end{bmatrix}$$

$$\mathbf{Q}_X = \begin{array}{c|cccc} & x_1 & x_2 & \cdots & x_n \\ \hline x_1 & q_{1,1}(t) & q_{1,2}(t) & \cdots & q_{1,n}(t) \\ x_2 & q_{2,1}(t) & q_{2,2}(t) & \cdots & q_{2,n}(t) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_n & q_{n,1}(t) & q_{n,2}(t) & \cdots & q_{n,n}(t) \end{array}$$

The process is expected to transition from some state x_i to some state x_j with an intensity of $q_{i,j}(t)$. This indicates that the transition rate may change as a function of time, meaning that the process may transition with a higher or lower intensity at a different points in time. There is a more restrictive class of CTMPs that make the assumption that intensities do not depend on time, and are instead constant. More formally, $q_{i,j}(0) = q_{i,j}(t)$ for all $t \in [0, \infty)$. This is referred to as a homogeneous Markov process, and for the remainder of this dissertation, any reference to a CTMP is assumed to be homogeneous. Conversely, the rates in an inhomogeneous intensity matrix change as a function of time. The matrix $\mathbf{Q}_X$ can now be rewritten in terms of fixed intensities, which may also be referred to as rates.

$$\mathbf{Q}_X = \begin{array}{c} \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & \cdots & x_n \end{array} \\ \begin{pmatrix} q_{1,1} & q_{1,2} & \cdots & q_{1,n} \\ q_{2,1} & q_{2,2} & \cdots & q_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n,1} & q_{n,2} & \cdots & q_{n,n} \end{pmatrix} \end{array}$$

The transition behavior of process X can be described using exponential distributions parameterized using the intensities from matrix $\mathbf{Q}_X$. Let X start in a state of x_i at time $t = 0$. The amount of time that X stays in state x_i before transitioning to another state is exponentially distributed with parameter $q_{i,i}(t)$. The probability density function (PDF) $f_{q_{i,i}}$ and the cumulative distribution function (CDF) $F_{q_{i,i}}$ are given below. These functions provide the likelihood that the process X remains in the same state x_i from time $t = (0, t]$.

$$\begin{aligned} f_{q_{i,i}}(t) &= q_{i,i} \exp(-q_{i,i}t), && t \geq 0 \\ F_{q_{i,i}}(t) &= \int f_{q_{i,i}}(t)dt = 1 - \exp(-q_{i,i}t), && t \geq 0 \end{aligned}$$

Example 2.3.1. *Barometric Pressure CTMP*

Let B be a continuous time random process for barometric pressure. A CTMP may be used to model this process using three states $\{$ b_0 = falling, b_1 = steady, b_2 = rising $\}$. Using hours as the time unit, such a CTMP is given as follows.

$$P_B = \begin{bmatrix} 0.0 & 0.2 & 0.8 \end{bmatrix}$$

$$\mathbf{Q}_B = \begin{pmatrix} -0.21 & 0.20 & 0.01 \\ 0.05 & -0.10 & 0.05 \\ 0.01 & 0.20 & -0.21 \end{pmatrix}$$

The CTMP in the barometric pressure example provides all the information necessary to model the state of the process through time. Note that the process is expected to start either in state b_2 (rising), or possibly b_1 (steady). If the process is in state b_2 (rising) at time $t = 0$, then the entry $q_{2,2} \in \mathbf{Q}_B$ indicates that the process will stop rising with an intensity of 0.21. It is expected that the process will stop rising in $t = 1/0.21 \approx 4.76$ hours. Using the CDF, the probability of transitioning to another state by time $t = 3.5$ hours in the future is $F_{0.21}(3.5) = 1 - \exp(-0.21 \cdot 3.5) \approx 0.52$. Upon leaving the rising state, the process will enter the steady state with a probability of $0.2/0.21 \approx 0.95$ or to the falling state with probability $0.01/0.21 \approx 0.05$.

One factor that has a strong influence on the behavior of a process is whether or not there exist absorbing states.

Definition 2.3.2 (Absorbing State). *A state x_i in a CTMP X is said to be* absorbing *if and only if $\forall j(q_{i,j} = 0)$.*

In other words, if the transition intensity to all other states is zero, then the state is said to be absorbing. Note that if all of the intensities in a row of an intensity matrix are zero, then the diagonal is constrained to a value of zero as well. Once a process enters an absorbing state, it will remain there permanently. Let x_i be an absorbing state in process X. If $X(s) = x_i$, then $X(s + t) = x_i$ for all $t \in [0, \infty)$. If a state is not absorbing, then it is considered transient. Absorbing states strongly influence the steady-state distribution, and properties of absorbing and transient states are leveraged during our work with phase-type distributions in Chapter 3.

It is worth mentioning that there are two equivalent approaches to parameterizing the transition behavior for a process. The first is referred to as a *pure intensity* parameterization and specifies transition intensities using a single intensity matrix $\mathbf{Q}$. The barometric pressure example uses a pure intensity parameterization. This type of parameterization lends itself to a "racing exponentials" interpretation, which views each of the off-diagonal entries in the row of an intensity matrix as the rates for separate exponential distributions. For instance, consider the barometric pressure system where the process starts in state b_0. The next state can be determined by "racing" two exponential distributions with rates $q_{0,1} = 0.2$ and $q_{0,2} = 0.01$, which can be accomplished by sampling from each distribution and choosing the transition time that comes first.

A second representation of the parameters is referred to as the *mixed intensity* parameterization. In this representation, a homogeneous CTMP X is represented using two sets of parameters $\mathbf{q}$ and $\boldsymbol{\theta}$. Each $q_i \in \mathbf{q}$ is a rate value used to parameterize the exponential distribution associated with the next transition time for state x_i, while each $\theta_i \in \boldsymbol{\theta}$ is a discrete probability distribution over the remaining states $x_{j \neq i}$. In other words, $\mathbf{q}$ provides the information about *when* the process transitions, while $\boldsymbol{\theta}$ indicates *where*. This type of parameterization often uses a sojourn time interpretation, which first identifies the amount of time spent in a state x_i, and then identifies the location of the transition using the discrete probability distribution. For barometric pressure starting in state b_0, the transition time would first be obtained by sampling the sojourn time from an exponential distribution with a rate of $q_{0,0} = 0.21$. The next state would then be determined by drawing from a discrete distribution over the remaining states: $\theta_0 = [0.2/0.21 \ \ 0.01/0.21]$.

Note that the *pure intensity* representation and the *mixed intensity* representation are functionally equivalent methods for parameterizing homogeneous CTMPs.

The relationship between the two representations can be inferred by looking at the alternate transition models for the barometer example. Specifically, each mixed intensity parameter $q_i \in \mathbf{q}$ corresponds to the negation of the diagonal entry $q_{i,i} \in \mathbf{Q}$ in the pure intensity parameterization. Furthermore, θ_i in the mixed intensity parameterization is a vector corresponding to the off-diagonal entries of row i in $\mathbf{Q}$, normalized to transform the row into a unit vector. In other words, the parameters for $\boldsymbol{\theta}$ are computed as the ratio $q_{i,j}/q_{i,i}$, for all $j \neq i$. This guarantees that each θ_i is a valid probability distribution over the remaining states. Although these representations are functionally equivalent, the interpretations are distinct. Chapter 3 discusses the mixed intensity representation in greater detail, and advocates for its use in scenarios involving non-exponential distributions. For notational convenience however, the majority of this dissertation presents models using the pure intensity representation.

Using the distributions defined by the rates in the intensity matrix, a CTMP is able to represent the dynamics of a discrete system as a function of continuous time. This representation has significant advantages over a discrete space model when there is no natural choice of discretization available. A CTMP encodes the distribution over a process X as a function of time, which here is treated as a continuous random variable.

$$P_X(t) = P_X(0)\exp(\mathbf{Q}_X t).$$

In this equation, the initial distribution $P_X(0)$ and intensity matrix $\mathbf{Q}_X$ are defined as part of the CTMP specification. The $\exp(\cdot)$ operation is known as the matrix exponential, which is defined as

$$\exp(\mathbf{Q}) = \sum_{k=0}^{\infty} \frac{\mathbf{Q}^k}{k!},$$

although computation of the matrix exponential is typically achieved by employing efficient approximations [77]. Using this operation, inference can be performed to identify the probability distribution over a process at any continuous point in time t. As t increases, the distribution over the process approaches its steady state distribution, which can be computed using eigenvalue analysis [66].

Unlike BNs or DBNs, each state in a CTMP X is a single random variable indexed by time. How then can such a model be used to describe a collection of discrete variables $\mathbf{Y} = \{Y_1, Y_2, \ldots, Y_n\}$? To achieve this, each state of the process X is used to describe a complete instantiation to the variables in $\mathbf{Y}$. As such, the number of states in X is defined by the states in $\mathbf{Y}$:

$$|X| = \prod_{Y \in \mathbf{Y}} |Y|.$$

This representation allows multiple discrete variables to be modeled by the same CTMP.

Example 2.3.2. *Pain System CTMP*

A continuous time random process consists of a subsytem for barametric pressure $B = \{b_0, b_1, b_2\}$, a subsystem for the pain experienced by a medical patient $P = \{p_0, p_1\}$, and a subsystem for the concentration of a drug in the patient's system $C = \{c_0, c_1\}$. Using hours as the time unit, a CTMP describing the process is given as follows.

$$P(0) = \begin{bmatrix} p_0 & p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & p_7 & p_8 & p_9 & p_{10} & p_{11} \end{bmatrix}$$

$$\begin{array}{r|cccccccccccc|}
 & b_0p_0c_0 & b_0p_0c_1 & b_0p_1c_0 & b_0p_1c_1 & b_1p_0c_0 & b_1p_0c_1 & b_1p_1c_0 & b_1p_1c_1 & b_2p_0c_0 & b_2p_0c_1 & b_2p_1c_0 & b_2p_1c_1 \\
b_0p_0c_0 & -1.71 & 0.20 & 0.10 & 0.05 & 0.25 & 0.20 & 0.10 & 0.00 & 0.00 & 0.01 & 0.30 & 0.50 \\
b_0p_0c_0 & 0.15 & -1.86 & 0.20 & 0.10 & 0.10 & 0.20 & 0.50 & 0.01 & 0.20 & 0.30 & 0.10 & 0.00 \\
b_0p_0c_0 & 0.05 & 0.20 & -1.76 & 0.10 & 0.01 & 0.50 & 0.10 & 0.00 & 0.00 & 0.10 & 0.20 & 0.50 \\
b_0p_0c_0 & 0.01 & 0.10 & 0.05 & -1.56 & 0.25 & 0.20 & 0.05 & 0.20 & 0.25 & 0.00 & 0.20 & 0.25 \\
b_0p_0c_0 & 0.30 & 0.50 & 0.05 & 0.25 & -2.02 & 0.10 & 0.10 & 0.20 & 0.20 & 0.01 & 0.30 & 0.01 \\
\mathbf{Q} = b_0p_0c_0 & 0.25 & 0.50 & 0.00 & 0.25 & 0.05 & -2.20 & 0.20 & 0.00 & 0.25 & 0.50 & 0.20 & 0.00 \\
b_0p_0c_0 & 0.25 & 0.25 & 0.05 & 0.20 & 0.10 & 0.05 & -2.45 & 0.50 & 0.20 & 0.30 & 0.30 & 0.25 \\
b_0p_0c_0 & 0.20 & 0.00 & 0.20 & 0.20 & 0.01 & 0.10 & 0.20 & -1.52 & 0.20 & 0.01 & 0.10 & 0.30 \\
b_0p_0c_0 & 0.05 & 0.00 & 0.00 & 0.01 & 0.30 & 0.05 & 0.50 & 0.00 & -1.31 & 0.05 & 0.25 & 0.10 \\
b_0p_0c_0 & 0.05 & 0.00 & 0.00 & 0.30 & 0.25 & 0.50 & 0.25 & 0.20 & 0.01 & -1.91 & 0.05 & 0.30 \\
b_0p_0c_0 & 0.01 & 0.20 & 0.05 & 0.25 & 0.01 & 0.00 & 0.50 & 0.05 & 0.30 & 0.00 & -1.62 & 0.25 \\
b_0p_0c_0 & 0.05 & 0.30 & 0.00 & 0.20 & 0.25 & 0.10 & 0.50 & 0.00 & 0.10 & 0.30 & 0.20 & -2.00
\end{array}$$

Although the CTMP is a powerful model capable of representing discrete variables as a function of time, the size of the initial distribution and intensity matrix is exponential in the number of variables. Even in the pain system, which consists of only three variables with a small number of states, the number of parameters required to specify the CTMP is already substantially larger than the original barometer process. Unfortunately, this exponential complexity makes representing and reasoning over CTMPs intractable for all but the simplest of problems. For practical applications, a more efficient model is required.

2.4 Continuous Time Bayesian Networks

This section describes the continuous time Bayesian network (CTBN), which is the model studied throughout this dissertation. An overview is provided regarding representation and reasoning, as well as extensions and applications.

2.4.1 Representation and Interpretation

A CTBN is a probabilistic graphical model that factors a CTMP in much the same way as a BN factors a joint probability distribution [84]. Rather than modeling all variables in the set $\mathbf{X}$ with a single initial distribution and intensity matrix, the individual variables are factored using a graph structure. Before describing the CTBN in detail, the concept of a Conditional Markov Process must first be introduced.

Definition 2.4.1 (Conditional Markov Process)**.** *A* Conditional Markov Process *is a special type of CTMP where the transition intensities for a variable X vary as a function of the current state of the variables in a set $\mathbf{U}$. Unlike with traditional inhomogeneous CTMPs, the intensities in a Conditional Markov Process change* indirectly *as a function of time. The intensity matrix for a Conditional Markov Process is referred to as a* Conditional Intensity Matrix *(CIM).*

An example of a CIM may be written as:

$$\mathbf{Q}_{X|\mathbf{U}} = \begin{array}{c} \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & \cdots & x_n \end{array} \\ \begin{pmatrix} q_{1,1}(\mathbf{U}) & q_{1,2}(\mathbf{U}) & \cdots & q_{1,n}(\mathbf{U}) \\ q_{2,1}(\mathbf{U}) & q_{2,2}(\mathbf{U}) & \cdots & q_{2,n}(\mathbf{U}) \\ \vdots & \vdots & \ddots & \vdots \\ q_{n,1}(\mathbf{U}) & q_{n,2}(\mathbf{U}) & \cdots & q_{n,n}(\mathbf{U}) \end{pmatrix} \end{array},$$

where each function $q_{i,j}(\mathbf{U})$ is a discrete function mapping instantiations of $\mathbf{U}$ to fixed transition intensities. As a result of the discrete function, a CIM may be represented equivalently as a set of intensity matrices indexed by an assignment $\mathbf{u}$ to the conditioning set $\mathbf{U}$. If the conditioning set $\mathbf{U}$ is empty, then the process is not conditioned on anything, and a single intensity matrix is used to represent the standard (unconditional) process. Note that CIMs are analagous to conditional probability tables from BNs, where instead of table cells corresponding to likelihoods, they map to intensity matrices.

Using the Conditional Markov Process as a building block, it is now possible to formally define a CTBN.

Definition 2.4.2 (Continuous Time Bayesian Network)**.** *A* Continuous Time Bayesian Network $\mathcal{N}$ *is a factored representation of a CTMP over a set of discrete random variables* $\mathbf{X} = \{X_1, X_2, \cdots, X_n\}$*. The model consists of two parts: a graph structure* $\mathcal{G}$ *and a set of parameters* P*. Graph* $\mathcal{G}$ *is a directed, possibly cyclic graph with nodes corresponding to the variables in* $\mathbf{X}$*. Parameterization* P *is a set of Conditional Markov Processes, one for each* $X_i \in \mathbf{X}$*, conditioned on its parents in graph* $\mathcal{G}$*.*

The compact representation achieved by factoring a CTMP using the graph structure works to address the issue of exponential complexity. Rather than defining a single CTMP over an exponential number of states, a series of conditional CTMPs are defined for each node conditioned on its parents. For each variable X, the size of the initial distribution and the dimensions of the intensity matrices in the CIM are equal to the size of the domain of X. Depending on the dependency structure, this can be a significant improvement over a standard CTMP.

Just as with a BN, each individual node in a CTBN represents a variable, while the directed edges describe direct dependence among these variables. The only difference between the graph structure of a BN and that of a CTBN is that the acyclicity constraint is lifted when dealing with continuous time. This is because dependencies occur through time, meaning that cyclic dependencies can in fact occur. This is analogous to how “cycles” are permitted in DBNs by acyclically connecting nodes from previous timesteps. In either case, a variable cannot depend on its current state, only its state at previous times.

Example 2.4.1. *Drug Effect Continuous Time Bayesian Network [84]*

Consider a system of eight discrete variables describing the effect of a drug on a patient: $\mathbf{X} = \{$ ***`Concentration`**, **`Pain`**, **`Barometer`**, **`Drowsy`**, **`Uptake`**, **`Full Stomach`**, **`Hungry`**, **`Eating`*** $\}$*. At a high level, the concentration of a drug in a patient's system is directly determined by the uptake of the drug and the contents of the patients stomach. The drug itself influences the patient's pain and drowsiness. The content's of the patient's stomach is dependent on if they were eating, which is dependent on if they were hungry, which is dependent on if their stomach was full. Finally, a patient's pain level is not only influenced by the concentration of the drug, but also the barometric pressure. The graph structure* $\mathcal{G}$ *for a CTBN describing process* X *is shown in Figure 2.3.*

The drug effect example network provides a demonstration of how a CTBN might be used to describe an interaction of variables that change in continuous time. Consider the `Barometer` node, representing barometric pressure. This node has no parents; therefore, it behaves according to an unconditional CTMP, identical to the one presented in Example 2.3.1. The `Pain` node is more complex, in that it depends on both `Barometer` and `Concentration`. It is instead described by a Conditional Markov Process, as shown below. Note that the CIM $\mathbf{Q}_{P|B,C}$ is specified as a set of intensity matrices associated with each state instantiation.

$$P_P = \begin{bmatrix} 0.1 & 0.9 \end{bmatrix}$$

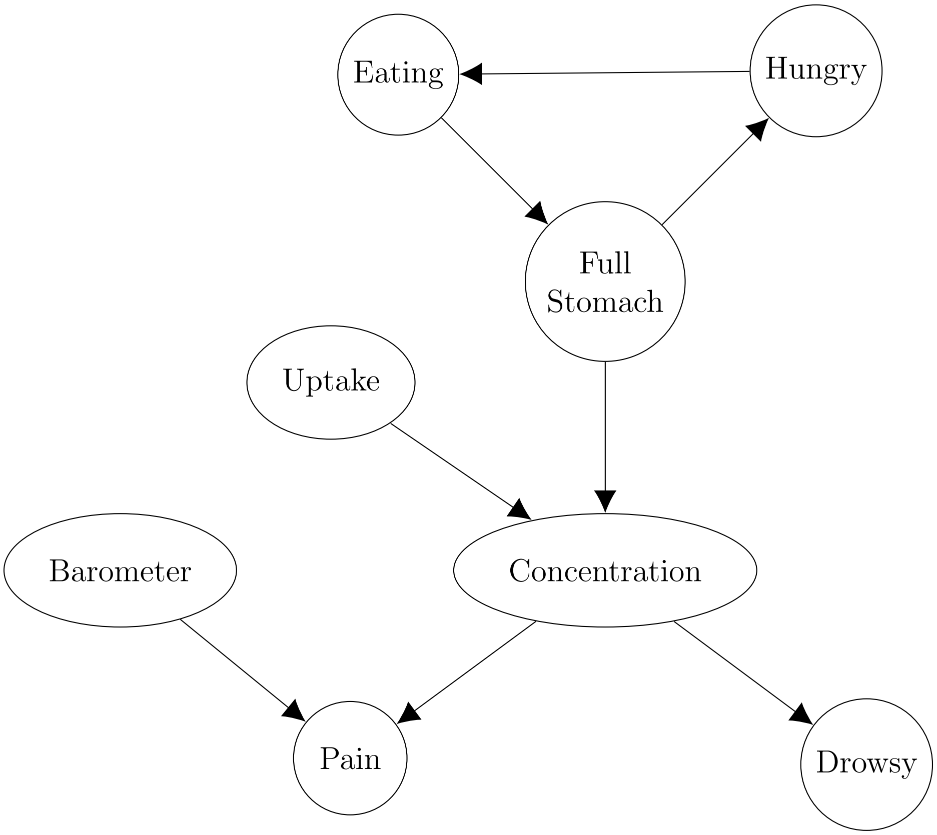


Figure 2.3: The drug effect example network describing the influence of a drug on a patient.

$$\mathbf{Q}_{P|B,C} =$$

$$\left\{\begin{array}{ccc}
\mathbf{Q}_{P|b_0,c_0} = \begin{pmatrix} -6 & 6 \\ 0.1 & -0.1 \end{pmatrix} & \mathbf{Q}_{P|b_0,c_1} = \begin{pmatrix} -1 & 1 \\ 0.1 & -0.1 \end{pmatrix} & \mathbf{Q}_{P|b_0,c_2} = \begin{pmatrix} -6 & 6 \\ 0.1 & -0.1 \end{pmatrix} \\
\mathbf{Q}_{P|b_1,c_0} = \begin{pmatrix} -1 & 1 \\ 0.3 & -0.3 \end{pmatrix} & \mathbf{Q}_{P|b_1,c_1} = \begin{pmatrix} -0.3 & 0.3 \\ 0.3 & -0.3 \end{pmatrix} & \mathbf{Q}_{P|b_1,c_2} = \begin{pmatrix} -1 & 1 \\ 0.3 & -0.3 \end{pmatrix} \\
\mathbf{Q}_{P|b_2,c_0} = \begin{pmatrix} -0.01 & 0.01 \\ 2 & -2 \end{pmatrix} & \mathbf{Q}_{P|b_2,c_1} = \begin{pmatrix} -0.01 & 0.01 \\ 2 & -2 \end{pmatrix} & \mathbf{Q}_{P|b_2,c_2} = \begin{pmatrix} -0.01 & 0.01 \\ 2 & -2 \end{pmatrix}
\end{array}\right\}$$

In this model, pain is discretized into two states, with b_0 and b_1 representing the absence and presence of pain respectively. As such, the initial distribution vector P_P has two entries, and the dimension of each of the intensity matrices is 2×2. The CIM $\mathbf{Q}_{P|B,C}$ consists of a total of nine distinct intensity matrices, each of which describes the transition behavior of the pain process in different situations. Specifically, one intensity matrix is provided for each state combination of B (`Barometer`) and C (`Concentration`). Both B and C have three states in their domain, resulting in the observed $3 \times 3 = 9$ intensity matrices in the CIM for P. This transition model indicates that when the barometric pressure is falling ($B = b_0$) and the concentration of the drug is low ($C = c_0$), then the patient will transition from a state of no pain (p_0) to a state of pain (p_1) with an intensity of 6. Conversely, when barometric pressure is rising (b_2) and the concentration of the drug is low (c_0), the patient will transition to a state of no pain with an intensity of 2. For additional detail on the drug effect model, see Nodelman [81].

CTBNs can be thought of as generative models, capable of producing samples of the system state. Since the CTBN is a continuous time model, a sample consists of a set of state assignments for each variable in the model during an interval of time $[t_s, t_e)$. For practicality, this time window is constrained to include only a period of interest, and it is typically assumed that $t_s = 0$. When the state of every variable is known for the entire time interval, the sample is referred to as a complete trajectory. If there are gaps in the knowledge, and the state of one or more variables is unknown for periods throughout the time window, the sample is referred to as a partial trajectory. As a generative model, a CTBN is capable of "completing" a partial trajectory by filling in the unknown gaps. In this way, existing knowledge can be encoded as partial trajectories, and the CTBN can be used to answer queries conditioned on this known information. This is referred to as evidence application, and the information obtained from queries is drawn from a posterior distribution conditioned on this evidence.

2.4.2 Inference and Learning Algorithms

The compact representation afforded by the CTBN framework is only as valuable as the inference algorithms that support it. To date, no tractable exact inference algorithms have been identified that work on the factored representation directly. Instead, exact inference relies on an operation called *amalgamation*, that combines the variables in a CTBN into a single amalgamation of the original variables. By performing this repeatedly on all variables in a model, a single entity is produced, resulting in an unstructured CTMP. Inference can then be performed over the CTMP as usual by computing the matrix exponential, providing an exact solution [**?**]. Although useful from a theoretic standpoint, amalgamation "un-factors" the model, thereby eliminating the benefits provided by the CTBN. To address this, a number of approximate inference algorithms have been proposed that work on the structured representation directly.

The first approximate inference algorithm to be introduced for use with CTBNs is expectation propagation (EP) [83, 84]. This algorithm is based on the clique tree inference procedure in BNs [117]. The primary difference is that the factor product operation in BNs is replaced with amalgamation for CTBNs. Inference starts by forming a clique tree from the CTBN graph structure. Messages are then passed between neighbors in the cluster graph until a consensus is reached regarding the expected distribution. The time granularity at which inference is performed can actually be chosen up front or dynamically. Work has been done to identify an optimal time granularity in real time using an information-theoretic criterion, which can be used to adapt each cluster dynamically based on its rate of convergence [114]. This allows computational resources to be assigned where it is most needed and avoids wasting time working on clusters that do not need additional compute power.

Another approximation algorithm that has been adapted to work on the CTBN framework is mean field variational approximation [24, 25]. The work is based on a variational method designed for the Markov jump process, which is another model built on CTMPs [88]. This algorithm works by approximating the distribution defined by the CTBN using a product of inhomogenous Markov processes. This is achieved using a density transform approach, where the set of all densities is restricted to a computationally manageable class. Specifically, the restricted class of density functions used are those that can be factored into a product of density functions. This transformation into a product of densities is also known as a "mean field approximation," originating from the physics domain. Kullback-Leibler (KL) divergence from the true posterior distribution can be minimized by solving a series of ordinary differential equations (ODEs), resulting in an approximate solution to the inference problem.

Belief propagation (BP) is another variational inference algorithm that builds on concepts from both expectation propagation and mean field approximation [34]. BP functionally operates in much the same way as EP; the difference being in how the approximation is represented. Like the mean field algorithm, BP relies on inhomogeneous CTMPs, requiring a solution to a system of ODEs. This belief propagation algorithm extends work by Yedidia *et al.* [151].

Node isolation is a relatively recent approach to performing approximate inference in CTBNs. The idea is to convert a CIM consisting of multiple intensity matrices and approximating the transition behavior using a single matrix. Three techniques are presented to achieve this: a sample-based approach and two variations of a closed-form approach. Once a node has been isolated, its dependencies are severed, which allows for efficient computation of several inference tasks.

In addition to variational approaches, there have been a number of sample-based inference algorithms introduced for CTBNs [38, 40]. Importance sampling is an algorithm capable of estimating the expected value of any function of a trajectory conditioned on any evidence [40]. The algorithm works by sampling from a proposal distribution P', which is more efficient than drawing samples from the original distribution P. The proposal distribution is constructed in a way that guarantees evidence will be adhered to. To account for the fact that samples are drawn from P' rather than P, a weight is assigned to each sample to indicate the likelihood that the sample was actually drawn from P. Rejection sampling was introduced to eliminate samples responsible for high variance, and has been shown to reduce the amount of time needed to obtain accurate results [142]. Importance sampling has since been extended support efficient online inference and deal more effectively with transitions into evidence periods using predictive lookahead and particle smoothing.

Particle filtering is another approach that has been taken to solve the problem of inference in CTBNs [14, 80]. Continuous time particle filtering is an extension to the more common discrete time particle filtering [31]. For instances where events are non-uniform, continuous time particle filtering demonstrates superior performance both in terms of inference speed and the quality of the approximation, as it is able to skip to any point in time without stepping through discrete time slices. Additionally, a filtering approach is capable of performing inference over a hybrid system containing discrete and continuous state variables by sampling for discrete variables and estimating continuous ones.

Another sample-based approach to inference is the Gibbs sampling algorithm [35]. Gibbs sampling is a member of the class of algorithms referred to as Markov Chain Monte Carlo (MCMC), which essentially amounts to a random walk over the distribution. Gibbs sampling works by sampling a complete, continuous trajectory for a single node X chosen at random, conditioned on the trajectories sampled from other nodes that are influential in the distribution of X. This process is performed repeatedly, resulting in samples that converge to the true distribution. A burn-in period is often performed, which is a fixed number of iterations during which samples are discarded. This removes any bias introduced by the initial state assignment by sampling process transitions prior to recording the samples that are used in computing statistics. An auxiliary Gibbs sampler has also been proposed that employs a technique referred to as uniformization, which approximates the underlying CTMP with a discrete time Markov chain [110, 111]. This allows "virtual" sojourn times to be drawn from a homogeneous Poisson process that defines the Markov chain [128]. This version of the algorithm is shown to be computationally more efficient than traditional Gibbs sampling, and the underlying Markov chain in this approach has since been shown to be geometrically ergodic, providing theoretical guarantees regarding convergence [74]. Finally, another MCMC algorithm adapted for CTBNs is the Metropolis-Hastings algorithm, which also utilizes uniformization, but varies slightly in its implementation when compared to that of Rao and Teh [75]. Specifically, operations in the Metropolis-Hastings algorithm must update just a few sufficient statistics, while Rao and Teh reevaluate the entire trajectory. As a results, Rao and Teh work better with models of moderate size, but is unable to compete with the approach taken by Miasojedow *et al.* on larger models.

In addition to inference algorithms, there have been a variety of algorithms published on learning CTBNs [85]. The first of such works was presented by Nodelman, where a conjugate prior for CTBNs was used as the basis for computing a Bayesian score. This allowed for parameter estimation and structure learning in the presence of complete data. This restrictive assumption was later lifted with the introduction of the expectation maximization (EM) and structural expectation maximization (SEM) algorithms, which can learn CTBNs from partially observed data [86]. EM works by making iterative improvements to parameters such that the expected log-likelihood of the data given the parameters is maximized. EM also allows more complex non-exponential transition distributions to be learned from the data, which is closely related to the work we present in Chapter 3. A recent approach to structure learning was proposed that analytically identifies structure rather than relying on an iterative maximization technique [130]. This is achieved exploiting the Markov chains that are implicitly defined by the CTBN framework. These Markov chains can be marginalized the Markov chain with respect to the parameters, resulting in a parameter free process that support efficient structure learning. Shi and You present an online parameter estimation algorithm that allows for batch updates to CTBN parameters as new samples are collected [121]. Finally, learning algorithms have been developed that exploit data characteristics found in specialized domains, including financial, relational, non-stationary, and [52, 140, 150]. This closely relates to our own work in that we present specialized methods for deriving CTBNs using diagnostic and reliability data. This is discussed in detail in Chapter 7.

2.4.3 Extensions and Applications

One of the major limitations to the CTBN as originally proposed is the requirement that all transition events be modeled using exponential distributions. For this reason, phase-type distributions were introduced as a method for approximating non-exponential distributions [82]. Initially, efforts involving phase-type distributions focused on modeling and largely omitted the process of learning the distributions. Nodelman and Horvits restrict the class of the phase-type distributions, considering only the Erlang subclass, which is highly restrictive and less representationally powerful than other more general representations. To address this issue, Gopalratnam *et al.* extended this work to provide learning algorithms for a more general class of phase-type distributions referred to as Erlang-Coxian [49]. Finally, Nodelman revisited the topic in his work on expectation maximization, which provides a natural method for learning fully general phase-type distributions from data. These extensions are relevant to our own work in phase-type distributions in which we approximate target parametric distributions using heuristic optimization techniques. This is discussed in greater detail in Chapter 3.

Another work that is similar to our own is the partition-based CTBN [141]. This model extension incorporates a set of partitions into the graph structure that attempt to compact parameterization further. Rather than using traditional CIMs, new structures referred to as conditional intensity trees and conditional intensity forests are learned from data. These map common intensities to a single value based on a partition defined over the full joint, rather than making the assumption that intensities change as a function of the parent states. The approach taken by Weiss *et al.* shares some features with the tree-structured CIMs that we introduce in this dissertation, but their goal is ultimately to generalize the representation of the CTBN rather than improve its scalability. More information on work by Weiss *et al.*, as well as our own, is provided in Chapter 5.

Substantial research efforts have been conducted on the CTBN classifier (CTBNC), which is an extension of the CTBN that was introduced to support classification tasks in continuous time [15,127]. The CTBNC represents a class using a single static node containing no parents. This enables classification of a static object conditioned on continuous, temporal evidence. Specialized learning algorithms have been developed for the CTBNC, including numerous scoring functions and a parallel implementation [16–18,138]. Furthermore, the model has demonstrated success when applied to practical classification tasks [19,139].

Another model that incorporates static nodes in the standard CTBN framework is the generalized CTBN (GCTBN) [23,105]. This combines standard CTBN nodes parameterized as Conditional Markov Processes with "immediate" or static nodes. These immediate nodes are treated similarly to a node in a BN, and are parameterized using a conditional probability table to describe their fixed probability distribution. Inference and analysis of the model is achieved by producing an equivalent Generalized Stochastic Petri Net (GSPN), which is able to represent delayed and immediate transitions [20–22]. Another model that attempts to unify concepts from CTBNs and the BN literature is the Asynchronous Dynamic Bayesian Network (ADBN) [104]. This model works by representing a stochastic process using a CTBN, but when inference is required, reasoning is performed over an approximating DBN that is created dynamically. The intent here is to take advantage of mature BN inference algorithms while avoiding explicit specification of a time granularity.

In addition to model extensions, there have been a number of contributions to the literature that focus on improving the representational power of inference queries as well. As originally specified, CTBNs supported only queries concerning probability distributions over the states of variables, and evidence was expected to be applied with absolute certainty. Performance Functions (PFs) have since been introduced to allow for more complex queries of a CTBN [132]. These PFs are capable of defining sophisticated functions beyond just state probability distributions and are able to define such a function using the CTBN's factored graph structure. Several aspects of our work rely on PFs as a method for assigning value to the state of a process, as will be seen in Chapters 4 and 7. A formal definition of the various continuous evidence types and how they relate has also been published [136]. This work also describes how existing inference algorithms can be adapted to support the new evidence types that did not exist previously, including uncertain evidence and various types of negative evidence.

In addition to the applications of CTBNCs and GCTBNs, there have been a number of domains in which the standard CTBN framework has been used successfully to reason over a continuous process. An early model was built to capture the behavior of a user in a system for the purpose of predicting activity and availability [82]. In a similar vein, social network dynamics have also been modeled using CTBNs [37, 39]. By modeling host-level network behavior, effective intrusion detection systems have been constructed [147–149]. Server dependencies, as well as sensor network dependencies, have both been modeled using the CTBN framework [53, 120]. CTBNs have been used to predict the trajectories of moving objects in databases [107]. In the healthcare domain, the framework has been used to reason over medical models to predict cardiogenic heart failure [44, 45]. CTBNs have been extremely successful in performing gene network reconstruction and analysis, and have since identified several regulators [2–4, 42]. Finally, CTBNs have been used to perform mechanical diagnostics and prognostics in a limited capacity [13, 152]. It is this last domain of diagnostics and prognostics that drives our own research, and

Chapter 7 provides a number of extensions that decrease the barrier to entry when applying CTBNs to this domain.

CHAPTER THREE

NON-EXPONENTIAL PARAMETRIC DISTRIBUTIONS

In addition to factors like scalability, a model must also be sufficiently expressive to warrant adoption in real-world applications. In many situations, systems are known to change state according to known parametric distributions. If the distribution is exponential, the transition behavior can be modeled directly with a CTBN using the rate parameters in the CIMs. The concern is that non-exponential parametric distributions are not inherently supported by the CTBN framework. In this chapter, we describe a method for automatically modifying a CTBN to approximate non-exponential parametric distributions. We also make an argument for an interpretation of CTBN parameters that best lends itself to this approach.

3.1 Background

To date, work on extending CTBNs through the use of phase-type distributions has focused on the problem of approximating a distribution that describes available data. This has been achieved by applying expectation-maximization algorithms to learn a model that optimizes the log-likelihood of the data. While this approach works well in the presence of abundant data, in many practical applications there is inadequate data to describe a distribution sufficiently. Fitting a distribution to data requires a sufficient number of data points to measure goodness of fit [8]. A viable alternative is to rely on domain knowledge to indicate transition rates with a well-defined parametric distribution. The purpose of this chapter is to detail how these parametric distributions can be learned and modeled within the CTBN framework, and as such we will refer to these probability distributions as “target” distributions.

3.1.1 Phase-Type Distributions

Phase-type distributions are a semi-parametric class of distributions that use combinations of exponential distributions as a means to approximate general positive distributions. Formally, phase-type distributions represent the time until absorption in a Markov process with n transient states and one absorbing state [1]. The transient states in the Markov process are also referred to as the phases of the phase-type distribution. A variable will move through these phases according to the exponential distributions defined for each phase by the rates in the Markov process, until the variable eventually reaches the absorbing state. The phase-type distribution is defined as the distribution of the entire time it takes the process to move through these phases.

Since a phase-type distribution depends only on a Markov process, its parameters are specified fully by the initial distribution and transition intensity matrix for the Markov process. It is generally assumed that the probability of starting in the absorbing state is zero, and in many cases it is further restricted to ensure that the process starts in the first transient phase.

The movement of a variable through the transient phases of the Markov process can be directed in a variety of ways. In the most general case, all transient phases are capable of transitioning to any other phase, including the absorbing state. Specific classes of phase-type distributions restrict the movement of a variable between the transient phase. One such type of restricted phase-type distribution is the Erlang distribution, in which loops are not permitted, and each phase is required to have the same rate parameter and must be visited in order before transitioning to the absorbing state [**?**]. A slightly more general class is the Erlang-Coxian phase-type distribution [**?**]. A Coxian distribution is similar to an Erlang distribution in that it does not permit cycles in the phases. In contrast, however, a Coxian distribution may be parameterized uniquely at each phase, and any of the phases may transition to the absorbing state. An Erlang-Coxian distribution combines these two ideas by forcing sequential progression through the Erlang phases until the Coxian phases are reached, at which point the phase may either transition to the next phase, or go to the absorbing state directly. Both of these restricted classes have been applied to CTBNs [49, 82].

The PDF for a phase-type distribution is given as:

$$f(t) = \alpha \exp(\mathbf{S}t)\mathbf{S^0}, \tag{3.1}$$

where α is a vector corresponding to the probability of starting in each phase, $\mathbf{S^0}$ is a vector of intensities for transitioning to the absorbing state from each of the other phases, and $\mathbf{S}$ is a square matrix of intensities for transitioning between non-absorbing phases. Recall from the previous chapter that the matrix exponential operation, $\exp(\mathbf{S})$, is defined by the Taylor series as:

$$\exp(\mathbf{S}) = \sum_{k=0}^{\infty} \frac{1}{k!}\mathbf{S}^k. \tag{3.2}$$

Generally calculating the matrix exponential is intractable. Fortunately, a variety of methods exist for calculating an approximate matrix exponential [77]. The work presented in this chapter utilizes the most commonly used approximation, known as the scaling and squaring method [54].

The idea behind scaling and squaring is to scale the matrix by a power of two to reduce the norm to order one, and then compute a Padé approximant[1] for the matrix exponential on the scaled matrix. Repeatedly squaring the resulting matrix undoes the scaling. By using the scaling and squaring method to calculate the matrix exponential, approximating the PDF of a phase-type distribution becomes tractable. The ability to approximate this PDF becomes important when evaluating the quality of a phase-type approximation for a target distribution.

3.1.2 Parametric Distributions

A parametric distribution, unlike an unknown distribution described by available data, is specified entirely by its parameters. Using target distributions as a means to describe system behavior is an established practice when performing tasks such as failure rate analysis [41]. Learning phase-type distributions from target distributions provides information about transition times for a variable while avoiding the need for significant amounts of data. The problem of learning CTBN-compatible representations of a target parametric distribution may be cast as an optimization problem that seeks to minimize the KL-divergence of a learned phase-type distribution from the target distribution. This is equivalent to maximizing the closeness of the approximation. Section 3.4 describes the learning process in more detail and presents several optimization techniques that can be applied to this problem.

A major motivation underpinning this dissertation is the desire to model system failures in continuous time. For this reason, the subset of distributions discussed in this chapter consist of those that are commonly used to model time-to-failure (TTF); specifically, the Weibull and Lognormal distributions [56]. Exponential

[1] An $[n, m]$ Padé approximant is a rational function consisting of a polynomial of degree m divided by a polynomial of degree n, which is useful for providing an approximation of the power series of another function [7].

distributions are also frequently used to model failure distributions, but CTBNs are already naturally suited to model these distributions. Note that although the scope of discussion has been limited to Weibull and Lognormal distributions, the framework being proposed allows for the approximation of any positive, continuous parametric distribution. The remainder of this section provides background on the Weibull and Lognormal distributions.

The Weibull distribution is a flexible distribution parameterized by a rate parameter λ and a shape parameter k [29]. Consider the case where the input t to the distribution is interpreted as the TTF. When $k < 1$, the Weibull distribution represents a decreasing failure rate, and an increasing failure rate when $k > 1$. In the special case where $k = 1$, the Weibull distribution reduces to an exponential distribution with a rate of λ. When t is positive, the probability density function (PDF) for the Weibull distribution is defined as follows:

$$f_t(\lambda, k) = \frac{k}{\lambda}\left(\frac{t}{\lambda}\right)^{k-1} \exp\left(-\left(\frac{t}{\lambda}\right)^{k}\right) \ .$$

Since the failure rate for a Weibull distribution can be increasing, decreasing, or constant, it can be used to model the “infant mortality” stage (where $k < 1$), the “useful life” stage (where $k = 1$), and the “end of life” stage (where $k > 1$) of an object. When spliced together in this order, these three piece-wise segments form what is often called the “bathtub” curve. This is considered an appropriate model for the failure behavior of many objects, as it reflects a higher rate of failure surrounding the birth and death of an object, and a smaller constant rate of failure during the rest of the object’s lifespan. Additionally, it has been suggested that this composite approach may prove to be a more realistic model for TTF than monotone failure rate

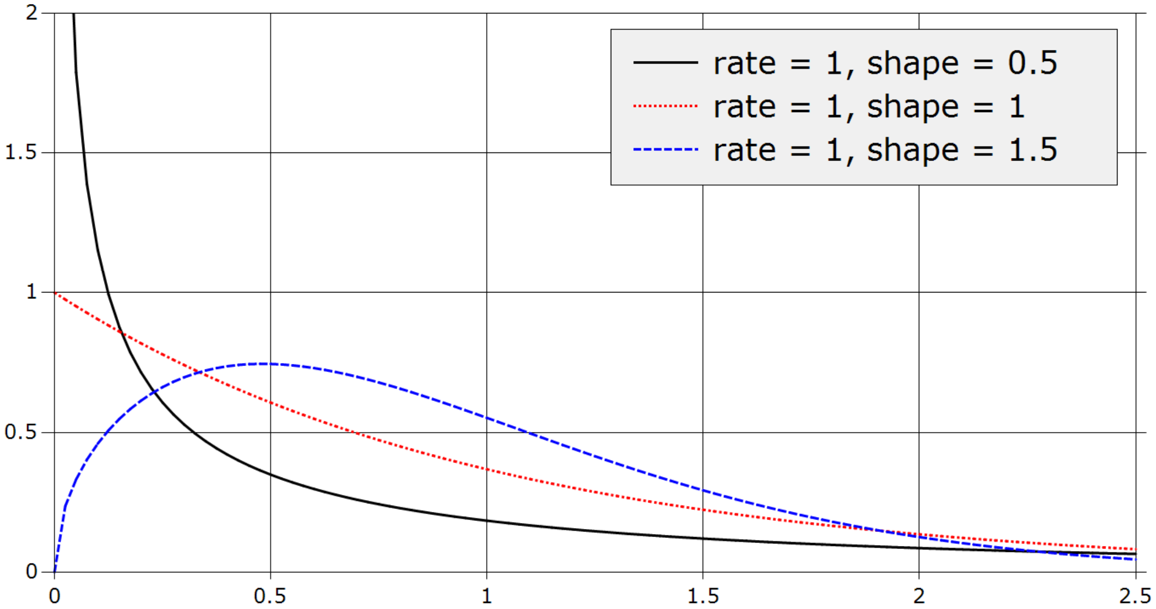


Figure 3.1: PDFs for various parameterizations of Weibull distributions.

models [109]. The plot for this distribution using several different parameterizations is shown in Figure 3.1.

Another frequently used TTF curve is the Lognormal distribution, which indicates that the log of a random variable follows a normal distribution. Due to its dependence on the normal distribution, the only necessary parameters for the Lognormal distribution are the mean parameter μ and the standard deviation parameter σ. The Lognormal distribution is suited for instances where failure occurs due to an accumulation of causes that have a multiplicative effect, a phenomenon known as multiplicative degradation [126]. When the input t is positive, the PDF for the Lognormal distribution is as follows:

$$f_t(\mu, \sigma) = \frac{1}{t\sigma\sqrt{2\pi}} \exp\left(- \frac{(\ln t - \mu)^2}{2\sigma^2} \right) .$$

Figure 3.2 provides a plot of various parameterizations for this distribution.

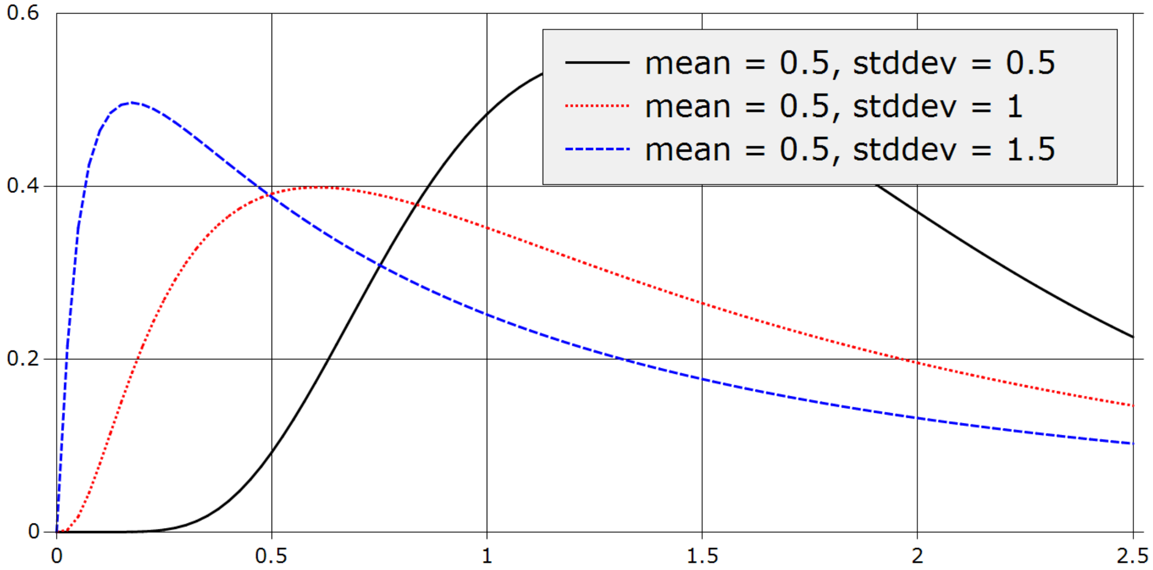


Figure 3.2: PDFs for various parameterizations of Lognormal distributions.

3.2 Related Work

The use of phase-type distributions for CTBNs was first proposed by Nodelman and Horvitz [82]. They demonstrated that phase-type distributions can be inserted as subsystems in a CTBN without altering the underlying mechanics of the model. Their work was restricted to Erlang distributions, which was shown to improve the performance of the CTBN model when the underlying distribution was non-exponential. The primary contribution made by the authors is the notion that phase-type distributions can be inserted into a CTBN without changing the framework as a whole. The specific details of how to parameterize the phase-type distributions, as well as the details regarding how these phase-type distributions are embedded into the CIMs, were omitted. The work by Nodelman and Horvitz was later extended from Erlang distributions to Erlang-Coxian distributions, which is a more expressive subclass of phase-type distributions [49].

The addition of the Coxian phases increases the expressiveness of the model but at the cost of added complexity through an increase in the required number of parameters. Instead of requiring a single rate parameter λ, two additional parameters are required for each phase in the Coxian distribution. This additional complexity can be managed by restricting the Coxian distribution to only two phases. Gopalratnam *et al.* propose a method for learning these parameters from data based on EM. Unlike the method we describe in this chapter, their technique attempts to parameterize the distribution such that the log-likelihood between the data and the model is maximized.

After demonstrating the utility of Erlang distributions as subsystems in CTBNs, in subsequent work Nodelman *et al.* discuss a method for performing EM and structural EM (SEM) to learn the parameters and structure of a CTBN model from partially observed data [86]. This EM algorithm further relaxes the restrictions on which subclasses of phase-type distributions are applicable, such that any phase-type distribution can be used. This allows phases to occur within a loop, significantly improving the expressiveness of the model. The experiments showed a marked improvement over a CTBN model that did not use phase-type distributions. Nodelman *et al.* demonstrated the utility of this approach, but we explore a different angle with our contributions. While Nodelman learns phase-type distributions from data, we learn the parameters for a phase-type distribution to fit a known distribution.

3.3 Embedding Process

Although other work has briefly explained how to embed phase-type distributions into the intensity matrices of a CTBN node, much of the details have been omitted about the process [82]. This section provides a more formal treatment of how phase-type distributions are inserted into intensity matrices and what information is necessary to produce such an embedding. As distinct from other descriptions, the procedure described here makes no assumption about the class of phase-type distribution and works for the most general case.

Let n_i be the number of phases in a phase-type distribution for the i^{th} state of a CTBN node. The phase-type distribution PT_i for the i^{th} state can be described by an entrance distribution $\alpha^i = (\alpha^i_1 \cdots \alpha^i_{n_i})$ and a transition intensity matrix $\mathbf{S}_i$.

$$\mathbf{S}_i = \left[\begin{array}{c|c} & \beta^i_1 \\ \mathbf{R}_i & \vdots \\ & \beta^i_{n_i} \\ \hline 0 \cdots 0 & 0 \end{array}\right] \tag{3.3}$$

Here, $\mathbf{S}_i$ is an $(n_i+1)\times(n_i+1)$ matrix whose last row is an absorbing state of all zeros. This matrix can be decomposed as an $n_i \times n_i$ block matrix $\mathbf{R}_i$ and an n_i-element exit distribution $\beta^i = (\beta^i_1 \cdots \beta^i_{n_i})$. $\mathbf{R}_i$ indicates transitions between transient states, while β^i consists of transitions to the absorbing state. Note that while each row of $\mathbf{S}_i$ is required to sum to zero, this will not be the case for $\mathbf{R}_i$ due to the omission of the β^i_j entry in each of the rows. For this reason, $\mathbf{R}_i$ is referred to as a rate matrix rather than an intensity matrix.

Phase-type distributions can be embedded into an intensity matrix $\mathbf{Q}$ with n states by representing each individual entry in the intensity matrix with a block matrix. For diagonal entries $q_{ii} \in \mathbf{Q}$, the replacement block matrices are the $n_i \times n_i$ rate matrices $\mathbf{R}_i$. This means that the off diagonal entries in $\mathbf{Q}$ also need to be replaced with matrices. We denote the matrix used to replace entry q_{ij} as $\mathbf{T}_{i \to j}$, and describe its construction later in this section. Let m be the number of rows or columns in the original matrix $\mathbf{Q}$, equating to the number of states for the variable.

The result after the embedding the matrices $\mathbf{R}_i$ and $\mathbf{T}_{i\to j}$ into $\mathbf{Q}$ is an $n' \times n'$ intensity matrix $\mathbf{Q}'$, where $n' = \sum_{0\leq i<m} n_i$.

Recall that $-q_{ii} \in \mathbf{Q}$ describes the time spent in state i, with the constraint that this time must be distributed exponentially. $\mathbf{R}_i \in \mathbf{Q}'$ also represents the amount of time spent in state i, but does so using a phase-type distribution that can approximate a broader class of distributions. Additionally, each $q_{ij} \in \mathbf{Q}$ indicates a transition from state i to state j. $\mathbf{T}_{i\to j}$ also indicates this transition between states, but includes additional information about which phase in state i and j are involved in the transition. Specifically, an entry t_{kl} in row k, column l of $\mathbf{T}_{i\to j}$ indicates a transition from state i, phase k to state j, phase l. The resulting block matrix is as follows:

$$\mathbf{Q}' = \begin{bmatrix} \begin{bmatrix} \mathbf{R}_1 \end{bmatrix} & \begin{bmatrix} \mathbf{T}_{1\to 2} \end{bmatrix} & \cdots & \begin{bmatrix} \mathbf{T}_{1\to n} \end{bmatrix} \\ \begin{bmatrix} \mathbf{T}_{2\to 1} \end{bmatrix} & \begin{bmatrix} \mathbf{R}_2 \end{bmatrix} & \cdots & \begin{bmatrix} \mathbf{T}_{2\to n} \end{bmatrix} \\ \vdots & \vdots & \ddots & \vdots \\ \begin{bmatrix} \mathbf{T}_{n\to 1} \end{bmatrix} & \begin{bmatrix} \mathbf{T}_{n\to 2} \end{bmatrix} & \cdots & \begin{bmatrix} \mathbf{R}_n \end{bmatrix} \end{bmatrix}.$$

The matrix $\mathbf{Q}'$ can be constructed using a phase-type distribution PT_i and a multinomial state transition distribution $\rho^i = (\rho^i_1 \cdots \rho^i_{n_i})$ for each state i. Each $\mathbf{R}_i$ block matrix in $\mathbf{Q}'$ can be obtained directly from the decomposition of $\mathbf{S}_i$. The only remaining task is to construct each matrix $\mathbf{T}_{i\to j}$.

Let $\mathbf{e}^i_j$ denote the unit vector corresponding to the j^{th} row or column of an identity matrix I_{n_i}. In other words, $\mathbf{e}^i_j$ is an n_i-element vector where the j^{th} element is one, while all other elements are zero. The process for constructing each matrix

$\mathbf{T}_{i \to j}$ using the multinomial distribution for each state, along with the entrance and exit distributions for the phase-type distributions is as follows:

$$T_{i \to j} = (\mathbf{e}^i_j \cdot \rho^i) \cdot \beta^i \alpha^j.$$

Note that the multinomial distribution is already defined by a typical intensity matrix based on the ratio between the transition intensities for each state in a row. This means that the only additional parameters required for the embedding process are the parameters for a phase-type distribution. Given a method for learning the parameters for a phase-type distribution that approximate another parametric distribution, the user needs only to specify the new parametric distribution to replace the usual exponential distribution for each state. Moreover, this entire process reduces to a standard intensity matrix when an exponential distribution is chosen that is approximated with a phase-type distribution with only one phase. In this case, the approximation is perfect since a single phase is sufficient to approximate an exponential distribution, and each $\mathbf{R}_i$ and $\mathbf{T}_{i \to j}$ block matrix in $\mathbf{Q}'$ contains only a single entry equivalent to the original entries in the matrix $\mathbf{Q}$.

3.4 Learning Phase-Type Distributions

The intent of this chapter is to provide a method for learning a phase-type distribution that accurately approximates a given target distribution. To achieve this, the task of parameterizing the approximating distribution is cast as an optimization problem. This problem is then solved, resulting in a parameterization of a phase-type distribution that accurately approximates the specified target distribution.

As an example, consider a Weibull distribution with a rate parameter of 1.0 and a shape parameter of 1.5. The learned initial distribution and transition matrix for a

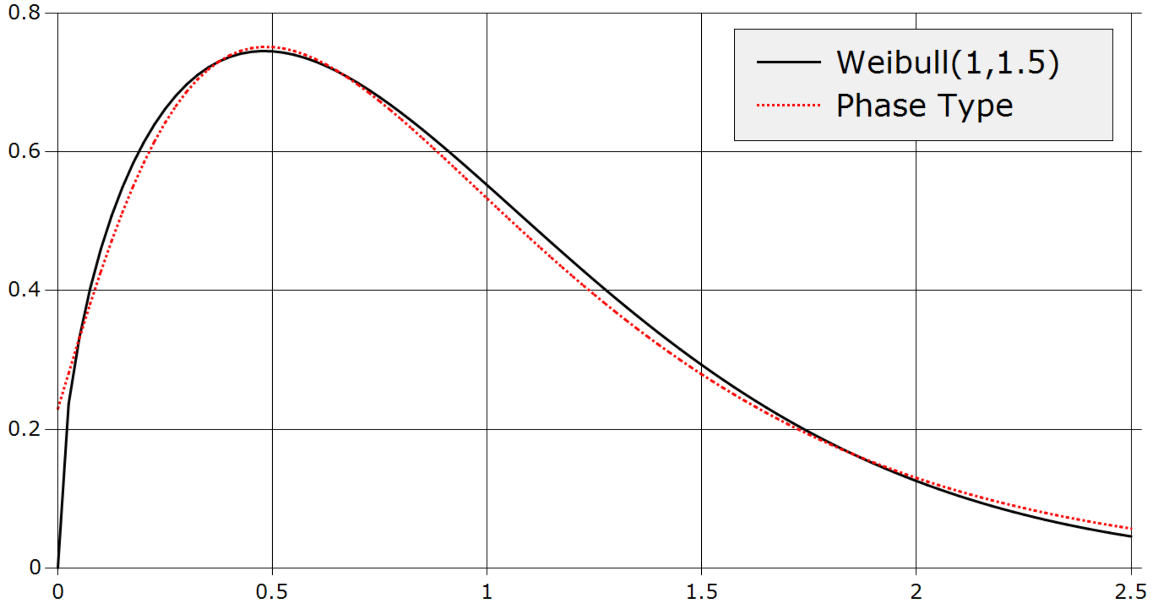


Figure 3.3: Example of a phase-type distribution fitted to a Weibull distribution with a rate of 1.0 and a shape of 1.5.

phase-type distribution that approximates this Weibull distribution is as follows:

$$P(X) = \begin{bmatrix} 0.98 & 0.02 & 0.00 \end{bmatrix},$$

$$\mathbf{Q} = \begin{pmatrix} -2.09 & 1.98 & 0.11 \\ 0.00 & -1.99 & 1.99 \\ 0.00 & 0.00 & 0.00 \end{pmatrix}.$$

A plot of the PDFs for both the Weibull distribution and the approximating phase-type distribution is shown in Figure 3.3. For reference, the goodness of fit for this approximation is quantified by a KL-divergence value of 0.0361.

3.4.1 Kullback-Leibler Divergence

The optimization process makes use of the Kullback-Leibler divergence measure, often referred to simply as KL-divergence. KL-divergence serves as a measure of the information lost when approximating a target distribution with an approximating distribution and is equal to zero if the two distributions are identical. Using this principle, it is possible to construct an accurate approximation by choosing phase-type parameters that minimize the KL-divergence of the phase-type distribution from the target distribution.

The KL-divergence of distribution Q from distribution P is denoted $D_{KL}(P||Q)$. For the case when P and Q are continuous, KL-divergence is defined as follows.

$$D_{KL}(P||Q) = \int_0^\infty P(t) \log \frac{P(t)}{Q(t)} \,\mathrm{d}t \tag{3.4}$$

In practice, a discrete approximation can be used that evaluates the PDF for each distribution at specified intervals. The work presented in this chapter uses the following discrete version:

$$D_{KL}(P||Q) \approx \sum_{i=1}^{n} P(i) \left| \log \frac{P(i)}{Q(i)} \right|. \tag{3.5}$$

Note that in addition to evaluating the distributions at discrete intervals, the upper bound of the summation is restricted to be a finite value. An upper bound of 2.5 was specified for this application, which is deemed the area of interest for both the Weibull and Lognormal distributions in this work. An automated approach could be taken that chooses the upper bound based on the percentage of the distribution covered, as determined by the cumulative distribution function (CDF). Furthermore, the equation contains an absolute value operator applied to the log term, which is a modification to the standard KL-divergence equation. This is to avoid situations where underestimation in one section of the approximate PDF might mask overestimation in another, which may occur due to the fact that the log term can be positive or negative.

Calculating Equation 3.5 requires knowledge of the PDFs of both distributions. The PDF for the target distribution is unique to each distribution, but it is generally a trivial calculation for most parametric distributions. We use the scaling and squaring approximation of the PDF for a phase-type distribution, along with the PDF for the target distribution, to compute the KL-divergence.

3.4.2 Optimization

Selecting parameters for the phase-type approximation of a target distribution is an optimization problem that seeks to minimize KL-divergence. Particle Swarm Optimization (PSO) has demonstrated utility in solving these types of problems, and for a baseline comparison, a genetic algorithm (GA) and a hill-climbing algorithm were implemented.

3.4.2.1 Particle Swarm Optimization

PSO begins by initializing a population of particles, each of which has a position in the search space that represents a possible candidate solution [58]. The quality of each particle's position can be evaluated using a fitness function that is problem-specific. Particles move through the search space as defined by their velocity, which is updated during each iteration of the algorithm according to the following velocity update equation:

$$v_i = \omega v_i + U(0, \phi_1) \otimes (p_i - x_i) + U(0, \phi_2) \otimes (p_g - x_i).$$

Here, v_i is the velocity for particle i, x_i is the position of particle i, p_i is the best position seen by particle i, and p_g is the best position seen by any particle in the swarm. In this application, KL-divergence is the fitness function, and the "best" solution is defined to be the solution with the lowest KL-divergence value. The first term is known as *inertia*, which pulls the particle in the direction it was previously going. The second term is the *cognitive component*, which draws the particle toward the best solution it has ever found, and the third is the *social component*, which draws the particle toward the best solution any particle in the swarm has ever found. The parameters ϕ_1, ϕ_2, and ω are user-defined constants that are tuned manually to control the degree to which each term influences the particle's movement.

PSO was chosen due to its success in related applications [59, 113]. Additionally, it is an anytime algorithm, which gives the user the option to accept a suboptimal solution prior to convergence. This can be especially useful when attempting to approximate a user-specified distribution, since a solution may be deemed sufficient by a user during the modeling process.

3.4.2.2 Genetic Algorithm Genetic algorithms (GAs), as their name suggests, are an attempt to bring the advantages of Darwinian evolution to optimization [76]. The idea behind a GA is to start with a population of randomly generated solutions, which are in this case assignments of values to the parameters of a phase-type distribution. Each solution in the population is called an *individual* or *chromosome*, and in this case the fitness for these solutions is calculated as their KL-divergence from the target distribution. Iteratively, a new "offspring" population is created from the existing population as follows.

First, two parent chromosomes are chosen such that assignments with lower KL-divergence are more likely to be selected. In this work, weighted tournament selection was employed during the implementation of the genetic algorithm. This compares a small pool of candidate parents uniformly selected from the population and chooses a parent from this pool using a probability distribution weighted by fitness. The parent is then returned to the population and another parent is selected in the same manner; thus, it is possible to have the same chromosome as both parents.

Once the parents have been selected, crossover takes place. This work uses multi-point crossover, which involves randomly selecting sections of parameters and swapping them between the parents to create two offspring chromosomes. Experimentation showed that five-point crossover, which involves swapping three sections, gave the best results. The produced offspring are then mutated, a process that randomly changes the values of some of the parameters by a small amount.

This process repeats until a desired number n of new offspring have been created, at which point the generation is completed and is used to replace all n individuals from the old population. The algorithm is repeated for a specified number of generations, after which the fittest individual in the population is returned as the solution.

3.4.2.3 Hill-Climbing with Simulated Annealing Simulated annealing is a technique for improving basic hill-climbing search. It works by using a numerical analogue to the process of slowly cooling metals so that they crystallize at their minimum energy state [106]. In the original hill-climbing algorithm, the candidate solution, which in this case is a parameterization of the model, is initialized at a random state in the search space. Then, at each iteration, a random neighbor state is considered. If the neighboring state is found to have a better fitness value than the current state, the neighbor is accepted and becomes the current state. In this implementation, a better fitness value is defined as a lower KL-divergence.

Simulated annealing introduces an extra step at each iteration to avoid becoming stuck in local optima. Once a neighbor state is selected, if the neighbor is worse, it is accepted or rejected based on the acceptance probability:

$$P(accept) = \exp\left(\frac{\text{energy}(current) - \text{energy}(neighbor)}{kT}\right)$$

where the energy of a given parameterization is its KL-divergence from the target distribution, k is the Boltzmann constant, and T is a value known as the *temperature*, which is initialized to some positive number and is slightly decreased at each iteration. A better neighbor state will still always be accepted, but now a worse solution may also be accepted based both on how much worse it is and on the temperature at that iteration.

The gradual lowering of the temperature parameter produces the desired annealing effect: the likelihood of accepting a worse solution is initially high but decreases as a function of time. This enables the algorithm to avoid becoming stuck in local optima early in the search process while still converging on a close-to-optimal solution in later iterations of the search.

3.5 Experiments

In addition to comparing optimization algorithms, this section explores the effects of several other factors on the ability to learn phase-type approximations. Specifically, experiments are designed to determine how an increase in the number of phases might improve the expressive power of the model. Additionally, several alternatives to random initialization of solutions during optimization are presented and evaluated.

3.5.1 Optimization Methods

We ran the PSO, GA, and hill-climbing with simulated annealing (SA) optimization algorithms on various parameterizations of Weibull and Lognormal distributions. For the Weibull distribution we varied both the rate and shape parameters from 0.5 to 2.0 by increments of 0.1 for a total of 225 instances. The same values were used with the Lognormal distribution for the mean and standard deviation parameters.

Each configuration was run 100 times to account for the stochastic nature of each of the optimization algorithms.

For this initial experiment, the size of the learned distribution was fixed at three phases. We employed general phase-type distributions, which have no structural restrictions. The intensity matrix and initial distribution for each phase-type distribution were serialized so that they conformed to the optimization algorithm frameworks. This was done by extracting all non-diagonal entries from the intensity matrix excluding the last row. This is because the diagonal entries for each row of the intensity matrices can be computed as the negative sum of the rest of the row, and the last row was always set to 0 since the corresponding state is absorbing. We also included all of the values in the initial distribution in the serialization and normalized the values to ensure that the probabilities sum to 1.0. The result is a vector of size $\Theta(n^2)$, where n is the number of phases. We bounded the valid search space for the optimization such that the initial distribution values were smaller than one, while the entries for the intensity matrix were bounded by some positive user-specified value, which we set to 2.00 for all experiments. The deserialization process reversed the described procedure to produce a phase-type distribution that can be used to evaluate candidate solutions for the optimization algorithms.

Algorithms were compared using the KL-divergence between the final learned phase-type distribution and the target distribution. We used 10-fold cross validation, and manually tuned the parameters for each algorithm to improve performance. Hill-climbing used an initial temperature of 100 and decreased this value by 0.05 at each iteration. For the genetic algorithm, we used 100 individuals, five-point crossover, and two-parent tournament selection with a 0.75 probability of choosing the fittest parent. PSO used five particles, the behavior of which was dictated by a velocity update equation with an inertia of 0.9, a personal learning rate of 1.0, and a social

Table 3.1: Comparison of Optimization Algorithms

	PSO	GA	SA
Weibull	**0.0498**	0.0815	0.2996
Lognormal	**0.0154**	0.0394	0.0888

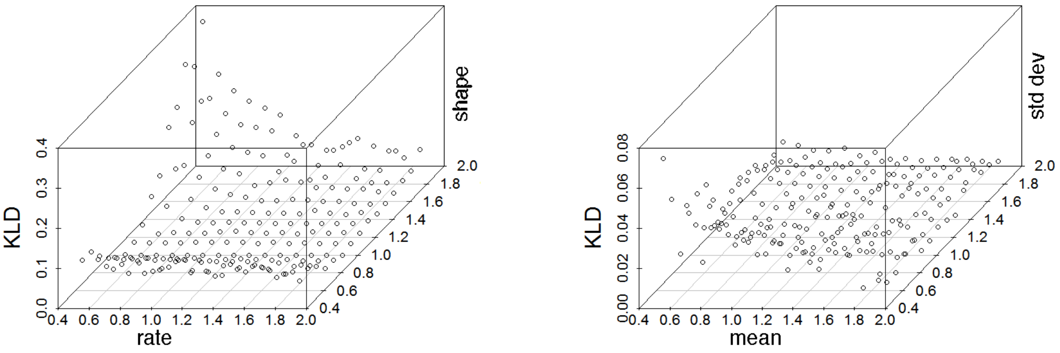
learning rate of 1.5. Each of the algorithms were run for 1000 iterations, which in all cases appeared to be sufficient for convergence.

The mean performances for each algorithm measured in KL-divergence from the target distribution are summarized in Table 3.1. Algorithms were compared with the Wilcoxon signed-rank test with a confidence level of 0.95, a nonparametric test chosen because the datapoints originate from different distributions. Based on our statistical analysis, PSO significantly outperforms the GA, which in turn significantly outperforms SA. For this reason, we focus the rest of our experiments on how well PSO performs under varying conditions and omit any further results for GA and SA.

We also investigated PSO's ability to approximate distributions over different regions of the parameter space. As specified above, we used the 225 parameterizations for Weibull and Lognormal and plotted the KL-divergence values obtained using PSO in Figure 3.4. Figure 3.4a illustrates that relatively low KL-divergence values are obtained for the majority of the Weibull search space, with the exception of those cases when the rate is low and the shape is high. Similarly, we find from Figure 3.4b that most parameterizations of the Lognormal distribution can be approximated well, but performance degrades when both the mean and standard deviation are low.

3.5.2 Number of Phases

The next experiment investigated the effect of varying the number of phases in the phase-type distribution. Phase-type distributions with more phases have more representational power, therefore allowing for a more accurate approximation.



(a) Approximating Weibull (b) Approximating Lognormal

Figure 3.4: KL-divergence values for approximating parameterizations of Weibull and Lognormal distributions using PSO.

However, more phases implies more parameters in the resulting CTBN, with a corresponding increase in model complexity.

For this experiment, we used five representative parameterizations each for the Weibull and Lognormal distributions. These parameterizations are intended to cover a variety of common distribution shapes that may be used to specify the behavior of a continuous system. For the Weibull distribution, these values were $(0.8, 1.7)$, $(1.0, 1.5)$, $(1.3, 1.7)$, $(0.7, 0.7)$ and $(1.0, 0.5)$, where the first value in each pair is the rate and the second value of the pair corresponds to the shape. In the case of the Lognormal distribution, we used $(0.8, 1.2), (1.0, 1.2), (1.0, 1.0), (1.2, 1.0)$ and $(0.95, 1.0)$, where the first value of each pair is the mean and the second is the standard deviation. Once again, each configuration was run 100 times to account for stochasticity in the optimization algorithms.

For each of these ten target distributions, we used PSO to learn phase-type distributions with varying numbers of phases. Specifically, we started with a single phase (which is equivalent to the exponential distribution) and increased

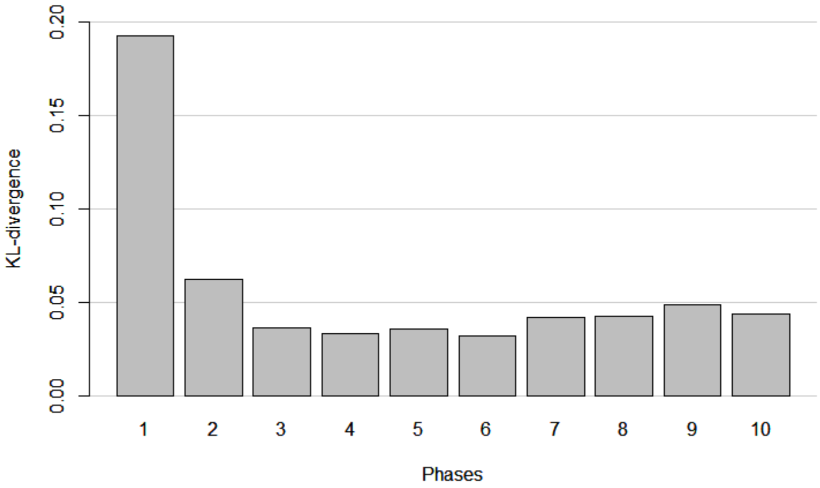


Figure 3.5: A comparison of KL-divergence values when using varying numbers of phases.

incrementally to ten phases. A summary of the results is shown in Figure 3.5. Our analysis of these results consisted of a series of Wilcoxon signed-rank tests with a confidence level of 0.95. Of particular note is that a single-phase distribution was significantly outperformed by every other case, and using only two phases was significantly worse than using three, four, five, and six phases. It is also interesting that the approximations become slightly worse as the number of phases increases, which is likely a result of the increased search space leading to a more challenging optimization problem. Essentially there is no substantial advantage obtained from the increased representational power when using a large number of phases, and worse approximations are obtained by optimizing over the increased number of parameters in the approximating phase-type distribution. While this is the case for the Weibull and Lognormal distributions, more challenging target distributions may require a larger number of phases.

3.5.3 Informed Initialization

Our final experiment tested an extension to our proposed algorithm, which we call informed initialization. The idea is that rather than initializing particles randomly, we can use a more intelligent starting solution. This is accomplished by first approximating a variety of distributions, which can be chosen using a grid search over the parameters. An approximation for each of these distributions is learned using randomly initialized particles and the resulting parameters for the phase-type distributions are saved. When learning a new distribution, the algorithm can then initialize particles using a similar cached solution by calculating the sum of the differences between parameters and sorting the list.

For this experiment, the set of saved solutions was generated using the experiments run in Section 3.5.1, resulting in 225 potential starting positions for each distribution. For the target distributions, we used the ten distributions discussed in Section 3.5.2 and the parameters for each were perturbed by ± 0.05 so that they could not be found exactly in the set of saved solutions. We varied the number of particles in the swarm that were initialized using a saved solution from zero (equivalent to random initialization) to all five. When multiple particles used informed initialization, solutions were drawn from the saved set in order of similarity.

The performance of the algorithm as a function of the number of iterations for the cases when zero, one, and five particles use informed initialization is shown in Figure 3.6. This chart shows results for 100 runs approximating a Weibull distribution, although similar results were found for Lognormal. Using more particles initialized with informed starting positions resulted in faster convergence to lower KL-divergence values. We omitted two, three, and four initialized particles from the graph for clarity, but we noted that there was an incremental decrease in the KL-divergence for each case. We also used a Wilcoxon signed-rank test with a confidence level of 0.95 to

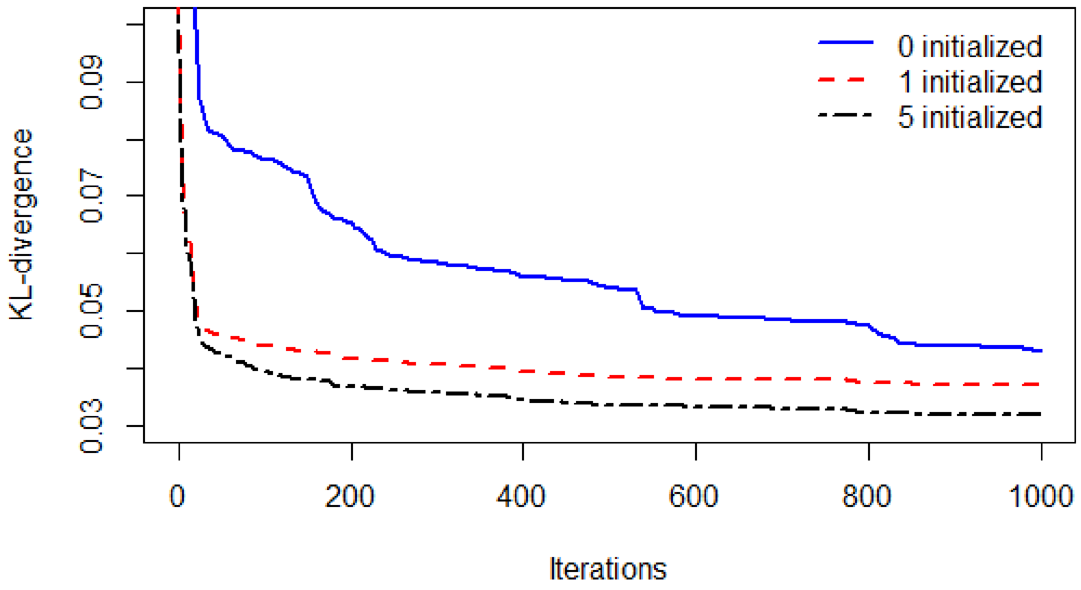


Figure 3.6: Effect of informed initialization on KL-divergence.

compare the results of zero initialized particles to five initialized particles after every 200 iterations. Results indicate that the informed initialization performs significantly better than random initialization at 0, 200, 400, 600, and 800 iterations. Although informed initialization appears to perform better after 1000 iterations as well, the decrease in KL-divergence from standard PSO is no longer statistically significant.

3.5.4 Discussion

The experiment from Section 3.5.1 showed that of the optimization methods considered, PSO performed best. In addition, Figure 3.4 gives a sense of how well phase-type distributions are able to approximate various parameterizations of Weibull and Lognormal distributions. KL-divergence values were higher when the target distribution had harsher peaks in its PDF. This indicates that phase-type distributions are better at approximating smooth distributions. Intuitively, using a larger number of phases should mitigate this problem.

The conclusion to be drawn from the experiment in Section 3.5.2 is that a phase-type distribution with a single phase or few phases may lead to unsatisfactory approximations. Six phases seems optimal, as further increasing the number of phases does indeed increase expressiveness but also makes optimization more difficult. Since the search space increases quadratically with the number of phases, adding more phases greatly expands the parameter space the optimization method must search. For the distribution we investigated, we found that three phases is likely sufficient to get a reasonable approximation, and two phases may also work when model complexity is a concern.

As discussed in Section 3.5.3, informed initialization does appear to improve the approximations. Initializing the entire swarm in this way produced significantly better results at intermediate stages of the optimization process. In addition, Figure 3.6 shows that the solution converges much faster when informed initialization is used, which could be beneficial to applications where learning time is important.

3.6 Parametric Representation and Interpretation

This chapter has described how approximations of non-exponential parametric distributions can be learned and embedded into the CTBN framework. Before proceeding to the next chapter, we first take a brief moment to discuss the ramification of this work the semantic interpretation of CTBN parameters. Recall from the previous chapter that there are two equivalent parametric representations. A pure intensity representation directly uses the intensities from an intensity matrix, and is associated with a racing exponential interpretation. A mixed intensity representation uses a single rate to determine a transition time and determines the location of the transition using a discrete distribution over the possible destination states. With the potential for embedding approximations of more complex distributions, it is our contention that the mixed intensity representation is a more general approach to viewing the parameters and can be readily extended to describe more complex, non-exponential parametric distributions.

Traditionally, the mixed intensity representation for homogeneous CTMPs are comprised of two sets of parameters; namely, a set of rates $\mathbf{q}$ defining transition times and a set of discrete distributions $\boldsymbol{\theta}$ defining transition destinations. Equivalently, $\mathbf{q}$ is a set of parameterized exponential distributions defining transition times for each state. With the ability to approximate non-exponential distributions using phase-type distributions, the set $\mathbf{q}$ can be generalized to contain any parameterized distribution, so long as it is positive and continuous. The discrete distributions defined by $\boldsymbol{\theta}$ remain the same.

For example, an intensity matrix for a node with three states might be defined using $\mathbf{q} = \{\text{Weibull}(1.0, 1.5), \text{Exponential}(5.0), \text{Lognormal}(3.0)\}$. In other words, the expected sojourn time when the process is in each state is defined by a different distribution. A phase-type distribution can be learned for the Weibull and Lognormal distributions, resulting in a matrix of the form shown by Equation 3.3. The final intensity matrix can then be constructed using the embedding process, the learned phase-type distributions, and the discrete distributions from $\boldsymbol{\theta}$.

This is achievable because the mixed intensity parameterization is described using sojourn times and discrete probability distributions over destination states, which work well with the phase-type embedding process. A pure intensity representation does not readily describe the discrete distributions required for embedding a learned phase-type distribution. Furthermore, the racing exponential interpretation loses meaning when considering non-exponential distributions, while the sojourn time interpretation promoted by the mixed intensity representation works well with new parametric distributions.

3.7 Summary

This chapter was dedicated to developing a method for approximating known parametric distributions using phase-type distributions. We demonstrated that this can be accomplished by using an optimization algorithm to minimize a modified KL-divergence value. The PSO, GA, and simulated annealing algorithms were compared, and experimental results showed that PSO produced better approximations. We explored how well this procedure performs for a variety of parameterizations of both Weibull and Lognormal distributions and also tested how the number of phases impacts the quality of the approximation. We also proposed and tested an extension that uses informed initialization to improve convergence speed of the optimization algorithm. Finally, the embedding process for phase-type distributions was presented in a more formal context, and we made the argument for a mixed intensity representation for cases involving these phase-type embeddings.

The techniques in this chapter provide a methodology for encoding approximate non-exponential parametric distributions using CTBNs, thereby paving the way for additional CTBN applications in domains exhibiting transition events that are known to follow more complex distributions. This is certainly true of the reliability domain, where system failures often follow known distributions like Weibull and Lognormal. The formalization of the embedding process in this chapter also marks a valuable milestone in improved CTBN adoption, in that it provides a fully general description of how a learned phase-type distribution can be embedded into a CIM. Using this formalization in addition to an algorithm for minimizing D_{KL}, it is possible to directly specify the parameters for a non-exponential distribution and obtain a CTBN representation in an automated fashion. We expect these contributions to reduce the level of difficulty associated with building CTBNs for practical applications.

CHAPTER FOUR

CONTINUOUS TIME DECISION NETWORKS

In the last chapter, methods for extending the CTBN framework to support non-exponential parametric distributions were discussed. The intent was to extend the class of problems that can be represented with CTBNs. In this chapter, we continue toward this goal by introducing a variation on the CTBN framework that allows for the representation of decision problems. This new model, which we refer to as a Continuous Time Decision Network (CTDN), is designed to support optimization over a set of actions. This further extends the class of problems to which CTBNs may be applied to include continuous time decision problems.

4.1 Background

This section provides a review of related topics in the literature upon which this work is founded. First, the decision network model is presented as it exists in a static context. Factored performance functions are discussed, which are used as the basis of the utility nodes in CTDNs. Finally, a review of the literature related to optimization and decision making in CTBNs is provided.

4.1.1 Decision Networks

In its natural form, BNs are capable of representing static probability distributions over a set of variables. A decision network is a BN that has been adapted to represent and allow for reasoning over decision problems as well [115, 116]. Decision networks have also gone by the name decision diagram or influence diagram. Since its introduction, the decision network has received substantial focus in the literature for applications in domains requiring decision modeling, and often provide a tractable alternative to decision trees, which can grow exponentially with the number of variables [60].

Decision networks make use of three distinct types of nodes. The first is the *chance node*, also known as an uncertainty node, which is identical to the nodes used in a standard Bayesian Network that describe the conditional probability distribution over a variable in the system. The second is the *decision node*, or action node, which represents potential decisions to be made. These nodes can be thought of as a special kind of uncertainty node where the state is chosen ahead of time and assigned as evidence, rather than being inferred algorithmically. As a result, decision nodes do not have parents by construction. Finally, decision networks make use of the *utility node*, also sometimes referred to as a value node. Utility nodes assign a numeric value to each state combination of its parents, rather than defining a probability distribution over a set of states. Conventionally, chance nodes are depicted as ovals, decision nodes are rectangles, and utility nodes are diamonds in a rendered graph.

An example decision network is shown in Figure 4.1. This network models the oil wildcatter drilling decision problem [108]. In this problem, a wildcatter must decide which tests to perform, and whether or not to drill. The problem depends on several uncertain factors, including the test results, seismic structure, amount of oil, and the cost to drill. The decision should maximize profit, which depends on a number of factors, including the test and drill decisions. By considering the possible decisions and applying evidence, standard Bayesian network inference can be used to obtain utilities, and solving the problem therefore amounts to choosing the decision that produces the best expected utility. This example demonstrates how decision diagrams can be used to model complex interactions, which can in turn be used to identify the best course of action when faced with uncertainty. These models have

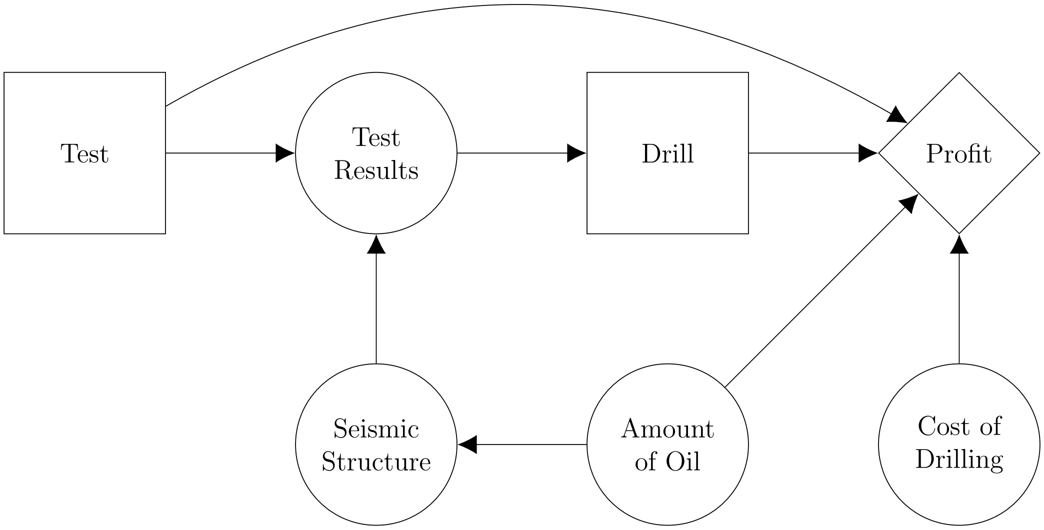


Figure 4.1: Oil wildcatter drilling decision problem

since been extended to allow for consideration of multiple utilities, as well as multiple decision agents [60, 62].

4.1.2 Factored Performance Functions

A performance function measures user-specified cost/reward values over the behavior of a system. For CTBNs, this can be achieved by evaluating a sample σ using a function f. The resulting value $f(\sigma)$ represents the global performance of a system. The concern is that defining a function in terms of a sample σ may be intractable, given that the number of possible states in σ is exponential in the number of variables. This makes defining and even computing a function impractical for most applications.

To address this issue, factored performance functions were proposed as a method for representing a performance function while still adhering to the factored representation of the CTBN [132]. This is applicable for cases where the global function exhibits local decomposition in terms of each variable. To achieve this, local performance functions are associated with nodes in the network that are involved in the global performance function. More formally, if f is a global performance function, it is factored over a subset of nodes $\mathbf{X}' \in \mathbf{X}$, resulting in local functions f_X, one for each $X \in \mathbf{X}'$. Furthermore, each function f_X is factored with respect to time as well, avoiding the need to specify a function over every possible trajectory of a variable. Instead, each local function defines a performance value assigned to a single observation of a single variable. Let $\sigma[X]$ be the portion of the trajectory describing transitions of X, and let $f_X(t_s, t_e, X(t))$ be a function over a single state observation in $\sigma[X]$ from time $t = [t_s, t_e)$. The global performance function over a full CTBN is therefore factored as:

$$f(\sigma) = \sum_{X \in \mathbf{X}} \left(\sum_{\langle t_s, t_e, X(t_s) \rangle \in \sigma[X]} f_X(t_s, t_e, X(t_s)) \right).$$

For a more formal definition, see Sturlaugson and Sheppard [132]. The factored performance function f_X is able to represent both fixed and variable costs over the states of X. As an example, consider a performance function for a node X with states x_0 and x_1, and let $\Delta t = t_e - t_s$. A local performance function over X is defined as follows:

$$f_X(t_s, t_e, X(t_s)) = \begin{cases} c_1 + c_2 \Delta t & \text{if } X(t_s) = x_0 \\ 0 & \text{if } X(t_s) = x_1 \end{cases}.$$

Here, the time unit is hours, and c_1 and c_2 are two constants representing dollars and dollars per hour respectively. The function f_X indicates that a fixed cost of c_1 occurs every time X enters state x_0, and will continue to receive a cost of c_2 for every hour that it remains in this state. When X is in state x_0, no cost is assigned. This function is assigned over the states of variable X only and does not depend on the location in the trajectory. In other words, the value of t_s is irrelevant, only the relative difference between variables t_e and t_s is necessary to compute the function.

Once concern with the factored approach is that there are some global performance functions that cannot be entirely factored into local functions f_X.

Definition 4.1.1 (Performance Synergy)**.** *Performance synergy is the case in which, for at least two nodes X and Y, with joint performance function $f_{\{X,Y\}}$, there do not exist functions f_X and f_Y such that, for all $x \in X$ and $y \in Y$,*

$$f_{\{X,Y\}}(\{x, y\}) = f_X(x) + f_Y(y).$$

For instance, let c_a be the cost of purchasing some item A, and let c_b be the cost of buying item B. A discount may be applied such that the cost of buying both items is less than the sum of each event: $c_{ab} < c_a + c_b$. To account for this, synergy nodes have been introduced as a means of compensating for performance synergy. These nodes work by augmenting the CTBN with an additional deterministically parameterized node with parents corresponding to the variable exhibiting synergy. Any addition or subtraction of value associated with a state combination of the variables is then accounted for using this new node.

Note that a network is not restricted to a single performance function f. A family of performance functions $\mathcal{F} = \{f^1, f^2, \ldots, f^m\}$ may be defined for a single CTBN. Each performance function gives one "view" of the network. $\mathcal{F}$ could be constructed to represent several potentially competing metrics, such as the cost and quality of a product. Furthermore, it is possible to evaluate $\mathcal{F}$ with a single set of samples $\mathcal{S}$, regardless of the size of $\mathcal{F}$.

4.1.3 Related Work

Cao's work is most similar to our own in that it aims to perform multi-objective optimization using CTBN inference as the basis for metric evaluation [13]. Despite similarities, Cao's approach is more application oriented, while the ideas presented in this chapter are focused more on providing a general framework for making decisions in continuous time. In their work, metrics are encoded directly into the fitness function for a GA. By using performance functions, we are able to take advantage of the factored representation when evaluating metrics. Furthermore, metrics are inherently built into the model itself, and computation of fitness is handled implicitly within the CTBN framework using inference, rather than relying on a manually defined evaluation function external to the model. Furthermore, Cao's work defines a unique CTBN corresponding to each decision, while our approach encodes decisions directly into the model itself. Finally, our work does not require that decisions be represented as static events, but instead allow for decisions that are applied at specific time intervals. In this way, the CTDN presented in this chapter can be thought of as a generalized framework for the manually defined optimization problem conducted by Cao.

4.2 Continuous Time Decision Networks

As with traditional decision networks, a CTDN is comprised of three different types of nodes: chance nodes, decision nodes, and utility nodes. Analogous to BNs, the chance nodes in a CTDN are identical to the nodes found in a CTBN. That is to say, a chance node defines a Conditional Markov Process for a variable, which describes the distribution of the variable as a function of continuous time conditioned on the state of its parents. This leaves utility nodes and decision nodes that have yet to be defined within this framework.

A utility node in a BN assigns a numeric value to each state instantiation of its parents. Although this could be applied directly to the CTBN framework, this assumes a static distribution, which would require a fixed time point for evaluation and fails to capture the temporal aspect of the model. Instead, a utility should be defined in a way that incorporates the time spent in each state over a period of interest. Defining this type of utility can be expensive, especially if the function depends on many variables in the network. To address this, we employ the families of factored performance functions presented by Sturlaugson and Sheppard [132]. In this way, each utility node in a static decision network is represented in continuous time as a performance function f^i in a family of functions $\mathcal{F}$. Note that unlike a utility node in a decision network, the utility function $f^i \in \mathcal{F}$ of a CTDN may be modeled using more than one node in the graph, and may indeed be built directly into an existing chance node. For visual clarity, these factored functions can be expanded out into their own utility function nodes using the same deterministic strategy employed when handling synergy nodes in factored performance functions.

Finally, there is the need for a decision node to represent decisions in continuous time.

Definition 4.2.1 (Continuous Time Decision Node)**.** *A* continuous time decision node *X is a special type of CTBN node that has no parents, and where the state of the process is known at all times $t = [0.0, \infty)$, thereby defining a local trajectory over the variable $\sigma[X]$. The states in the trajectory $\sigma[X]$ must conform to a possibly empty set of constraints $\mathbf{C}$, defining valid decisions for the system. Each constraint $c \in \mathbf{C}$ consists of a set of tuples $(t_s, t_e, \mathbf{Y})$ defining the set of possible states $\mathbf{Y}$ that may be assigned over the time interval $[t_s, t_e)$.*

A continuous time decision node is identical to a static decision node, except that its state must now be assigned as a trajectory over time. It is certainly possible to represent static decisions, which is achieved by assigning a single state $x_i \in Val(X)$ to a decision node X for the entirety of time $t = [0, \infty)$. The state of a decision node is defined ahead of time, so the node's behavior does not depend on anything in the model itself, and therefore has no parents. Furthermore, any transition behavior is specified explicitly, meaning that the choice of an initial distribution and transition parameters is irrelevant, so long as it allows each state to be reached at any point in time. For simplicity, and to provide good mixing during sampling procedures, each parameter can be set so they are equal to $(1/n)$, where n is the number of states in the decision node. The defined states for any decision node are applied as evidence, and inference is used to fill in the remaining unknown portions of the process' trajectory.

4.2.1 Drug Effect Decision Network

This section presents a revised version of the drug effect network described in Chapter 2. This new model applies slight variations to the original dependence structure and adds decision and utility nodes to transform the CTBN into a CTDN capable of representing decisions.

Example 4.2.1. *Drug Effect Decision Network*

Figure 4.2 shows an example of a Continuous Time Decision Network. This model is a modified version of the original drug effect network, recast as a decision problem. The network describes the temporal behavior of a patient's condition as influenced by concentration of a drug in their system. The decision problem requires determining how to apply the drug in order to maximize comfort and minimize weight gain.

In the example, the model still contains chance nodes for a Uptake, Concentration, Drowsy, Pain, Barometer, Full Stomach, Eating, and Hungry. These are aspects of the patient system that, in general, cannot be controlled directly. One aspect that can be modified indirectly is the uptake of the drug, which can be modified with an application of the drug itself. The CTDN version of the model therefore incorporates a new decision node Drug, assigned as a parent to the existing node for Uptake. In the original model, Uptake transitions from state u_1 to state u_0 with an intensity of 0.5 and remains in state u_0 for the remainder of the process. With the addition of the two-state Drug node, this behavior remains true when Drug is in state d_0, but Uptake transitions to u_1 again with an intensity of ∞ when Drug is in state d_1. Furthermore, the initial distribution for Uptake is adjusted so that the variable deterministically starts in u_0. Conceptually this means that uptake will not occur unless the drug is applied, at which point it occurs instantaneously. After the drug is not longer applied (Drug returns to d_0), the uptake stops after a time that is distributed exponentially with a rate of 0.5. This behavior models the application of the drug directly, rather than assuming the event occurred at some previous time. Since the application of the drug is a decision, the Drug node is drawn as a rectangle rather than the oval shape used for chance nodes.

In addition to the Drug node, the model is also modified to account for the effect that a drug may have on hunger. Some drugs affect appetite, and to capture this behavior, an edge was added from the three-state Concentration node to the two-state Hungry node in the original network. To account for the drug's effect on appetite, the first rows of the Hungry CIMs are multiplied by a factor of ϕ_0, ϕ_1, or ϕ_2, depending on if Concentration is in state c_0, c_1, or c_2, respectively. Similarly, the second rows of the Hungry CIMs are multiplied by a factor of $\frac{1}{\phi_0}$, $\frac{1}{\phi_1}$, or $\frac{1}{\phi_2}$, depending on the state of Concentration. We set $\phi_0 = 1.0$, so that when Concentration is

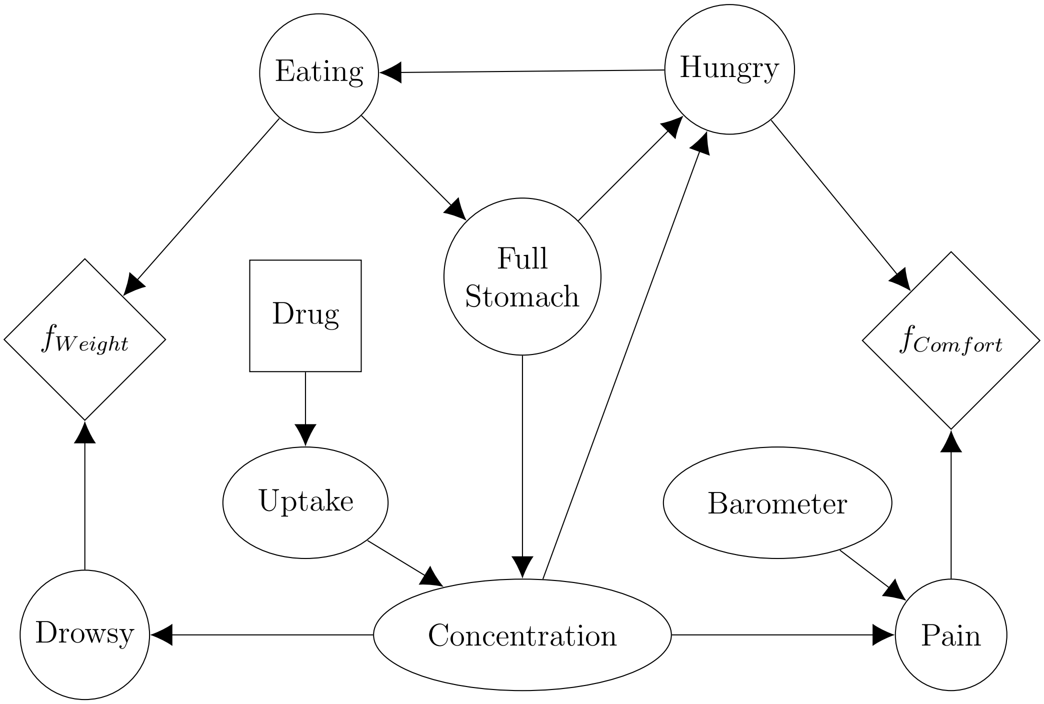


Figure 4.2: Example drug effect decision network.

in state c_0, `Hungry` behaves the same as in the original drug effect network. We set $\phi_1 = 5.0$ and $\phi_2 = 10.0$, so that as the concentration of the drug increases, the intensity for transitioning into the h_1 state increases while transitioning back to state h_0 decreases. This effectively models the scenario where the drug causes the patient to feel hungrier. By setting ϕ_1 and ϕ_2 to some constant $c < 1.0$, it would have also been possible to model loss of appetite as a side effect.

This decision problem is concerned with both a patient's comfort level and their potential for weight gain due to the application of the drug. To model these factors of interest, two performance functions are developed to encode these utilities. First, a function f_{Comfort} is defined that describes patient comfort in terms of the pain experienced by the patient and how hungry the patient is. In an attempt to capture the instant gratification received by a patient, benefits are awarded when either `Pain` or `Hungry` transition back to a state of p_0 or h_0. In addition, a constant benefit is also awarded for every hour the patient remains in either the p_0 or h_0 state. In a similar fashion, penalties are assigned whenever a transition is made into the states p_1 or h_1, and a constant negative contribution to the function is dispensed for every hour spent in either one of these states. This function describing comfort is captured by the newly introduced f_{Comfort} utility node, rendered as a diamond shape and added as a child of the existing `Hungry` and `Pain` nodes.

The drug's potential to induce weight gain for a patient is described by the f_{Weight} performance function. This function is defined in terms of the `Eating` and `Drowsy` nodes, with the notion being that the calorie intake will be higher if the patient eats more, and the calories burned will be lower when the patient is drowsy for longer periods of time. While immediate pleasure or displeasure was attained for the comfort variable upon entering a state, the concept of weight gain has no corresponding instantaneous behavior. For example, while *being* drowsy may cause weight gain, *becoming* drowsy does not. For this reason, f_{Weight} is a more simplistic performance function that simply adds a positive contribution for every hour spent in the Eating $= e_1$ or Drowsy $= d_1$ states. This function is encoded by the f_{Weight} in Figure 4.2. The introduction of these two utility nodes, along with the `Drug` decision node, transform what was a continuous time uncertainty problem into a continuous time decision problem.

4.2.2 Multi-Objective Optimization using CTDNs

The CTDN provides a framework for solving decision problems in continuous time. Solving the decision problem reduces to finding a decision or set of decisions that maximize the defined utility functions. In other words, the goal is to optimize the utility function outputs over the possible inputs as represented by the decision nodes. This can be achieved by using the CTDN framework in conjunction with standard CTBN inference algorithms.

Let $\mathbf{X}$ be the set of nodes in a CTDN, and let $\mathbf{D} \subset \mathbf{X}$ be the set of decision nodes in the model. A complete decision $\sigma[\mathbf{d}]$ is a complete trajectory for each variable in the set $\mathbf{D}$ and represents an external action that can be taken. Next, let f be a performance function defined over the CTDN using utility nodes. Inference can be used to compute the expected utility conditioned on the decision $\sigma[\mathbf{d}]$, which is assigned as evidence in the network. This expected utility $E(f|\sigma[\mathbf{d}])$ is the value associated with choosing that specific action, incorporating the uncertainty defined by the model. Let $\mathbf{D}_C$ be the set of all trajectories over the decision set $\mathbf{D}$ subject to constraints $\mathbf{C}$. The CTDN optimization problem attempts to find an assignment to the decision set that maximizes the expected value:

$$\operatorname*{argmax}_{\sigma[\mathbf{d}]\in\mathbf{D}_C} E(f|\sigma[\mathbf{d}]).$$

For multi-objective optimization, optimization is performed over a family of m user-defined performance functions $\mathcal{F} = \{f^1, \ldots, f^m\}$,

$$\operatorname*{argmax}_{\sigma[\mathbf{d}]\in\mathbf{D}_C} \left(E(f^1|\sigma[\mathbf{d}]), E(f^2|\sigma[\mathbf{d}]), \ldots, E(f^m|\sigma[\mathbf{d}])\right).$$

As is often the case with multi-objective optimization, when all of the functions of $\mathcal{F}$ cannot be combined into a single value that can be optimized, the task is then to estimate the Pareto frontier [70]. In this context, the Pareto frontier is the set of solutions where an improvement for any value of a performance function f^i implies a degradation in the value of some other performance function f_j. More formally, let the set of solutions $\mathbf{Z} \subseteq \mathbf{X}$ be the Pareto optimal set. Then the remaining set of solutions $\mathbf{Y} = \mathbf{X} \backslash \mathbf{Z}$ is the dominated set, where each solution $Y \in \mathbf{Y}$ is strictly worse with respect to all dimensions that a solution $Z \in \mathbf{Z}$.

The constraints on $\mathbf{C}$ are used to define the *allowable* trajectories. The constraints are necessary to prevent infeasible actions, but the constraints can also be used to make the optimization problem more tractable. The action trajectories are defined over continuous time, so the user may specify discrete time intervals over which the action holds. Optimization can then maximize the expected utility functions by enumerating these discrete choices rather than optimizing over the entire continuum.

4.3 Experiments

This section provides a demonstration of multi-objective optimization using CTDNs. The Drug Optimization experiment in section 4.3.1 presents the results of an optimization problem using the network presented in Example 4.2.1. This problem involves identifying the number of doses taken by a patient per day to find an amount that maximizes comfort level while minimizing weight gain. The Vehicle Fleet Optimization experiment in section 4.3.2 also defines two utility functions; here the goal is to assign an optimal set of technicians to maximize vehicle uptime and minimize cost.

4.3.1 Drug Optimization

This experiment makes use of the CTDN from Figure 4.2. The goal of this multi-objective optimization problem is to find a trajectory for the `Drug` node that maximizes comfort while minimizing weight gain. First, the possible decisions associated with the decision variable `Drug` are constrained to eliminate behavior that is not possible or does not align with the model. In this case, application of the drug is intended to be an event that occurs at a specific time. This can be simulated by ensuring that upon transitioning to state d_1, the variable transitions back to state d_0 after a short expected time interval ϵ. For this experiment, ϵ is set to 0.001. When the drug is applied (`Drug` enters state d_1), `Uptake` instantly transitions to state u_1 due to an infinite transition intensity forcing uptake to match the drug application. When `Drug` transitions back to state d_0 in roughly ϵ time later, `Uptake` begins its normal behavior of transitioning back to u_0 with rate 0.5.

While the described constraint ensures only valid trajectories are considered, there is still the intractable problem of evaluating all possible times at which to transition into state d_1. To overcome this problem, domain knowledge can be applied to constrain the trajectories further to a set of feasible options. In this case, it is known that the drug should be administered at routine intervals. This constraint reduces the problem of evaluating all possible transition times to the more tractable problem of identifying the best time between drug dose applications.

This decision problem is focused on the effect of drug applications administered throughout a day, and therefore inference is run from time $t_s = 0$ to time $t_e =$ 12 hours. Identifying the optimal time between dose applications is equivalent to the problem of identifying the number of doses administered per day. We begin by evaluating the case where no doses are applied, which equates to a time between doses greater than 12. We then increase the doses per day from 1 to 8, applying evidence to the `Drug` node that forces a transition to and from state d_1 at routine intervals throughout the day. At 8 doses per day, this equates to an application event occurring every 1.5 hours.

The computed values for f_{Comfort} and f_{Weight} as a function of the number of applied doses are shown in Figure 4.3. The comfort level for the patient increases as the number of doses increases, but the payoff levels off after two or three applications

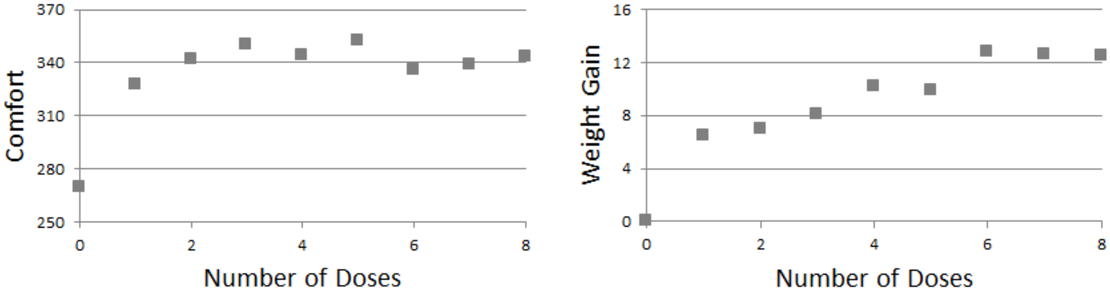


Figure 4.3: Performance estimates of Comfort (left) and WeightGain (right) as the number of doses is increased.

of the drug. Any number of doses above three remains at a comfort level of approximately 340. The same positive increase can be observed for weight gain, but the trend continues until hitting the maximum of eight doses. Optimizing either function independently is a simple matter of choosing the largest or smallest value. The task of optimizing both functions simultaneously is more difficult.

To identify the optimal choice for both performance functions, Figure 4.4 plots weight gain as a function of comfort. The goal is to maximize comfort and minimize weight gain, so an ideal datapoint is located in the bottom right of the graph. Here it can be seen that zero drug applications minimize the expected weight gain, while five drug applications maximizes Comfort. The darker nodes make up the Pareto frontier, which strictly dominate the performance of the lighter nodes. The dominated nodes occur due to the ceiling for the comfort values that occurs at around four doses, which is not shared by the weight gain performance function. Any of the darker nodes in the Pareto frontier are a valid choice for the optimal drug applications depending on how much comfort is valued as compared to weight gain. In this case, two or three doses appear to provide a good balance between comfort and weight gain.

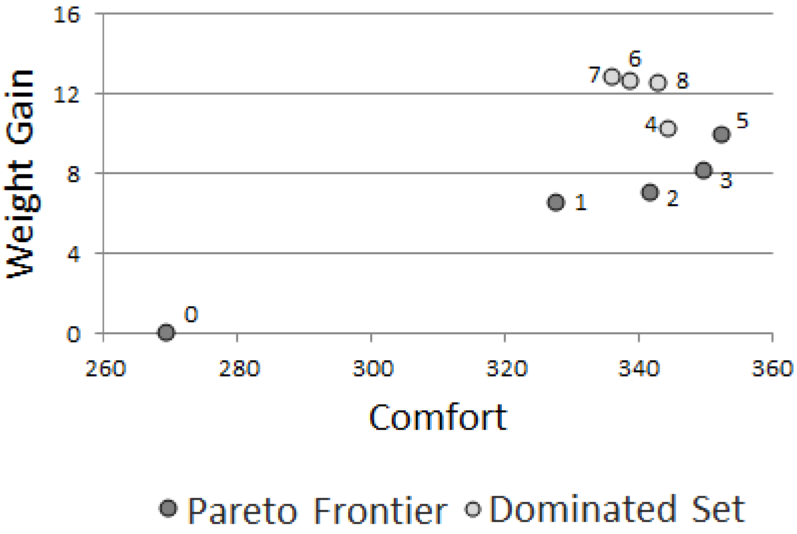


Figure 4.4: Performance estimates of WeightGain displayed as a function of Comfort. Each datapoint is labeled with the number of dose applications.

4.3.2 Vehicle Fleet Optimization

This next experiment is based on a real-world reliability model originally presented by Cao, describing the expected uptime for a military vehicle [13]. This system consists of three major subsystems: chassis (CH), powertrain (PT), and electrical (EL). The chassis is comprised of four components, each having their own failure and repair rates: suspension (SU), brakes (BR), wheels and tires (WT), and axles (AX). Likewise, the powertrain subsystem is comprised of three subsystems: cooling (CO), engine (EG), and transmission (TR).

In this experiment, the model was adapted to cover a fleet of vehicles and has also been adapted to convert the CTBN into a CTDN capable of representing a decision problem. The resulting model is shown in Figure 4.5. Here, the fleet size is set to five, meaning that there are five separate replicated models of the vehicle network. Each vehicle model can incorporate its own evidence, e.g., repair and usage history.

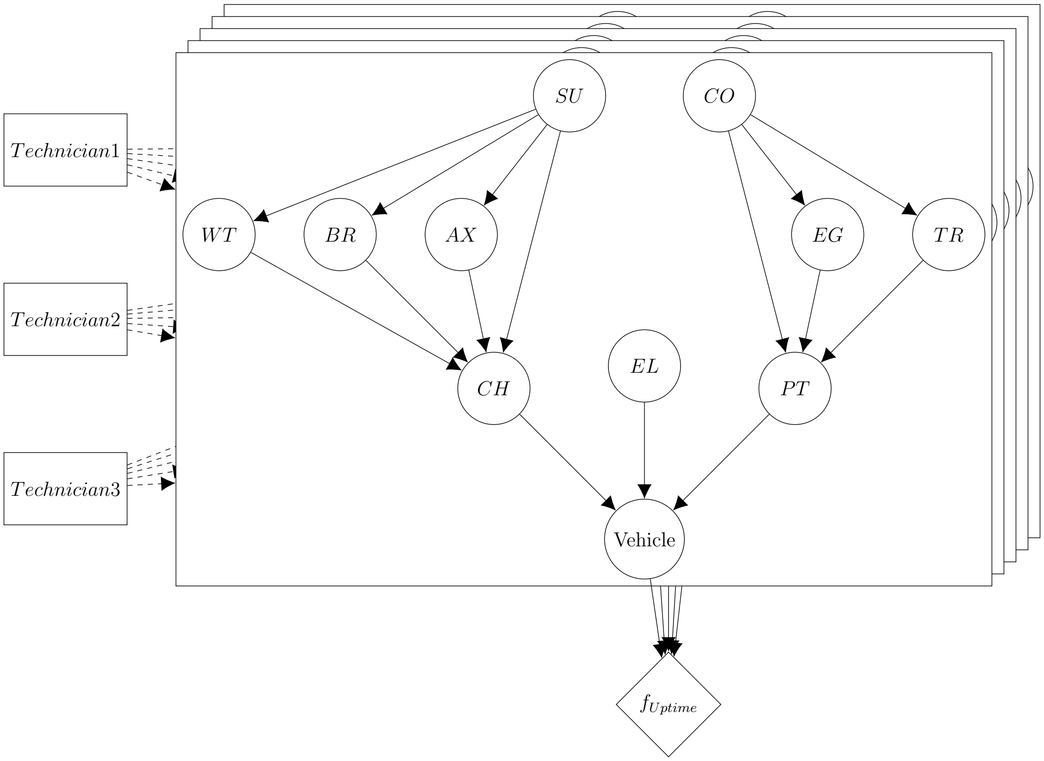


Figure 4.5: Vehicle decision network.

Decision nodes are added that represent the option to assign specialized technicians who are capable of maintaining and repairing the vehicles in the fleet. We include three technician nodes in the network, which are added to the parent set of all remaining nodes in the network. This means that the behavior of all components in each vehicle is affected by the behavior of the technician nodes. The intent is for these to provide added expertise in maintaining the vehicles beyond that already covered by the model.

The technician nodes have two states corresponding to the absence or presence of the technician. The presence of a technician decreases the time to repair and increases the expected time to failure for a component due to preventative maintenance. This equates to increasing the intensities for transitioning back to state 0, while decreasing the intensities for transitioning away from state 0. Technician 1 and Technician 2 are considered to be Level I technicians, which scale the repair rates by a factor of 1.5 and the failure rates by a factor of 0.8. Technician 3 is intended to model a Level II technician, which has a stronger influence on the repair and failure rates. In this case, the repair rate is increased by a factor of 2.0 and the failure rate is reduced by a factor of 0.5.

Performance is defined in terms of the uptime for each vehicle, encoded using the utility node f_{Uptime}, which is attached as a child of each `Vehicle` node in each of the networks. A value of 1.0 is assigned for every hour a vehicle is in an operation state, and a double value of 2.0 for each vehicle is assigned for instances where all vehicles are operational at the same time. The function is represented using a single utility node rather than as separate factored utility nodes over each vehicle, since it is expected that the value of the entire fleet of vehicles is worth more than the sum of the individual uptime values.

Table 4.1: Repair Costs for Vehicle Components

Component	**Repair Cost**
BR (Brakes)	950
WT (Wheels/Tires)	700
AX (Axles)	2000
SU (Suspension)	850
EG (Engine)	450
TR (Transmission)	6500
CO (Cooling)	200

Furthermore, another performance function f_{Cost} is defined to account for the monetary cost required to maintain the fleet of vehicles. There are two sources of cost that are modeled in this system. The first is the cost of a repair event. A unique cost is associated with the repair event for each component in the vehicles. No cost is associated with repair events for entire subsystems, as the individual components account for the cost required to bring the subsystem back online. The specific costs associated with each component are shown in Table 4.1.

The second contributor to the f_{Cost} performance function is the wages paid to the technicians. This cost is measured in terms of how much money is spent per hour retaining the technician to work on the vehicle fleet. Technicians are also assigned other billable jobs as well; therefore, the hourly cost for each technician with respect to this fleet of vehicles is only a portion of the entire wage for the technician. The cost per hour to retain Technician 1 and 2 is 2.0 respectively, while the cost to maintain Technician 3 is 3.25. The discrepancy between wages is due to the difference is skill levels, where Technicians 1 and 2 are at skill level I, and Technician 3 is at skill level II.

Note that the utility nodes associated with cost are not shown explicitly in Figure 4.5. Technically, a single f_{Cost} utility node could be added as a child of the seven component nodes from Table 4.1 and the three technician nodes. With 10 binary parents, this would mean the utility node would require $2^{10} = 1024$ deterministic intensity matrices. Instead, the function can be factored over each of its parents individually, since the function f_{Cost} exhibits no synergy. This could be achieved explicitly using ten different utility nodes, each with one parent. For visual clarity, the model in Figure 4.5 does not show each of these utility nodes, but instead encodes the function directly using the seven chance nodes and three decision nodes.

The model now contains two performance functions: f_{Uptime}, which are dependent on the amount of time each vehicle spends in the operational state, and f_{Cost}, which is defined in terms of repair costs and technician wages. Here again we are faced with a multi-objective optimization problem where the goal is to maximize vehicle uptime while minimizing cost, and the decision variables are the use of technicians. Formally we wish to identify a trajectory for the three technician nodes that will optimize the expected values for the performance functions. We start by constraining the search space. In our system, we either hire a technician or not, meaning that the state of the technician is static and does not change throughout the entire time of interest. This reduces the problem to that of identifying the initial states for each technician node.

To find the optimal assignment of technicians, we set evidence that each technician node is either zero or one for the entire time period and evaluate the vehicle uptime and cost performance functions. We do this for every assignment of technician nodes, resulting in $2^3 = 8$ datapoints. Figure 4.6 shows each assignment's cost plotted against the vehicle's performance. The datapoints are labeled as (t_1, t_2, t_3), where t_i is the state assignment for Technician i. The goal is to minimize cost and maximize

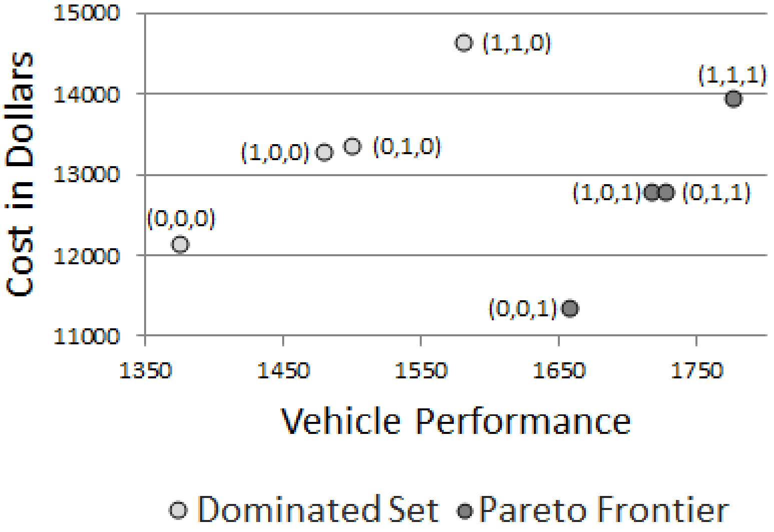


Figure 4.6: Performance estimates of Cost displayed as a function of VehiclePerformance. Each datapoint is labeled with the assignment of technicians in the form (Technician1, Technician2, Technician3).

vehicle performance, so here again the best solution can be found in the lower right side of the graph. The Pareto frontier consists of the darker points, which dominate the remaining lighter colored nodes. Depending on the importance placed on the vehicle's performance vs cost, any of the assignments in the Pareto frontier are a valid choice for the optimal solution.

This experiment demonstrates how CTDNs can be used to solve complex optimization problems. Intuitively it may seem that as more technicians are employed, vehicle performance will increase along with cost. While this is true for vehicle performance, we see that the assignment with the minimal cost is not $(0, 0, 0)$, but rather $(0, 0, 1)$, which corresponds to assignment of Technician 3. This occurs because Technician 3 ultimately reduces the number of necessary repair events, each of which has a cost associated with it. The amount of money saved in repair events is greater than the amount spent on wages, and in the end Technician 3 actually saves money in the model. At first glance, it may seem tempting to spend less money on a Level I technician, but this model shows that there is absolutely no benefit to either cost or vehicle performance in hiring a Level I technician if a Level II technician is available. If vehicle performance is absolutely critical and money is less of a concern, hiring a Level I technicians to supplement a Level II technician can provide some increased performance. The interactions between technician skill, wages, vehicle performance, and repair cost are complex and not immediately obvious. Through the use of CTDNs, it is possible to solve this multi-objective optimization problem, which allows for more informed decision making.

4.4 Summary

In this chapter, we introduced the Continuous Time Decision Network; a variation on the CTBN capable of modeling decision problems in continuous time. The CTDN uses traditional CTBN nodes in addition to decision and utility nodes to represent possible actions, rewards, penalties, and uncertainty. The newly introduced node types are specializations of the nodes in a CTBN, allowing for evaluation of decisions using standard inference algorithms designed to support the CTBN. The utilities defined within the CTDN framework can be optimized as a function of the possible decisions, providing a means of solving a decision problem in continuous time. We demonstrated the capabilities of this framework by optimizing two decision problems adapted from models presented in the literature, showing that a set of optimal decisions can be determined even in the presence of uncertainty.

The introduction of the CTDN framework allows for the representation of decision problems in continuous time using the standard CTBN framework. This supports applications of CTBNs in areas requiring decision optimization with respect to some utility. The framework is a natural extension to decision networks using

Bayesian Network semantics, and provide a temporal alternative to the static representation. CTDNs provide a formal method for performing optimization in continuous time, and once again extend the class of problems that can be addressed using CTBNs.

CHAPTER FIVE

COMPACT REPRESENTATIONS

Chapters 3 and 4 both focused on the representational capabilities of the CTBN. Although this is certainly an important factor, even the most representationally powerful models can be impractical if they are unable to scale. In this chapter, we attempt to address this concern by introducing compact representations of conditional CTMPs. Specifically, mapped conditional intensity matrices and tree-structured conditional intensity matrices are presented as two alternative representations capable of encoding transition behavior more compactly than a traditional CIM. These representations have the potential to alleviate scalability issues, especially in the presence of structured data.

5.1 Background

In this section, motivation and research in the area of compact representations are discussed. In each case, the objective is to provide a more scalable solution to the problem of modeling systems of variables in the presence of uncertainty. Some of the discussed works use a similar approach to our own but are applied in a different domain. Others strive to achieve a similar goal of compact CTMP representation but employ different methods of doing so. In each case, we attempt to set our contributions in context. Note that our work in this chapter also makes use of basic principles of decision trees and function composition, but these principles are not themselves part of the novel contributions presented in this dissertation.

5.2 Motivation

By defining a conditional CTMP for each variable in the system, the size of the parameterized model can be reduced greatly. The number of entries in an intensity matrix for a CTMP is

$$n_{ctmp} = \left(\prod_{V \in \mathbf{V}} |V| \right)^2 .$$

Alternatively, the total number of entries in the CIMs of a CTBN over the same set of variables is

$$n_{ctbn} = \sum_{V \in \mathbf{V}} \left(\left(\prod_{U \in \mathbf{Pa}(V)} |U| \right) \times |V|^2 \right), \tag{5.1}$$

where $\mathbf{Pa}(V)$ is the set of parents for node V in the graph. If these parent sets are smaller than the total number of nodes in the graph, then $n_{ctbn} < n_{ctmp}$. In other words, so long as the behavior of a variable only depends on a subset of the other variables in the system, a CTBN is able to model the process more compactly.

Referring back to the drug effect network from Chapter 2, a Conditional Markov Process is specified for each of the eight nodes in the network. As an example, consider the `Joint Pain` node, which has parents `Barometer` and `Concentration`. For the sake of conciseness, these nodes can be referred to using their first initials J, B and C. Node J can be parameterized with an initial distribution of size $|J|$, and a CIM where the number of rows/columns is also $|J|$. In accordance with the definition for a Conditional Markov Process, the CIM can be viewed as a set of homogeneous intensity matrices, one for each unique state instantiation of the parent nodes. The CIM for node J will therefore consist of $|B| \times |C|$ intensity matrices. If it is assumed that $|B| = 3$ and $|C| = 2$, then there are $3 \times 2 = 6$ homogeneous intensity matrices that make up the CIM for node J.

$$\mathbf{Q}_{J|B,C} = \{\mathbf{Q}_{J|b0,c0}, \mathbf{Q}_{J|b0,c1}, \mathbf{Q}_{J|b1,c0}, \mathbf{Q}_{J|b1,c1}, \mathbf{Q}_{J|b2,c0}, \mathbf{Q}_{J|b2,c1}\}$$

Although the CTBN representation may be substantially more compact than a CTMP over the same set of variables, in many situations the model may still be unmanageably large. Note the product in Equation 5.1, which accounts for the requirement that all parent instantiations must be enumerated. If a node in a CTBN has a large number of parents, then storing and using a CIM for the node may be difficult due to the exponential number of homogeneous intensity matrices. In the worst case, all nodes may depend on every other node. In this case, no independencies can be exploited, and a CTBN provides no benefits over a CTMP representation. For CTBNs with large parent sets, a more concise representation may be required. This chapter introduces two such representations, both capable of reducing the complexity of specifying the parameters of a CTBN node when the number of parents is large.

5.2.1 Related Compact Representations

The CTBN is not the only framework designed to induce structure on a CTMP; there is also the Kronecker representation and the decision-diagram [118]. Kronecker algebra is able to decompose the intensity matrix of a CTMP into basic matrix operations [11]. These operations, known as Kronecker products and Kronecker sums, allow smaller matrices to be stored that can be used to reconstruct the original process after applying the Kronecker operations:

$$R = \sum_{e=1}^{E} \bigotimes_{l=1}^{L} R_e^{(l)}.$$

Here, R is a full joint rate matrix over L variables, $R_e^{(l)}$ is a rate matrix over variable l, and E can be viewed as the individual events in the process. Although the Kronecker representation strives toward a similar goal of compact representation, our approach starts with the already structured CTBN representation and seeks to add additional structure to the conditional CTMPs. Our approach is also distinctly different from the decomposition via matrix operations used in Kronecker algebra.

Decision diagrams are used to encode functions over discrete domains compactly. This same concept can be used to encode intensity matrices, providing another way to represent CTMPs compactly [43]. Decision diagrams are represented as directed acyclic graphs where each layer corresponds to a different variable. The outgoing edges for each node in a layer corresponds to values that the variable can take on. Traditionally decision diagrams are binary, but extensions have been made to allow for multiway decision diagrams (MDDs) [125]. The final value is determined by combining the values along the edges on a path from the root to a leaf node. This means that the behavior of a decision diagram largely depends on the operator that is used to combine edge values. Shelton and Ciardo discuss versions where the edge values are summed (EV$^+$MDD) and where the edge values are multiplied (EV*MDD) [118]. Here again, decision diagrams are used to add structure to a flat CTMP, while we look into further improving the CTBN framework.

The partition-based CTBN is another structured representation of a CTMP, and can be viewed as a generalization of the CTBN [141]. The concern being addressed here is that while complexity of unstructured CTMPs is unmanageable, there are assumptions made by the CTBN framework that may not suit certain data structures. Specifically, the conditional CTMP defined by a CTBN assumes that transition intensities only change as a function of the parent states. Weiss *et al.* lift this restriction by allowing multiple intensities to be specified for each parent combination, and instead map intensities to partitions defined over the full joint of the variables in the model. These partitions are represented using trees or forests, which bears a loose resemblance to the work presented in this chapter. The difference is that we use trees to encode a CIM more efficiently, rather than replace the CIM with a different representation entirely. Furthermore, while Weiss *et al.* are addressing an issue of representation, we are attempting to improve scalability. The partition-based CTBN is actually a *more* complex model in general, requiring more parameters as well as an additional partitioning structure that is specified in addition to the original CTMP. In contrast, our approach adds structure to the CTBN rather than replacing the existing structure, and generally requires less parameters.

Boutilier *et al.* introduce the concept of context-specific independence in BNs [10]. In their work, they formalize scenarios where a variable is independent of the state of a subset of its parents. This allows for conditional probability tables to be encoded using a decision tree, reducing the total number of parameters required to specify the BNs. The tree-structured CIMs presented in this chapter also make use of context-specific independence and are the continuous time analog of tree-structured conditional probability tables. As such, the work by Boutilier *et al.* serves as a static foundation to the techniques presented in Section 5.6. This chapter not only extends these methods to work in continuous time, but Section 5.5 also introduces an alternative method for compactly encoding CTBN parameters that is unrelated to tree structures.

5.3 Distance Metrics for Transition Distributions

Transitions between the states of a variable in a CTBN are described using exponential distributions. Let p_1 and p_2 be exponential distributions with rates λ and μ respectively. Furthermore, let $f(t)$ be the probability density function (PDF) for p_1, and let $F(t)$ be the cumulative distribution function (CDF) for p_1. Similarly, let $g(t)$ and $G(t)$ be the PDF and CDF for p_2. The distance between the two exponential distributions p_1 and p_2 can be quantified using a wide array of metrics, each of which has its own unique attributes and advantages [46]. In this dissertation, three common metrics used to compare probability distributions are considered. This section concludes by describing how these distance metrics may be used to compare intensity matrices.

5.3.1 Symmetric Kullback-Leibler Divergence

The Kullback-Leibler (KL) divergence, also referred to as relative entropy, measures the divergence of an approximating distribution from a target distribution [67]. The notation $D_{KL}(p_1||p_2)$ is used to indicate the KL divergence of distribution p_2 from p_1, where p_2 is the approximating distribution, and p_1 is the target distribution. The following shows the derivation for KL divergence for the special case where p_1 and p_2 are exponential distributions as described earlier in this section. Note that because p_1 and p_2 are positive distributions, the integral starts at zero.

$$\begin{aligned}
D_{KL}(p_1||p_2) &= \int_0^\infty f(t) \log\left(\frac{f(t)}{g(t)}\right) dt \\
&= \int_0^\infty \lambda e^{-\lambda t} \log\left(\frac{\lambda e^{-\lambda t}}{\mu e^{-\mu t}}\right) dt \\
&= \left(\lambda(\mu - \lambda) \int_0^\infty t e^{-\lambda t} dt\right) + \left(\lambda(\log \lambda - \log \mu) \int_0^\infty e^{-\lambda t} dt\right) \\
&= \left(\lambda(\mu - \lambda)(-\lambda^{-2} e^{-\lambda t}(\lambda t + 1))\right) + \left(\lambda(\log \lambda - \log \mu)(-\lambda^{-1} e^{-\lambda t})\right) \Big|_0^\infty \\
&= \frac{\mu}{\lambda} + \log(\lambda) - \log(\mu) - 1
\end{aligned}$$

Although useful for many applications, KL divergence is an asymmetric measure and is therefore unsuitable for a distance measure intended to test for equivalence between arbitrary distributions. There have been several different adaptations to KL divergence in the literature that extends the measure to enforce symmetry [9, 73]. Here, we use the definition that sums the standard KL divergence between the two probability distributions in both directions. This new metric is referred to as symmetric KL divergence, and is denoted $\widetilde{D}_{KL}(p_1, p_2)$.

$$\begin{aligned}\widetilde{D}_{KL}(p_1, p_2) &= D_{KL}(p_1||p_2) + D_{KL}(p_2||p_1)\\ &= \Big(\frac{\mu}{\lambda} + \log(\lambda) - \log(\mu) - 1\Big) + \Big(\frac{\lambda}{\mu} + \log(\mu) - \log(\lambda) - 1\Big)\\ &= \frac{\lambda^2 + \mu^2}{\lambda \cdot \mu} - 2\end{aligned}$$

5.3.2 Hellinger Distance

The Hellinger distance has been used successfully to quantify the distance between two probability distributions [26, 146]. Denoted $D_H(p_1, p_2)$, the Hellinger distance when considering exponential distributions is as follows.

$$\begin{aligned}D_H(p_1, p_2) &= \frac{1}{2}\int_0^\infty \left(\sqrt{f(t)} - \sqrt{g(t)}\right)^2 dt\\ &= 1 - \int_0^\infty \sqrt{f(t)g(t)}dt\\ &= 1 - \int_0^\infty \sqrt{\lambda e^{-\lambda t}\mu e^{-\mu t}}dt\\ &= 1 - \sqrt{\lambda\mu}\int_0^\infty e^{-(\lambda+\mu)t/2}dt\\ &= 1 - \sqrt{\lambda\mu}\left(-\frac{2}{\lambda+\mu}e^{-(\lambda+\mu)t/2}\right)\Big|_0^\infty\\ &= 1 - \frac{2\sqrt{\lambda\mu}}{\lambda+\mu}\end{aligned}$$

5.3.3 Kolmogorov Metric

The Kolmogorov metric, also referred to as the uniform metric, is another means of measuring differences in probability distributions [63, 153]. This metric indicates the largest deviation between the cumulative distribution functions and is denoted $D_K(p_1, p_2)$. The following definition shows the Kolmogorov metric when comparing exponential distributions:

$$\begin{aligned} D_K(p_1, p_2) &= \sup_t |F(t) - G(t)| \\ &= \sup_t |(1 - e^{-\lambda t}) - (1 - e^{-\mu t})| \\ &= \sup_t |e^{-\mu t} - e^{-\lambda t}| \end{aligned}$$

where sup_t is the supremum over the domain of t. To identify the maximum value, the terms within the absolute value function can be derived.

$$\frac{d}{dt} e^{-\mu t} - e^{-\lambda t} = \lambda e^{-\lambda t} - \mu e^{-\mu t}$$

By setting this equal to zero and solving for t, it is possible to determine the time at which the maximal difference occurs.

$$t = \frac{\log \lambda - \log \mu}{\lambda - \mu}$$

This value may then be substituted back into the original equation to obtain the supremum, and therefore the distances according to the Kolmogorov metric.

$$\begin{aligned} D_K(p_1, p_2) &= \left| e^{-\mu((\log\lambda - \log\mu)/(\lambda-\mu))} - e^{-\lambda((\log\lambda - \log\mu)/(\lambda-\mu))} \right| \\ &= \left| \left(\frac{\lambda}{\mu}\right)^{\mu/(\mu-\lambda)} - \left(\frac{\lambda}{\mu}\right)^{\lambda/(\mu-\lambda)} \right| \end{aligned}$$

5.3.4 Applying Distance Metrics to Intensity Matrices

To compare intensity matrices, the distance metrics described previously in this section can be leveraged. Each rate $q_{i,j}$ in an intensity matrix Q specifies a potentially unique exponential distribution p_1, which can be compared to the corresponding exponential distribution p_2 defined by rate $q'_{i,j}$ in a matrix Q'.

Let $D(q_{i,j}, q'_{i,j})$ be the distance between the exponential distributions defined by rates $q_{i,j} \in Q$ and $q'_{i,j} \in Q'$. Then let $D(Q, Q') = \sum_{i,j|i\neq j} D(q_{i,j}, q'_{i,j})$. Each of the distances $D(q_{i,j}, q'_{i,j})$ may be computed using one of the metrics discussed in this section. Note that the assumption of symmetry holds at the matrix level as well, such that $D(Q, Q') = D(Q', Q)$.

5.4 Hierarchical Clustering of Intensity Matrices

The basic idea behind compactly representing a CIM is that similar intensity matrices that comprise the CIM may be consolidated. The task then becomes how best to group intensity matrices so as to eliminate the largest number of intensity matrices while still retaining as much of the original semantics encoded by the network as possible. Although it would be possible to identify a score for a particular grouping of intensity matrices based on the distance metrics discussed in Section 5.3, it is infeasible to consider every possible grouping for all but the smallest of CIMs. Instead, clustering techniques are employed to manage the complexity of identifying a suitable grouping of intensity matrices.

Although there are a variety of clustering algorithms available, we make use of hierarchical clustering [79]. Hierarchical clustering is either performed using an agglomerative or divisive approach. Agglomerative clustering works by starting with each datapoint in a separate cluster and iteratively merging the clusters that are closest to one another. Conversely, divisive clustering initially places all datapoints into a single cluster and divides clusters into groups that are farthest away from one another. In this research, an agglomerative approach to clustering is used to identify groups of similar intensity matrices.

The clustering process depends on the ability to quantify the similarity between two arbitrary clusters. Cluster similarity can be achieved by making use of a linkage criterion that indicates distance between clusters based on pairwise distances between elements within the clusters. In this context, the elements within clusters are intensity matrices, and determining the distance between these intensity matrices can be achieved by using the metrics discussed in Section 5.3. While a variety of linkage criteria have been defined in the literature, we make use of maximal linkage clustering. The criterion works by computing the distance between sets of clustered intensity matrices as the maximum of the pairwise distances between the elements in the sets and is a common choice when the underlying structure of the data is unknown [137].

Consider a node with parent variables $\mathbf{U}$ having eight possible state instantiations. The corresponding CIM therefore consists of eight intensity matrices, which are denoted: $\{\mathbf{Q}_a, \mathbf{Q}_b, \mathbf{Q}_c, \mathbf{Q}_d, \mathbf{Q}_e, \mathbf{Q}_f, \mathbf{Q}_g, \mathbf{Q}_h\}$. Clusters of these matrices are denoted $C_{\mathbf{u}}$, where $\mathbf{u}$ indicates the matrices contained in the cluster. For example, C_{be} is a cluster containing the matrices $\mathbf{Q}_b$ and $\mathbf{Q}_e$. Figure 5.1 shows an example of how these matrices might be clustered hierarchically, using a dendrogram representation [**?**].

Here, the clustering starts at the bottom with each of the eight matrices assigned to individual clusters. Clusters are then combined in ascending order of distance, which is represented using the y-axis in the graph. For this example, it is determined that C_e and C_f are the closest to one another, and the two clusters are therefore combined into a single new cluster C_{ef}. This process is repeated continually until all clusters have been merged or until a stopping criterion is met.

Typically a stopping criterion either places a limit n on the depth of the tree or specifies a maximum distance threshold τ that determines whether or not to merge a pair of clusters. We make use of the distance threshold to determine which intensity matrices should be merged. Here, the distance threshold can be used as an approximation parameter, where smaller thresholds indicate more accurate approximations with smaller cluster sizes, and a distance threshold of zero will combine only intensity matrices that are exactly equal to one another. The choice of the maximum distance threshold depends on the desired approximation accuracy, as well as the underlying distance metric that is used to compare intensity matrices.

The dashed lines and nodes in Figure 5.1 represent potential cluster merges that are not taken in this example because their distances exceed a specified threshold τ, indicated by the horizontal line. The clusters on the frontier of the solid region in the hierarchy are C_a, C_b, C_{cd}, and C_{efgh}. Table 5.1 lists these four clusters in terms of the matrices they contain. Here, each row can be treated as a single intensity matrix, since each entry in a row is identical or nearly identical to the rest. For instance, C_4 consists of four intensity matrices that can all be treated as equivalent to one another. By taking the mean of these intensity matrices, a single matrix $\mathbf{Q}_4$ may be used to represent the entire cluster. As a result, even though there are eight intensity matrices in this example, the entire set can be represented with only four unique intensity matrices that are obtained by computing the mean of each cluster.

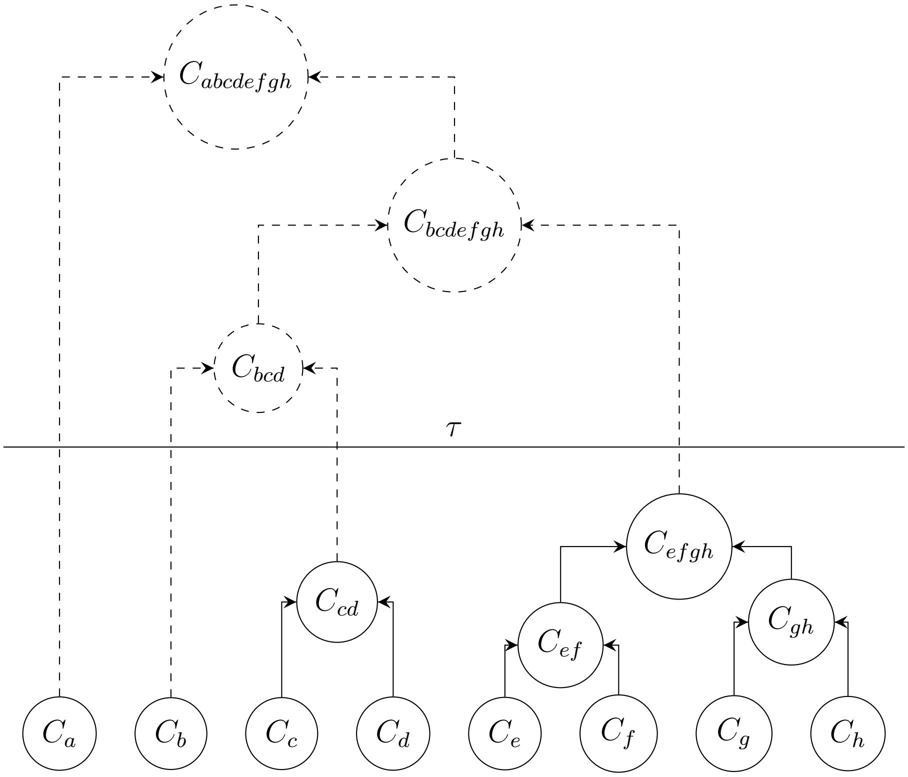


Figure 5.1: Intensity matrix clustering.

Table 5.1: Example Clustering of a CIM

C_1:	$\mathbf{Q}_a$			
C_2:	$\mathbf{Q}_b$			
C_3:	$\mathbf{Q}_c$	$\mathbf{Q}_d$		
C_4:	$\mathbf{Q}_e$	$\mathbf{Q}_f$	$\mathbf{Q}_g$	$\mathbf{Q}_h$

5.4.1 Clustering Experiments

In this section, an experiment is developed to compare the distance metrics derived in the previous section and determine their impact when clustering intensity matrices. This section focuses exclusively on the behavior of the clustering algorithm, and an investigation on the impact of the underlying CTBN dynamics is reserved for Sections 5.5.2 and 5.6.1. There are many ways of evaluating the quality of a clustering. Some evaluation techniques take a statistical approach, considering clusters with respect to the underlying data [50]. Another approach is to perform a relative comparison of the clusters, which is less expensive than performing statistical testing [51]. A traditional goal is to maximize inter-cluster distances while minimizing intra-cluster distances. For our purposes, we are interested in the total number of consolidated matrices; therefore, we treat the number of clusters as the variable of interest. For more specific applications where data is known to behave in a predictable fashion, distance metrics can be chosen based on more sophisticated criteria.

For this experiment, network structures were fixed to consist of five parent nodes and a single child node, each with two states. The networks are parameterized randomly, such that each of the off-diagonal rates in the intensity matrices are drawn from a uniform distribution $U(0.0, 1.0)$. Diagonal entries are defined as the negative sum of the remaining row, as per the definition of a Markov process. To obtain a reasonable sample base, 100 different networks were generated in this fashion. For each network, the hierarchical clustering algorithm was run using each distance metric, with the maximal difference threshold ranging from 0.0 to 2.0. The results are shown in Figure 5.2, where the total number of unique, unconsolidated matrices are shown on the y axis, and the maximal distance threshold is given by the x axis. The error bars show standard error for the 100 runs of the clustering algorithm.

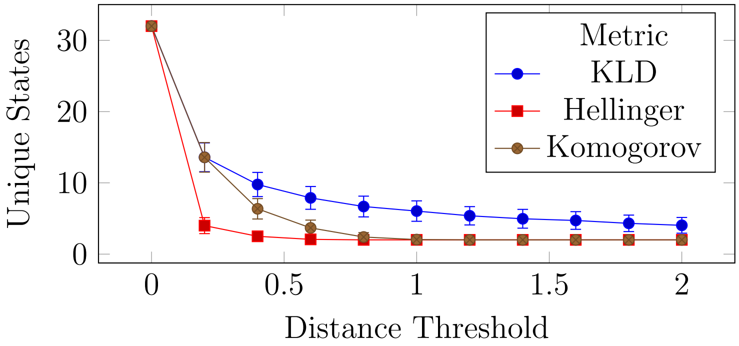


Figure 5.2: Consistent distance metric experiment.

The most important information that can be drawn from these results is the points at which each of the distance metrics tend to converge. For instance, the Hellinger metric appears to converge almost immediately, while KLD continues to decrease until nearly the end of the range. This provides an upper bound for a reasonable threshold value for each metric given the generated models. These ranges differ from metric to metric, meaning that the distance threshold should not necessarily be interpreted the same for each. As such, it does not necessarily make sense to compare each metric's impact on the clustering algorithm using a fixed range of 0.0 to 2.0. To account for this, the experiment is rerun with the same setup, except that the approximation threshold for Hellinger is now varied from 0.0 to 0.5, Kolmogorov ranges from 0.0 to 0.75, and the threshold for symmetric KL-divergence remains at the range 0.0 to 2.0. This normalizes each metric to a range that extends only to its observed convergence point. The plot for this modified version of the experiment is shown in Figure 5.3, where the x axis now signifies the percentage into the threshold range for each metric. For instance, 0.5 indicates a threshold of 1.0 for symmetric KL-divergence, but a threshold of 0.25 for Hellinger.

By changing the threshold ranges to suit each metric, the behavior of the clustering algorithm is more clearly depicted. Specifically, we are now able to observe

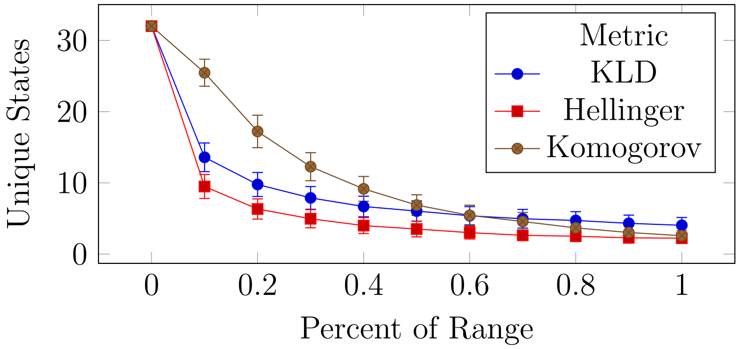


Figure 5.3: Normalized distance metric experiment.

the early stages of the Kolmogorov and Hellinger metrics with finer granularity, now giving the appearance that Kolmogorov tapers off much more slowly than Figure 5.2 implies. Based on this new plot, it is evident that symmetric KL-divergence and Hellinger exhibit similar behavior in that cluster sizes increase rapidly with the size of the approximation threshold. Furthermore, although the number of clusters decreases rapidly with symmetric KL-divergence, this behavior slows rather quickly, and by the end of the ranges considered it actually produces more clusters than either Hellinger or Kolmogorov. Finally, over the threshold interval from 0.0 to 0.75, a clustering algorithm using Kolmogorov combines intensity matrices at a more gradual rate. Based on the apparent convergence exhibited by the symmetric KL-divergence metric that occurs prior to reaching a single cluster, we choose to use KL-divergence going forward for the remainder of our experiments. Again, the choice of distance metric will generally depend on the underlying data, and future work in this area should consider distance metrics within the context of the problem being solved. Regardless of the metric used, a threshold value should be chosen based on the change in behavior of the underlying CTBN, as discussed in later sections.

5.5 Mapped Conditional Intensity Matrices

The hierarchical clustering algorithm discussed in the previous section provides groups of intensity matrices that may be treated as equivalent Markov processes within a CIM. To exploit these equivalences, a computationally efficient data structure must be leveraged that reduces either the space requirements to store a CTBN or the time needed to run inference. Given the existing one-to-many relationship in the output of the clustering algorithm, a discrete mapping function is a natural choice to represent intensity matrix similarities. The basic idea is to consolidate similar intensity matrices by mapping their corresponding parent instantiations to a single output state. Let $\mathbf{u}$ be the set of unique instantiations for the parent variables $\mathbf{U}$. A discrete function f may be used to map inputs $\mathbf{u}$ to a set of output states $\mathbf{m}$, where $|\mathbf{m}| < |\mathbf{u}|$, and each element $m \in \mathbf{m}$ is mapped to an element $u \in \mathbf{u}$. In other words, the function f is surjective and traditionally denoted as $f : \mathbf{u} \twoheadrightarrow \mathbf{m}$.

Referring back to the example clustering from Table 5.1, a discrete function f may be obtained simply by reversing the direction of the cluster notation.

$$\begin{aligned}
\{\mathbf{a}\} &\mapsto \mathbf{Q}_1 \\
\{\mathbf{b}\} &\mapsto \mathbf{Q}_2 \\
\{\mathbf{c}, \mathbf{d}\} &\mapsto \mathbf{Q}_3 \\
\{\mathbf{e}, \mathbf{f}, \mathbf{g}, \mathbf{h}\} &\mapsto \mathbf{Q}_4
\end{aligned}$$

Here, recall that each element $\mathbf{a}, \mathbf{b}, \ldots, \mathbf{h}$ represents a unique state instantiation to the parent variables $\mathbf{u} \in \mathbf{U}$. Each of the output states $\mathbf{m} \in \mathbf{M}$ is an intensity matrix, defined to be the mean of the intensity matrices in the corresponding cluster. Using this mapping, it is possible to represent the example CIM with only four intensity

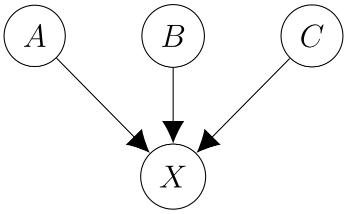


Figure 5.4: Three-parent network.

matrices rather than the original eight. This new representation is referred to as a mapped CIM, or simply an MCIM.

To place this in a more explicit context, see Figure 5.4, which depicts an example network with a single child node with three parents. The mapping for this example is as follows:

$$f(\mathbf{U}) = \begin{cases} \{a_0b_0c_0,\ a_0b_1c_1\} & \mapsto \mathbf{Q}_1 \\ \{a_1b_0c_1,\ a_1b_1c_0,\ a_1b_1c_1\} & \mapsto \mathbf{Q}_2 \\ \{a_0b_0c_1\} & \mapsto \mathbf{Q}_3 \\ \{a_0b_1c_0\} & \mapsto \mathbf{Q}_4 \\ \{a_1b_0c_0\} & \mapsto \mathbf{Q}_5 \end{cases}. \tag{5.2}$$

This mapping can be represented in two different ways. The first is that the standard CIM representation can be augmented to accommodate the new discrete mapping function f and will only specify an intensity matrix for each output state of the function. When an algorithm requires rates from an intensity matrix in a CIM, the intensity matrix is first looked up in the mapping function f based on the state instantiation of the parents.

The second option is to encode the mapping function f indirectly using standard CTBN semantics. This can be achieved by inserting a new synthetic variable, referred to as a mapping variable, into the network. In the three-parent example, the resulting

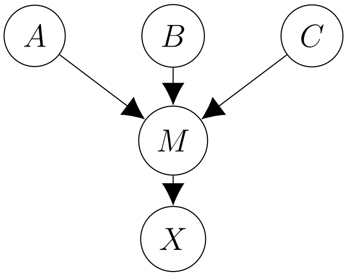


Figure 5.5: Three-parent network after inserting mapping variable.

structure after inserting the mapping variable is shown in Figure 5.5. The mapping variable M has one state for every cluster and is parameterized such that states are entered deterministically according to the parent instantiations. For example, consider the following matrix.

$$\mathbf{Q}_{M|a_1b_1c_0} = \begin{array}{c} \\ m_1 \\ m_2 \\ m_3 \\ m_4 \\ m_5 \end{array} \begin{array}{c} \begin{array}{ccccc} m_1 & m_2 & m_3 & m_4 & m_5 \end{array} \\ \left(\begin{array}{ccccc} -\infty & \infty & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & \infty & -\infty & 0 & 0 \\ 0 & \infty & 0 & -\infty & 0 \\ 0 & \infty & 0 & 0 & -\infty \end{array} \right) \end{array}$$

Here, the process transitions to state m_2 instantaneously due to the infinite transition rates, regardless of the current state of the process. Upon entering state m_2, the process remains there indefinitely, until the parent instantiation changes and a different intensity matrix is employed. The process is forced into state m_2 because f maps the instantiation $a_1b_1c_0$ to matrix $\mathbf{Q}_2$. For conciseness, this type of deterministic matrix is denoted Q_s^n, where n is the number of states, and s is the state into which the process transitions. In the general case, the entries for these matrices are defined as follows.

$$q_{i,j} = \begin{cases} \infty, & \text{if } i \neq s \wedge j = s \\ -\infty, & \text{if } i \neq s \wedge j = i \\ 0, & \text{otherwise} \end{cases}$$

These deterministic intensity matrices are used to parameterize the CIM for the mapping variable to encode the deterministic function f. Finally the child node X in Figure 5.5 is parameterized using the mean intensity matrices associated with each cluster. For instance, in the running example, $\mathbf{Q}_{X|m_2} = \mathbf{Q}_2$, as determined by the mapping function f.

5.5.1 Context-Independent Equivalence

Although the MCIMs discussed so far allow for the elimination of extraneous intensity matrices, there is a cost associated with representing the mapping between parent state instantiations and the corresponding matrices. Standard parameterization of a node X requires $n = \prod_{U \in Pa(X)} |U|$ separate intensity matrices, and although a discrete function f may substantially reduce the number of required intensity matrices, the function itself must contain the n original parent state combinations in order to associate them with the new subset of intensity matrices. This problem is only exacerbated when the function is encoded using intensity matrices, since the number of deterministic intensity matrices that are introduced is equivalent to the number of intensity matrices in the original parameterization. As an example, consider the network in Figure 5.5, which actually has $|M|$ *more* intensity matrices than the original network from Figure 5.4. Although this type of mapping has the advantage of improving interpretability by reducing the number of non-trivial parameters in the model, it does not reduce the overall complexity and may, in fact, have a negative impact. To combat this issue, this section discusses cases where discrete mappings can be decomposed to cover a smaller subset of parent variables, thereby reducing the space complexity.

To describe discrete mapping decomposition properly, some new terminology must be introduced. First, consider a subset $\mathbf{V}$ of the original parent set $\mathbf{U}$ for a node X. Going forward, this subset of variables is referred to as the *subject set*, and an instantiation $\mathbf{v}$ to the subject set is called a *subject*. Similarly, the remaining parents $\mathbf{C} = \{\mathbf{U} \backslash \mathbf{V}\}$ are referred to as the *context set*, and an instantiation $\mathbf{c}$ is a *context*. The basic idea is that if the intensity matrices for two or more subjects $\mathbf{V}' \subseteq \mathbf{V}$ are equal across all contexts $\mathbf{c} \in \mathbf{C}$, then the matrices may be consolidated using a discrete function over the reduced domain of $\mathbf{V}$. In this case, the subjects $\mathbf{V}'$ are said to be contextually equivalent to one another. As before, matrix equivalence can be determined using a clustering algorithm, and the notion of equivalence may be relaxed to allow for approximate mappings.

Contextual equivalence is described further by way of example. Consider a node X with parent nodes $\mathbf{U} = \{A, B, C, D\}$, each having two states. A standard CIM for node X will consist of $2^4 = 16$ intensity matrices. An example of a discrete map $f_{\mathbf{U}}$ obtained by clustering the intensity matrices is shown below. Here, the total number of intensity matrices is reduced from 16 to 8, but the number of state instantiations that are stored for the domain of $f_{\mathbf{U}}$ is still 16.

$$f_{\mathbf{U}} = \begin{cases} \{a_0b_0c_0d_0,\ a_0b_1c_0d_0,\ a_1b_0c_0d_0\} & \mapsto \mathbf{Q}_1 \\ \{a_0b_0c_0d_1,\ a_0b_1c_0d_1,\ a_1b_0c_0d_1\} & \mapsto \mathbf{Q}_2 \\ \{a_0b_0c_1d_0,\ a_0b_1c_1d_0,\ a_1b_0c_1d_0\} & \mapsto \mathbf{Q}_3 \\ \{a_0b_0c_1d_1,\ a_0b_1c_1d_1,\ a_1b_0c_1d_1\} & \mapsto \mathbf{Q}_4 \\ \{a_1b_1c_0d_0\} & \mapsto \mathbf{Q}_5 \\ \{a_1b_1c_0d_1\} & \mapsto \mathbf{Q}_6 \\ \{a_1b_1c_1d_0\} & \mapsto \mathbf{Q}_7 \\ \{a_1b_1c_1d_1\} & \mapsto \mathbf{Q}_8 \end{cases}$$

To compact the representation further, the discrete function $f_{\mathbf{U}}$ can be decomposed by taking advantage of context-independent equivalence. In this case, $\mathbf{V} = \{A, B\}$ will be treated as the subject set, implying that $\mathbf{C} = \mathbf{U} \backslash \mathbf{V} = \{C, D\}$ is the context set. By inspecting the first four mappings in the discrete function $f_{\mathbf{U}}$, it can be seen that subjects a_0b_0, a_0b_1 and a_1b_0 are equivalent across all contexts. Note that each of the three subjects appear in each of the first four sets in $f_{\mathbf{U}}$, and the context does not change within each of these sets. The basic idea is that if the context is known, then there is no difference between the consolidated subject states. For instance, if the context is fixed to the state c_1d_0, then the transition behavior is described by the intensity matrix $\mathbf{Q}_3$ if the subject is equal to a_0b_0, a_0b_1 or a_1b_0, and matrix $\mathbf{Q}_7$ if the subject is a_1b_1. This observed context equality can be used to derive two new discrete functions $f_{\mathbf{V}}$ and $f_{\mathbf{M}}$. The function $f_{\mathbf{V}}$ maps subject instantiations $\mathbf{v}_i$ to new states m_i. Function $f_{\mathbf{M}}$ then rewrites the original function $f_{\mathbf{U}}$ by replacing subject states with the corresponding state m_i obtained from function $f_{\mathbf{V}}$. These two functions are shown here.

$$f_{\mathbf{V}} = \begin{cases} \{a_0b_0,\ a_0b_1, a_1b_0\} & \mapsto m_0 \\ \{a_1b_1\} & \mapsto m_1 \end{cases}$$

$$f_{\mathbf{M}} = \begin{cases} \{m_0c_0d_0\} & \mapsto \mathbf{Q}_1 \\ \{m_0c_0d_1\} & \mapsto \mathbf{Q}_2 \\ \{m_0c_1d_0\} & \mapsto \mathbf{Q}_3 \\ \{m_0c_1d_1\} & \mapsto \mathbf{Q}_4 \\ \{m_1c_0d_0\} & \mapsto \mathbf{Q}_5 \\ \{m_1c_0d_1\} & \mapsto \mathbf{Q}_6 \\ \{m_1c_1d_0\} & \mapsto \mathbf{Q}_7 \\ \{m_1c_1d_1\} & \mapsto \mathbf{Q}_8 \end{cases}$$

Using function composition, the original function $f_{\mathbf{U}}$ may be rewritten in terms of $f_{\mathbf{M}}$ and $f_{\mathbf{V}}$, as follows.

$$f_{\mathbf{U}}(\mathbf{u}) = f_{\mathbf{M}}(\{f_{\mathbf{V}}(\mathbf{v})\} \cup \mathbf{c}) \tag{5.3}$$

Here, the original state instantiation $\mathbf{u}$ passed as input to $f_{\mathbf{U}}$ is decomposed into the subject $\mathbf{v}$ and the context $\mathbf{c}$. The subject instantiation is then passed as input to the subject mapping function $f_{\mathbf{V}}$, producing an output m_i which is combined with the existing context instantiation $\mathbf{c}$. This newly obtained state instantiation is passed as input to the mapping function $f_{\mathbf{M}}$, which returns the correct intensity matrix.

Note that the domain of $f_{\mathbf{V}}$ is comprised of four state instantiations, and that $f_{\mathbf{M}}$ requires eight instantiations. In other words, there are 12 instantiations required to define both of the decomposed functions, compared to the 16 used for the original function $f_{\mathbf{U}}$. Furthermore, the instantiations for $f_{\mathbf{V}}$ and $f_{\mathbf{M}}$ cover two and three variables respectively, compared to the four variable instantiations in function $f_{\mathbf{U}}$. This decomposed representation encodes the exact same information as the original function $f_{\mathbf{U}}$, despite its more compact representation. Note that although a reduction from 16 instantiations to 12 may seem trivial, the advantages observed in this example are compounded as the size of the parent sets increase.

For a more concrete example, refer back to the example drug effect network presented in Figure 2.3 from Chapter 2, which is repeated here in Figure 5.6 for convenience. There exist identical intensity matrices in the CIM $\mathbf{Q}_{P|B,C}$ that can be consolidated using the principle of context-independent equivalence. For instance, if the barometric pressure B is known to be in state b_0, then that same transition behavior is expected regardless of whether the concentration of the drug C is in state c_0 or c_2. Similarly, when $B = b_1$, there is no difference between c_0 and c_2. An even stronger example of context-independent equivalence is in the case where $B = b_2$, in which case all instantiations of C result in the same transition behavior. In other words, a patient's pain is fully independent of the concentration of the drug when the barometric pressure is known to be in state b_2.

It is worth noting that this decomposed mapping is a generalized version of the MCIM introduced earlier. Specifically, the mappings of the type shown in Equation 5.2 considered cases where the subject set was equal to the parent set ($\mathbf{U} = \mathbf{V}$); therefore, the context set was empty ($\mathbf{C} = (\mathbf{U} \backslash \mathbf{V}) = \emptyset$). This results in a mapping of $f_{\mathbf{U}}(\mathbf{u}) = f_{\mathbf{M}}(f_{\mathbf{V}}(\mathbf{u}))$, which can be represented more concisely using a single discrete function $f_{\mathbf{U}}$ relating inputs $\mathbf{u}$ to their corresponding intensity matrices.

As before, the decomposed functions can be encoded using standard CTBN semantics by introducing a new mapping variable. The key difference is that now

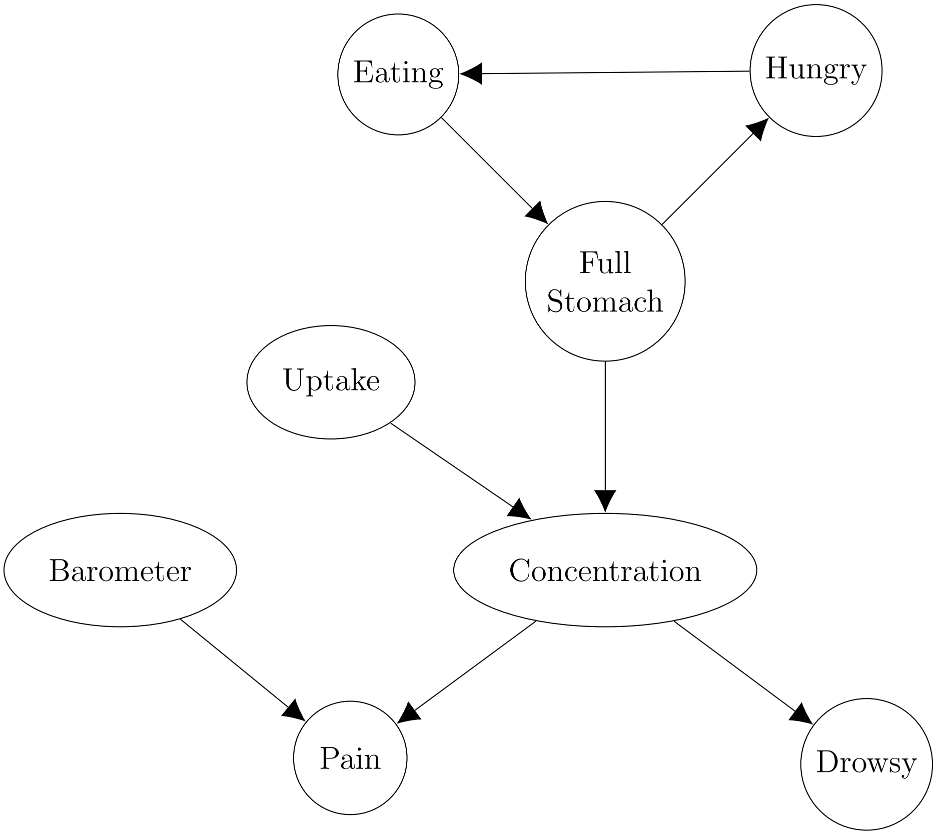


Figure 5.6: The drug effect example network.

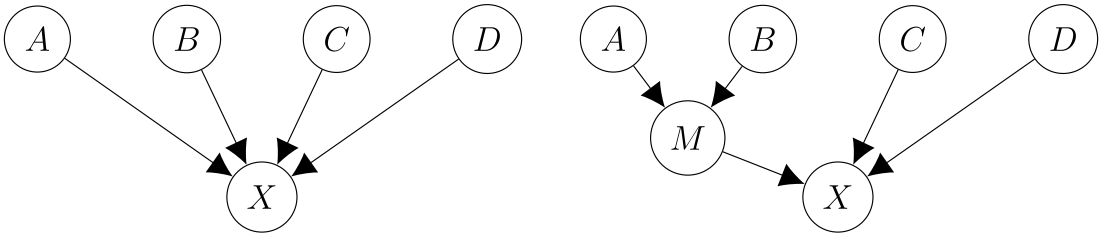


Figure 5.7: Four-parent network before (left) and after (right) inserting a mapping variable based on a discrete function with context-independent equivalence.

that contextual equivalence has been introduced, the subject set $\mathbf{V}$ contains only a subset of the total number of parents $\mathbf{U}$. The mapping variable is parameterized according to the function $f_{\mathbf{V}}$; therefore, the parent set for the mapping variable is defined by $\mathbf{V}$. Similarly, the original child node X is reparameterized according to $f_{\mathbf{M}}$, meaning that the new parent set for X is $\{M \cup \mathbf{C}\}$.

Figure 5.7 shows a network before and after a new mapping variable M has been inserted to encode the discrete mapping. Note that the behavior of M is determined by nodes A and B, which are the members of the subject set. As before, M is parameterized using intensity matrices with deterministic transitions that map the parent instantiations to a new set of states, where $|M| < |A| \cdot |B|$. For this particular example, $|A| \cdot |B| = 4$, and $|M| = 2$. This new network requires $|M| \cdot |C| \cdot |D| = 8$ intensity matrices to parameterize X and $|A| \cdot |B| = 4$ deterministic intensity matrices are needed to parameterize M. These correspond to the domains for functions $f_{\mathbf{M}}$ and $f_{\mathbf{V}}$ respectively. This compact MCIM representation uses only twelve intensity matrices compared to the sixteen required by the original network. Once again, these savings become more pronounced as the size of the networks grow.

An additional benefit gained when taking advantage of context-independent equivalence is that the decomposed mapping functions can be inserted for multiple subject sets. For instance, multiple state combinations of variables A and B may be treated as equivalent for all contexts of C and D. Similarly, there may be state combinations of C and D that are equivalent regardless of the state of A and B. In this case, multiple mapping variables could be inserted, further decomposing the representation. The process of identifying proper mappings can be done efficiently in a pairwise fashion between variables, building up to potentially larger groups. This is analogous to the process of hierarchical clustering described earlier.

5.5.2 MCIM Experiments

To evaluate MCIMs as a compact representation, experiments are conducted that compare CTBNs before and after introducing the mappings. For consistency, the discrete functions are represented using deterministically parameterized mapping nodes in each of the experiments. This section is broken into two parts, corresponding to unstructured and structured synthetic data. For the unstructured data experiments, models are produced by randomly generating each intensity matrix. Conversely, the structured data experiments generate intensity matrices using a pseudo-random procedure that follows specific constraints. The intent is to determine how structure in the data impacts the behavior exhibited by discrete mappings.

Throughout the experiments, the compact model is evaluated based on the change in complexity as well as the error introduced when compared to the original model. Since the compact models used in these experiments only change the parameterization of a single node X, error is determined by querying the probability distribution over the states of X through time. Let G be a baseline model containing a node X, and let G' be a compacted model that simplifies the parameterization of node X. Let $P_{X(t)}$ and $P'_{X(t)}$ be the distributions over the states of variable X at time t obtained from running inference over G and G' respectively. Then error can be defined using discrete (nonsymmetric) KL-Divergence, shown below. Given that X is discrete, calculation of the KL-Divergence can be computed numerically by summing over the individual states. This quantifies the extent to which the compacted model G' diverges from the target baseline model G at some time t.

$$D_{KL}(P_{X(t)}||P'_{X(t)}) = \sum_{x \in X} P_{X(t)}[x] \log \frac{P_{X(t)}[x]}{P'_{X(t)}[x]}$$

This can be generalized to encompass an entire window of time $t = [t_s, t_e)$ by integrating over t.

$$D_{KL}(P_X||P'_X) = \int_{t_s}^{t_e} D_{KL}(P_{X(t)}||P'_{X(t)})dt \tag{5.4}$$

Note that while this is one possible error measure, other approaches may be better suited for specific applications. For instance, perturbation realization is a useful technique for evaluating parameter sensitivity in Markov processes and has recently been adapted to work within the CTBN framework as well [135]. In essence, perturbation realization computes the expected long-term change in value of a function defined over a CTBN [132]. To identify long-term behavior, the steady-state distribution is identified. This process relies on the ergodicity assumption, which assures that no states are absorbing. For cases where this assumption can be guaranteed and interest lies in the long-term behavior of the process, perturbation realization could be used to evaluate the quality of the approximations discussed here. For our experiments, we continue with the KL-Divergence error measure from Equation 5.4, which does not require that the models be ergodic and focuses on short-term behavior rather than using the steady-state distributions.

5.5.3 Unstructured Data Experiments

For the unstructured data experiments, networks are generated with a set of nodes $\mathbf{X}$. Consider a node $X_k \in \mathbf{X}$ with n_k states. The node X_k is parameterized using a CIM $\mathbf{Q}_{X_k|\mathbf{U}}$ with $n_k \times n_k$ dimensional intensity matrices. Each intensity matrix is generated randomly such that each off-diagonal entry $q_{i,j|i \neq j}$ is drawn from a uniform distribution $U(0.0, 1.0)$. After each of the $(n_k - 1) \cdot (n_k - 1)$ rates are obtained, the diagonal entries are set to the negated sum of the remaining row, as per the requirements for a valid intensity matrix. This process is repeated for each intensity matrix in the CIM $\mathbf{Q}_{X_j|\mathbf{u}} \in \mathbf{Q}_{X_j|\mathbf{U}}$. This, in turn, is applied to each node $X_j \in \mathbf{X}$.

Given that the intent of the experiments is to demonstrate various CIM representations, the single-child multi-parent network structure used in previous examples is employed throughout the experiments to allow for a controlled means of changing the number of intensity matrices within the CIM. Formally, each network consists of a single child variable X, as well as m parent variables U_i, where $i \in (1, m)$. A directed edge is added from each parent variable U_i to the child variable X. As a result, each of the m parents has a CIM consisting of a single intensity matrix with dimensions $|U_i| \times |U_i|$ states each, while the CIM for the child variable has $\prod_i |U_i|$ intensity matrices, each with dimensions $|X| \times |X|$. The described network structure is shown in Figure 5.8. The number of states $|X|$, the total number of parent variables m, and the number of states $|U_i|$ for each parent are varied by experiment to investigate different aspects of compact CIM representations.

5.5.3.1 Approximation Threshold The first experiment is designed to investigate the effect of the approximation threshold in the clustering algorithm. The basic idea is that small approximation thresholds will produce clusters that consist of

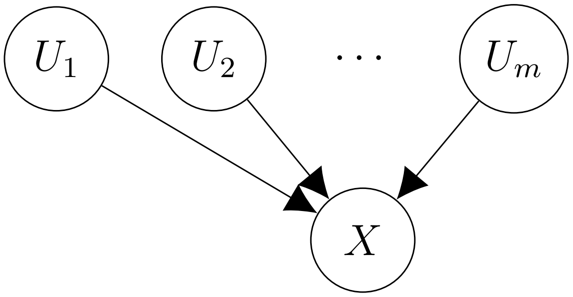
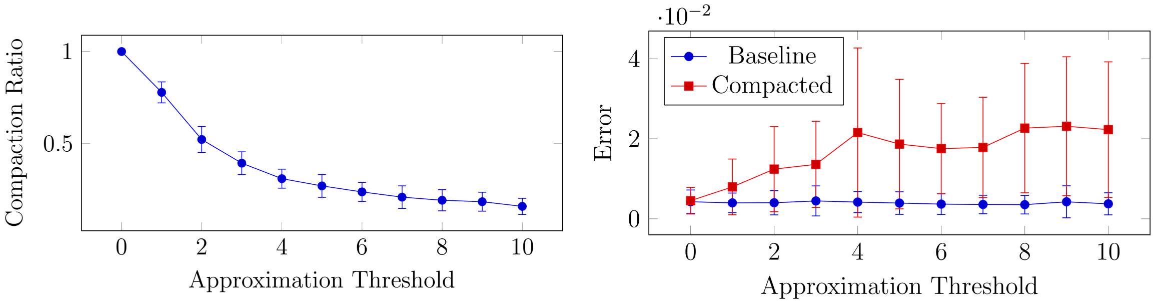


Figure 5.8: Network structure for CTBNs used throughout experiments.

intensity matrices that are very nearly identical to one another. Conversely, large thresholds will consolidate more intensity matrices, thereby producing larger clusters with intensity matrices that are more dissimilar. To begin, a fixed network structure containing five binary parent nodes and a five-state child node was generated. Then ten compact versions of the network were produced using an MCIM encoding obtained from clusters that were generated using a symmetric KL-divergence measure with approximation thresholds varying from 0.0 to 10.0. Finally, inference was run over the compact network and the original network, with the KL-divergence between the query results as shown in Equation 5.8 serving as an error measure. Importance sampling with 10000 samples was used as the inference algorithm [38,40]. To provide a baseline of comparison, error was measured against two inference runs over the same original network to quantify the error introduced by the approximate inference algorithm itself. This process was repeated 100 times to obtain a sufficiently large population of results.

Figure 5.9a shows the compaction ratio as a function of the approximation threshold, with error bars showing standard error. Here, the compaction ratio is defined as the ratio between the unique intensity matrices required by the MCIM representation and the original number of required intensity matrices. As such, smaller values indicate more efficient compact representations. The number of unique



(a) Scalability analysis.

(b) Accuracy analysis.

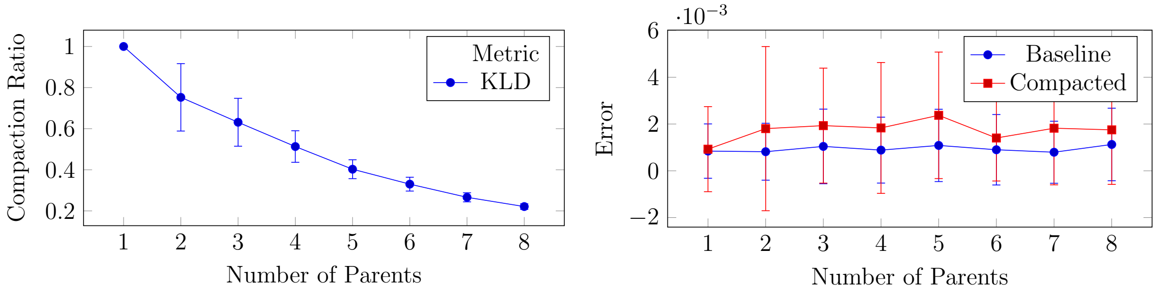
Figure 5.9: Impact of approximation threshold with unstructured data using a MCIM representation.

intensity matrices corresponds to the number of inputs to the function $f_{\mathbf{M}}$, or equivalently the number of intensity matrices for X in a CTBN with an inserted mapping variable M. Similarly, Figure 5.9b shows the error for the compacted model, as well as the baseline error measure as a function of the approximation threshold, with the error bars representing the standard deviation.

The behavior observed in Figure 5.9a matches expectation. When the approximation threshold is set to zero, no compaction occurs, resulting in a compaction ratio of 1.0. Due to the random nature of the data, all produced clusters contain only a single intensity matrix, meaning that no consolidation occurs. As this approximation threshold is increased, cluster sizes increase as well. These clusterings can be encoded using discrete mappings, which reduce the total number of unique intensity matrices. Note that a compaction ratio near zero indicates a small number of clusters. In the most extreme case, all intensity matrices may be merged into a single cluster, meaning that the child variable X acts approximately the same regardless of the states of its parents. If this were truly the case, the most efficient method for representing this independence would be to break all edges between the parents and the child and use a single intensity matrix to describe the now independent node's behavior. This is a process referred to as node isolation, and can itself be used as an inference algorithm [134]. Given the random nature of the data, however, it is almost certainly the case that a substantial reduction in the intensity matrices indicates an excessively large approximation threshold. Instead, a smaller approximation threshold should be employed that balances the accuracy of the approximation with the efficiency of the compact representation.

The error results shown in Figure 5.9b indicate a steadily positive trend, where small approximation thresholds correspond to smaller error values, and large thresholds produce a larger error. This is not surprising in that large approximation thresholds allow increasingly diverse intensity matrices to be merged into a single cluster. Ultimately, this means that larger approximation thresholds will result in models that represent the same information with fewer parameters, which has the potential to introduce error. Another interpretation of the observed error behavior is that there is a trade-off between complexity and reliability. A small approximation threshold will introduce very little error but may not provide much in the way of parameter reduction. Alternatively, a large approximation threshold will likely allow for substantial savings in the number of parameters but may introduce an excessive amount of error in the process.

5.5.3.2 Network Structure The next experiment is designed to determine the impact that the network graph structure has on the MCIM compact representation. Specifically, the structure from Figure 5.8 is used, where the number of parents m is varied from one to eight. Just as before, all parent nodes in the network are binary, while the child variable consists of five states. For all intensity matrices, rates are drawn from uniform distributions ranging from 0.0 to 1.0. In this experiment, the symmetric KL-divergence distance metric was used with an approximation threshold



(a) Scalability analysis.

(b) Accuracy analysis.

Figure 5.10: Impact of network structure with unstructured data using a MCIM representation.

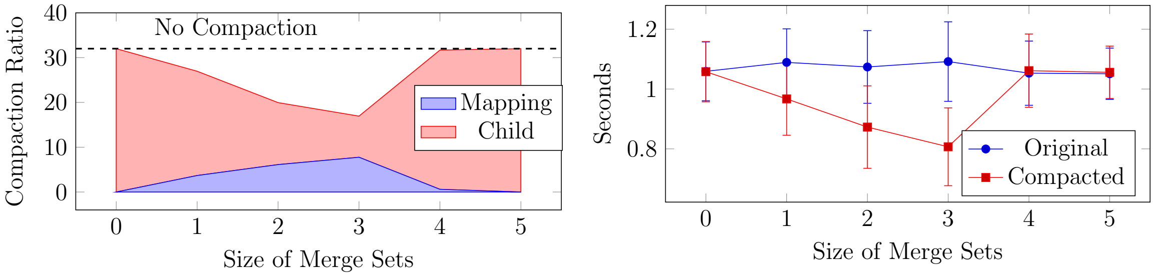
of 0.25 throughout. Models were compacted using the MCIM representation, and inference results were compared using the same error measure from the previous experiment. This process was performed 100 times, and the compaction ratio and error are plotted in Figures 5.10a and 5.10b respectively.

As shown by Figure 5.10a, the compaction ratio decreases with the size of the parent set. In other words, networks with more parents correspond to more consolidated intensity matrices. This increase in savings is due to the increase in the total number of intensity matrices. Networks are parameterized by drawing rates from a fixed uniform distribution, meaning that intensity matrices can be viewed as data points that fall within a fixed space of possible parameterizations. Given this random parameterization and the fixed approximation threshold, the total number of intensity matrices is the primary factor influencing the results of the clustering algorithm. Since an increase in the number of parents produces an exponential increase in the number of intensity matrices, more of the space is covered with larger networks, allowing for more effective clusterings.

Figure 5.10b shows error as a function of the number of parents, both for the compacted model and the baseline. For networks with a single parent, the error appears to match the baseline, which makes sense given that there is no compaction occurring. For all other network sizes, the compacted model produces a consistently larger error than the baseline, likely due to the constant approximation threshold. The important aspect to note is that there is no discernible pattern for the compacted error in terms of either a positive or negative trend. This indicates that the number of parents do not have a significant impact on the error introduced by the MCIM representation. Furthermore, the selected approximation threshold in this case produces a consistently small error, and based on the error bars, the difference from the baseline is not statistically significant. This demonstrates that the MCIM representation becomes increasingly effective as the number of parents grows, with no degradation in error.

5.5.4 Structured Data Experiments

For the structured data experiments, networks are generated initially in the same fashion as the unstructured data experiments, with identical structure and random parameterization. The clustering algorithm uses the symmetric KL-divergence distance measure with a constant approximation threshold of 0.25. Then models are modified to introduce structure synthetically into the data. Specifically, a subset $\mathbf{V}$ of parent variables is chosen randomly from the total set of parents $\mathbf{U}$. A set of state instantiations $\mathbf{v}'$ is chosen randomly from the total set of instantiations of $\mathbf{V}$. Each selected subset of state instantiations to the subset of parent variables is referred to as a merge set. The size of the merge sets as well as the total number of merge sets vary between experiments. The only stipulation is that the assignments are a valid subset such that $\mathbf{V} \subseteq \mathbf{U}$ and $\mathbf{v}' \subseteq \mathbf{v}$. These state instantiations are then merged such that the corresponding intensity matrices are assigned to be equivalent for each context, which is achieved by taking the mean of each group of $|\mathbf{v}'|$ matrices. To



(a) Space analysis.

(b) Time analysis.

Figure 5.11: Impact of the size of merge sets with structured data using a MCIM representation.

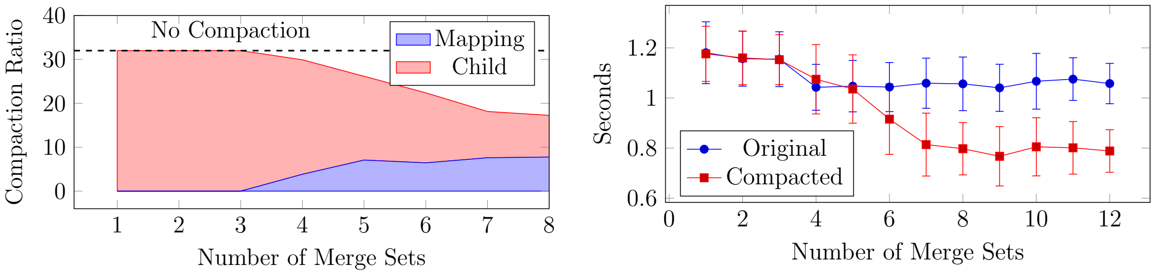
allow for approximations, noise is introduced to the rates in the intensity matrices. This is achieved by adding a random variable drawn from a uniform distribution $U(-\alpha, \alpha)$, producing intensity matrices that are approximately equal to one another. This procedure introduces context independence into otherwise unstructured data.

5.5.4.1 Merge Set Size For the first structured experiment, the size of the merge sets was varied to determine the effectiveness of the compact representation. Specifically, the size of the merge sets ranged from zero to five, where zero indicates no data structure, and five means that state instantiations to all five parents are merged with no context. For each case, a maximum value of eight was set as the number of merge sets, meaning that at most eight states were altered to be approximately equivalent. Results are shown in Figure 5.11.

Figure 5.11a shows a stacked line graph of the size of the model, broken down by both intensity matrices and the discrete function. Specifically, the upper red region shows the number of unique intensity matrices that are stored for the child node, and the lower blue region shows the number of intensity matrices needed to parameterize an inserted mapping node to achieve the compact representation. The dashed line represents the size of the original model with no compaction applied. An important aspect to note is that the size of the model decreases when the number of merge sets increases to three, and increases again as it approaches five. This is because the reduction in the number of intensity matrices for the child node outweighs the increase in size that is introduced by inserting the mapping variable. Charts showing the approximation error are omitted, as there is no discernible difference between the baseline and compact models. This is due to the choice of approximation threshold in combination with similarity in the structured data for these experiments, which results in a negligible loss of information.

Figure 5.11b shows runtimes for inference run over both the original model shown as blue circles, and the compacted model shown by red squares, with error bars indicating standard error. As expected, the original model remains consistent regardless of the size of the merge sets. Runtimes for the compacted model decrease and increase proportional to the size of the model shown in Figure 5.11a, and for the case where merge sets are of size three, the difference in runtimes is statistically significant. This decrease in runtime does not come from algorithmic improvements, but instead occurs due to the reduction in the model size.

5.5.4.2 Number of Merge Sets The next structured experiment looks at the effect that the number of merge sets has on the resulting compact representation. The same network structure from the previous experiment was used, and in all cases the size of the merge sets was fixed to a size of three. The number of merge sets was varied from one to eight, where one indicates a model with no compaction, and eight is the total number of state instantiations for three binary parent variables, meaning that all state combinations for the parents are equivalent. The results are shown in Figure 5.12.



(a) Space analysis.

(b) Time analysis.

Figure 5.12: Impact of the number of merge sets with structured data using an MCIM representation.

Figure 5.12a shows the size of the model, with the upper red region showing the number of intensity matrices required to parameterize the child node, and the lower blue region representing the number of intensity matrices introduced by inserted the mapping variable. Once again, the dashed line indicates the baseline number of intensity matrices in the original model prior to compaction. For very small numbers of merge sets, there is no compaction and the total number of intensity matrices matches the baseline. As the number of merge sets increase, discrete functions are introduced that ultimately decrease the total number of intensity matrices in the network, despite the need for parameterizing the new mapping variables.

Runtimes for this experiment are shown in Figure 5.12b. Once again, there is no statistical difference between the runtimes for inference applied to the original model, regardless of the number of the merge sets. However, the runtimes for the compacted models decrease proportionally to the size of the model, which is reduced as the number of merge sets grows. When the number of merge sets is greater than six, the difference in runtimes is statistically significant. This shows that models may be more represented efficiently for structured data where the number of merge sets is large, both in terms of the space required to parameterize the model, and the time required to perform inference.

5.6 Tree-Structured Conditional Intensity Matrices

In addition to discrete functions, a tree data structure may be used to represent a CIM compactly. This tree-structure representation of intensity matrices relates to context-specific independence in BNs, which uses a tree structure to store probabilities in a conditional probability table [10]. The intent of context-specific independence in BNs is to capture independencies that cannot be encoded using the graph structure alone, and the same principle holds in the domain of CTBNs. Boutilier *et al.* take advantage of regularities in the conditional probability table and encode these regularities using a tree structure. This section demonstrates how the same process can be applied to CIM regularities obtained via a clustering algorithm. The tree encoding of a CIM is referred to as a tree-structured CIM, or simply a TCIM.

Let X be a node in a CTBN with parents $\mathbf{U}$, and let $\mathbf{C}$ be the set of clusters obtained by hierarchically clustering the intensity matrices in the CIM for X. A tree T_X may be used to encode context-specific independence exhibited by the clusters $\mathbf{C}$. Let $\mathbf{I}(T_X)$ be the set of interior nodes in tree T_X, and let $\mathbf{L}(T_X)$ be the set of leaf nodes. Furthermore, let $\mathbf{P}(T_X)$ be the set of paths in the tree, where each path $P \in \mathbf{P}(T_X)$ is a sequence of nodes from the root N_r to a leaf node $N \in \mathbf{L}(T_X)$. A variable U from the parent set $\mathbf{U}$ is assigned to each interior node $N \in \mathbf{I}(T_X)$, such that no path $P \in \mathbf{P}(T_X)$ contains more than one assignment $\mathbf{u}$. There are $|\mathbf{U}|$ edges leaving an interior node with assignment $\mathbf{u}$, each labeled with the states associated with variable U. Each leaf node $N \in \mathbf{L}(T_X)$ is assigned a unique intensity matrix. To retrieve an intensity matrix from T_X associated with the parent set instantiation $\mathbf{u}$, the tree is traversed starting at the root, following edges corresponding to the state assignments in $\mathbf{u}$, until an intensity matrix at a leaf node is reached.

The primary advantage of the TCIM representation is that entire branches of a tree may be pruned if all leaves in the subtree contain equivalent or approximately equivalent intensity matrices. In that case, the root of the subtree is replaced with a mean intensity matrix, creating a leaf that occurs at a higher level in the tree. The paths of a pruned tree are therefore potentially shorter than the total number of parents $|\mathbf{U}|$. This pruning process is how the TCIM structure achieves its compact representation.

An example is now presented to demonstrate how a TCIM may be used to store intensity matrices in a CIM efficiently. Consider a four-node network with a variable X and three parent variables $\mathbf{U} = \{A, B, C\}$. Here, let $|A| = 3$, $|B| = 2$, and $|C| = 2$, for a total of twelve intensity matrices in the unmodified CIM $\mathbf{Q}_{X|\mathbf{U}}$. An example clustering of these intensity matrices is as follows.

C_1:	$\mathbf{Q}_{X\|\{a_0,b_0,c_0\}}$	$\mathbf{Q}_{X\|\{a_0,b_0,c_1\}}$		
C_2:	$\mathbf{Q}_{X\|\{a_0,b_1,c_0\}}$	$\mathbf{Q}_{X\|\{a_0,b_1,c_1\}}$		
C_3:	$\mathbf{Q}_{X\|\{a_1,b_0,c_0\}}$			
C_4:	$\mathbf{Q}_{X\|\{a_1,b_1,c_0\}}$			
C_5:	$\mathbf{Q}_{X\|\{a_1,b_0,c_1\}}$	$\mathbf{Q}_{X\|\{a_1,b_1,c_1\}}$		
C_6:	$\mathbf{Q}_{X\|\{a_2,b_0,c_0\}}$	$\mathbf{Q}_{X\|\{a_2,b_0,c_1\}}$	$\mathbf{Q}_{X\|\{a_2,b_1,c_0\}}$	$\mathbf{Q}_{X\|\{a_2,b_1,c_1\}}$

Take, for example, cluster C_1. Both matrices in this cluster are tied to states a_0 and b_0, while the value of C varies across its two states. In other words, once it has been determined that $A = a_0$ and $B = b_0$, then it does not matter what the state of C is, because the intensity matrix for this cluster will be the same, regardless of C's value. More formally, X is contextually independent of C given a_0 and b_0. This

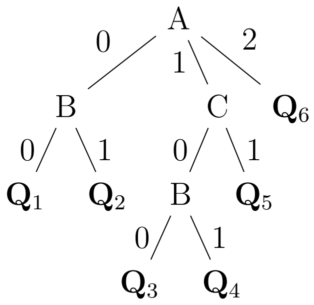


Figure 5.13: Tree-structured CIM for node X.

contextual independence can be represented using a tree, as shown in Figure 5.13. Note that if node A is followed down branch 0, and B is followed down branch 0, then the tree terminates at a leaf node with a value of $\mathbf{Q}_1$, which is the intensity matrix associated with cluster C_1. Following this path did not require evaluation of the variable C. Similar tree pruning is performed for the other clusters, and indeed there are a total of six distinct intensity matrices at the leaves of the tree, corresponding to each of the six clusters.

Just as with the discrete functions described previously, the tree structures used to encode a CIM may be represented using standard CTBN semantics. In the BN literature, the analogous process is referred to as structural decomposition. This is achieved by retaining the variable associated with the root node of the tree as a parent of X but removing all other variables. Additional variables are then inserted into the parent set of X, one for each state of the root variable, indicating the presence or absence of the state. These nodes in turn have parents associated with the next variable down the tree branch, along with all of the potential state instantiations associated with the variable.

Figure 5.14 shows the structural decomposition for the tree in Figure 5.13. First, note that nodes A, B, and C are the same nodes from the original network in Figure 5.4, with identical parameterization. Conceptually, the newly introduced nodes can

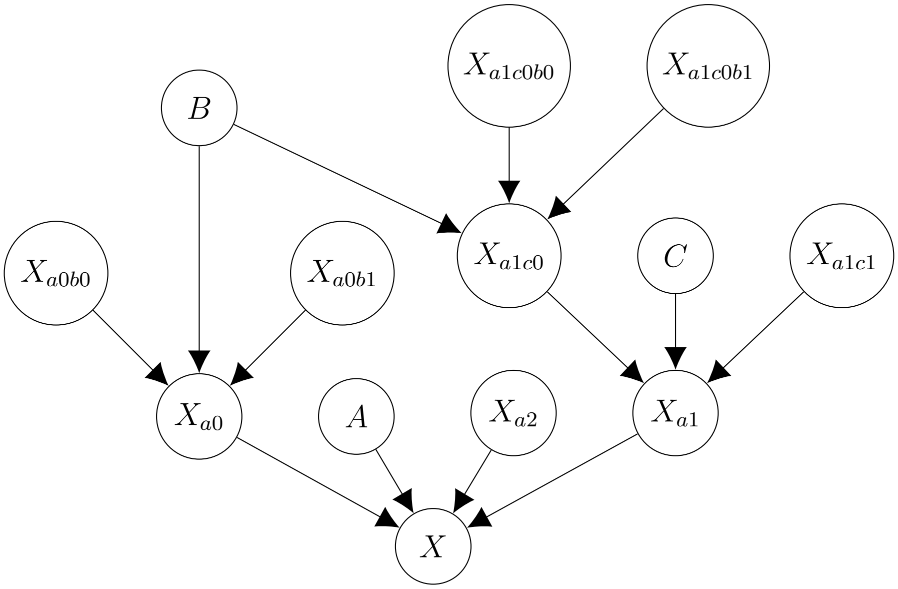


Figure 5.14: Structure decomposition using the tree-structured CIM from Figure 5.13.

be thought of as random variables indicating the value of X given a specific context. For example, X_{a0} encodes the value that X takes on in the event that $A = a_0$. Nodes X_{a0}, X_{a1}, and X_{a2} can be obtained independently of one another, and the "correct" value can be retrieved after the value of A becomes known.

Nodes X, X_{a0}, X_{a1}, and X_{a1c0} are parameterized using deterministic CIMs that force the node to one of the parent boolean values. As an example, if B is found to be b_0, then X_{a0} will transition with deterministic rates to ensure X_{a0} matches the state of X_{a0b0} at any point in time. Nodes X_{a2}, X_{a0b0}, X_{a0b1}, X_{a1c1}, X_{a1c0b0}, and X_{a1c0b1} are parameterized each with a unique intensity matrix corresponding to the six means of the clusters discussed previously. For instance, X_{a2} is parameterized using the intensity matrix corresponding to the mean of the matrices from cluster C_6 shown earlier in this section.

Again, consider the CIM $\mathbf{Q}_{P|B,C}$ in the example drug effect network for a more concrete example of context-specific independence. When the barometric pressure B

is known to be in state b_2, then the state of the drug concentration C is irrelevant. Using a tree representation, if B is the root of the node and path b_2 is taken, then an intensity matrix can be stored at this level without the need to specify the value of C. Note that although there are two other pairs of identical matrices in $\mathbf{Q}_{P|B,C}$, they cannot be accounted for by context-specific independence. Specifically, $\mathbf{Q}_{P|b_0,c_0} = \mathbf{Q}_{P|b_0,c_2}$ and $\mathbf{Q}_{P|b_1,c_2} = \mathbf{Q}_{P|b_1,c_2}$. The issue is that although states c_0 and c_2 are equivalent when B is in state b_0 or b_1, the behavior of pain is not independent of C because of the change observed for state c_1. As discussed in Section 5.7, these types of similarities are better described using a MCIM.

One concern when using standard CTBN semantics to represent a TCIM is the rapid increase in the number of nodes. Indeed, in many cases there is an increase in the total number of intensity matrices required to parameterize the network after introducing context-specific independence. However, just as with the discrete mapping, there are additional advantages beyond parameter reduction. In terms of interpretability, the structure shown in Figure 5.14 provides additional information about independence that was not immediately observable in the original network structure. In terms of efficiency, the new representation may reduce the maximal parent set size. Despite the increase in the total number of nodes, the size of the parent sets may be the more important factor when using inference algorithms that employ cluster graphs.

5.6.1 TCIM Experiments

This section presents the results of several experiments designed to investigate the properties exhibited by TCIM representations. The experiments are structured in much the same way as the MCIM experiments from Section 5.5.2. Specifically, a TCIM representation is applied to unstructured data in Section 5.6.2 and structured data in Section 5.6.3. These experiments use the same network structure shown in Figure 5.8 and the KL-Divergence error measure presented in Equation 5.4.

5.6.2 Unstructured Data Experiments

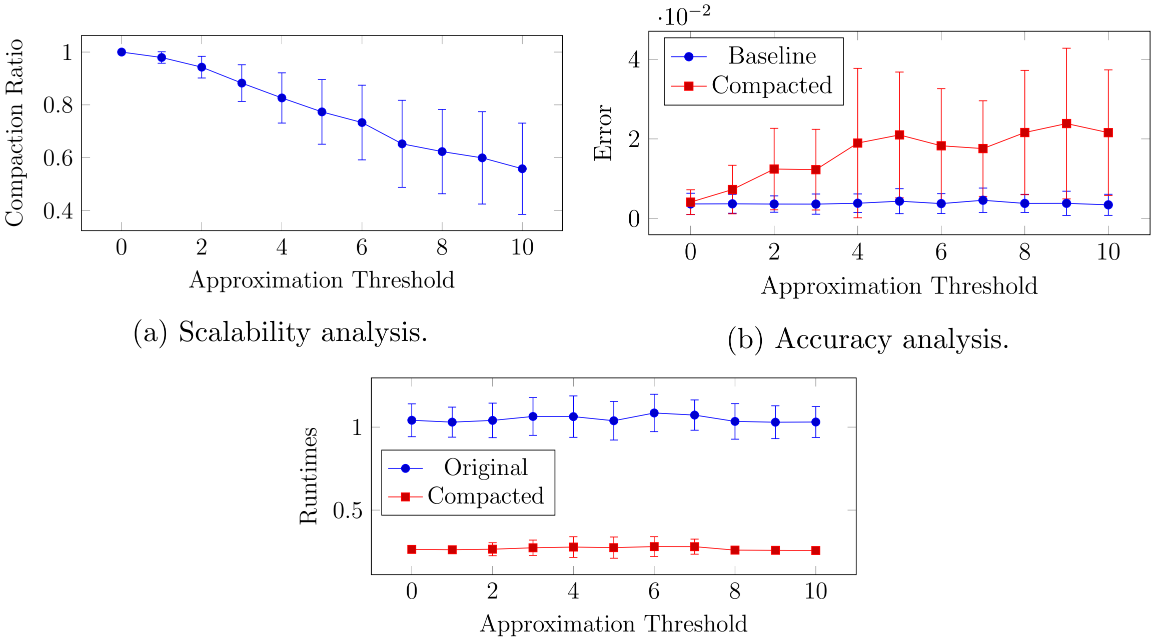
The TCIM unstructured data experiments are set up similarly to the MCIM unstructured experiments from Section 5.5.3. Each node in the network is parameterized using randomly generated intensity matrices. Specifically, each intensity matrix is produced by drawing off-diagonal transition rates from a uniform distribution $U(0.0, 1.0)$, with the diagonal entries in the matrix being defined implicitly by the remaining rates in the row. This type of network parameterization ensures that there are no dependencies between the intensity matrices, guaranteeing that there is no inherent underlying structure in the data.

5.6.2.1 Approximation Threshold

The first experiment conducted with unstructured data focused on determining the impact of the clustering approximation threshold. This experiment is set up identically to the first MCIM experiment from Section 5.5.4, except that the TCIM compact representation scheme is used in place of the MCIM representation. The results are shown in Figure 5.15.

Figure 5.15a shows that as the approximation threshold increases, the total number of intensity matrices required to specify the model decreases, and the variance increases. Furthermore, Figure 5.15b shows that the error observed by running inference on a compacted model increases with the size of the approximation threshold. In other words, the model may be represented more compactly by accepting worse approximations.

Note that the savings in space requirements shown in Figure 5.15a are less dramatic than the corresponding MCIM results shown in Figure 5.9a. This is due to the difference in representational capabilities. A single discrete function is capable of

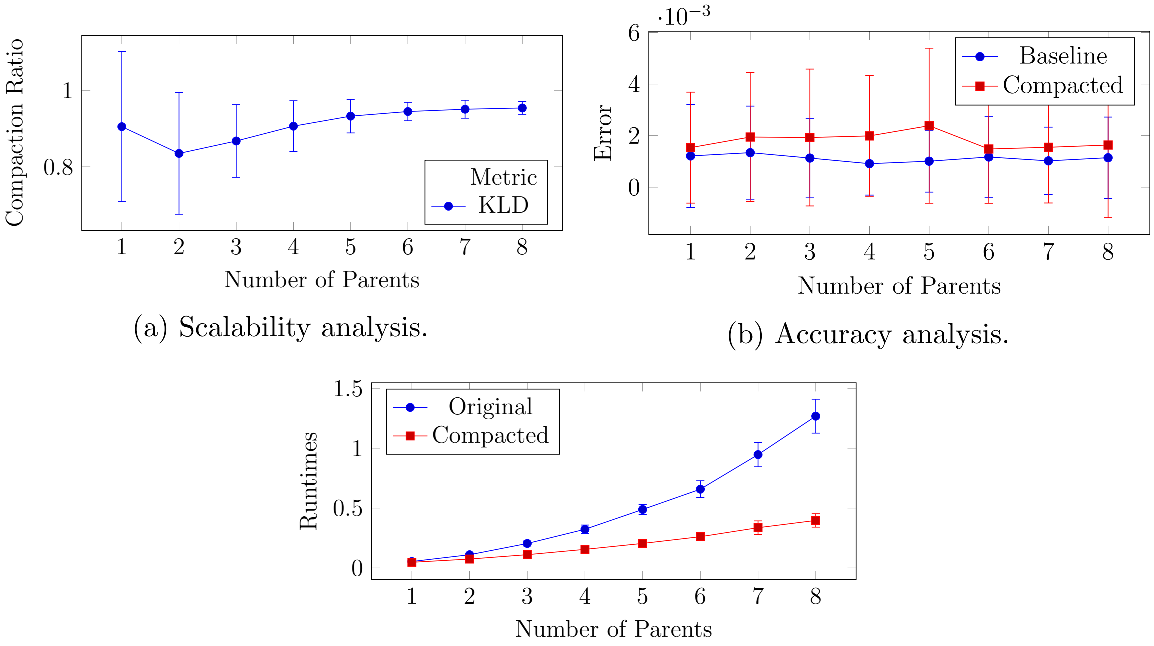


(a) Scalability analysis.

(b) Accuracy analysis.

(c) Runtime analysis.

Figure 5.15: Impact of approximation threshold with unstructured data using a TCIM representation.



(a) Scalability analysis.

(b) Accuracy analysis.

(c) Runtime analysis.

Figure 5.16: Impact of network structure with unstructured data using a TCIM representation.

fully encoding all clusterings, while a tree structure cannot necessarily capture all the same information. Despite this, a tree structure is a more efficient means of encoding information when compared to a single discrete mapping function.

5.6.2.2 Network Structure The next experiment is designed to investigate the impact of network size when using a TCIM representation. This experiment is set up to match the second MCIM experiment from Section 5.5.4, except a TCIM was used to model the child node in the network. The results are shown in Figure 5.16.

As shown by Figure 5.16a, the compaction ratio initially decreases, and then steadily increases as the size of the networks increase. This is likely because the TCIM representation relies on pruning portions of the tree, and the unstructured nature of the data makes it unlikely that large branches will be pruned when networks are large.

The reduction in variance indicates that the tree structure is consistently providing a small amount of compaction for large networks. In terms of the error introduced by the compact representation, Figure 5.16b shows no statistical significance between the two models, although just as shown by the MCIM experiments, there may be a small error introduced by the constant approximation threshold.

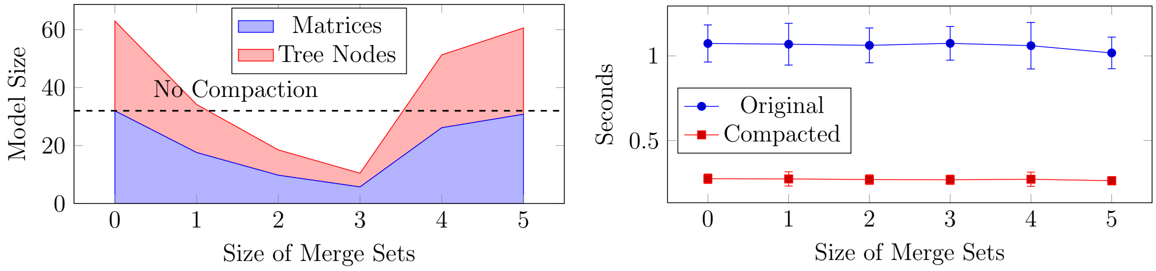
Despite the minimal improvements to space requirements, the efficient tree structure provides substantial improvements in terms of inference times. Figure 5.16c shows the runtimes for inference over a TCIM compacted model as well as a baseline model with no compaction. While the runtimes for the uncompacted models increase exponentially with the network size, the compacted model times appear almost linear. In addition, the variance for inference over the original model increases as shown by the error bounds, while the variance for the compacted model remains essentially negligible.

5.6.3 Structured Data Experiments

The structured data experiments in this section use models designed as described previously in Section 5.5.4. Specifically, random models are generated that are then modified to contain approximate equivalence using the notion of merge sets. The number of merge sets and the size of the merge sets vary based on experiments.

5.6.3.1 Merge Set Size

The results of the merge set size experiment are shown in Figure 5.17. In this case, Figure 5.17a shows the size of the model in terms of both the number of intensity matrices (the lower blue region), and the number of nodes in the tree that store the intensity matrices (the upper red region). Another interpretation is that the lower region shows the number of leaf nodes in the tree, while the upper region shows the number of interior nodes. With a set size of zero, the number of leaf nodes match the baseline model, which makes sense given that



(a) Space analysis.

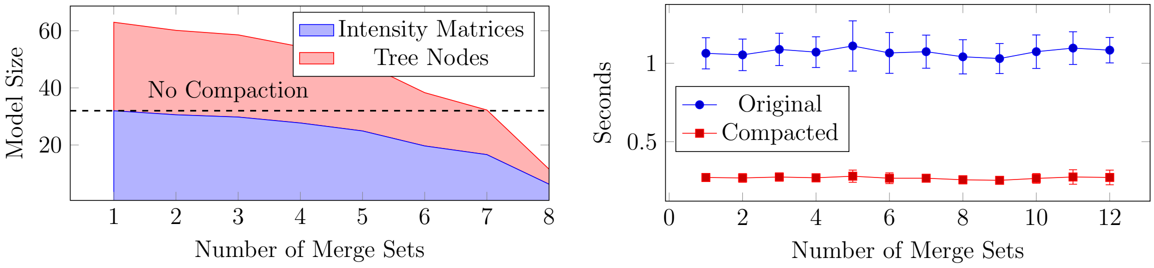
(b) Time analysis.

Figure 5.17: Impact of the size of merge sets with structured data using a TCIM representation.

there is no compaction occurring. Just as in the MCIM experiments, a set size of three produces the most compact models.

Figure 5.17b shows the runtimes for inference over the baseline and compacted models. Not only is inference over the TCIM compacted model substantially faster, the variance is reduced by switching to the compact representation as well, as indicated by the error bars. This improvement in inference speed does not appear to be affected by the merge set size, indicating that the efficiency of the TCIM representation overpowers any effect that model size has on inference runtimes.

5.6.3.2 Number of Merge Sets Just as with the MCIM experiments, the number of merge sets was varied from one to eight, to account for all possible state combinations of the three parents. Figure 5.18a shows the size of the model in terms of the number of intensity matrices and tree nodes. As the number of merge sets increase, the TCIM structure is able to encode increasingly more compact models. This follows the same trend as in the MCIM experiment, showing that in either case, the compact representations are more efficient when more states can be merged. Figure 5.18b shows runtimes for inference run over the original and compacted models. Once again, the TCIM compacted model results in substantially reduced runtimes



(a) Space analysis.

(b) Time analysis.

Figure 5.18: Impact of the number of merge sets with structured data using a TCIM representation.

with less variation. This indicates that the TCIM representation improves inference speeds, regardless of the type of structure exhibited in the data.

5.7 Comparison Between MCIMs and TCIMs

There are many different potential cluster possibilities when grouping intensity matrices. This section compares the representational capabilities of MCIMs and TCIMs by inspecting different types of clusterings. A graphical depiction of the results is shown in Figure 5.19. Before providing the theorems, we first present some formal terminology. A cluster is represented fully if the number of intensity matrices used in the structure is equal to the number of clusters. In other words, if a compact structure uses more intensity matrices than the number of clusters, then there are aspects of the clustering that are being omitted, meaning that it has not be represented fully. Alternatively, a clustering is represented exactly if is state instantiation maps to the correct cluster. Put another way, only the correct intensity matrices are used to describe the state instantiations. Furthermore, a non-trivial MCIM refers to a composition of functions containing a non-empty context set. Certainly a single mapping function with inputs $\mathbf{V}$ equal to the entire parent set $\mathbf{U}$ is

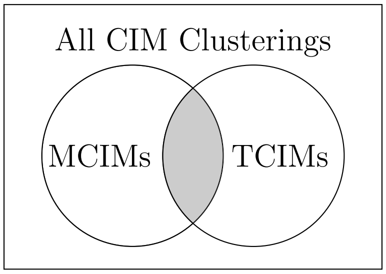


Figure 5.19: Venn diagram depicting CIM clusterings that can be represented using MCIMs and TCIMs. Theorems 2 and 3 prove that {MCIMs}/{TCIMs} and {TCIMs}/{MCIMs} are nontrivial. Theorem 1 shows that $\{\text{MCIMs}\} \cap \{\text{TCIMs}\}$ is nontrivial.

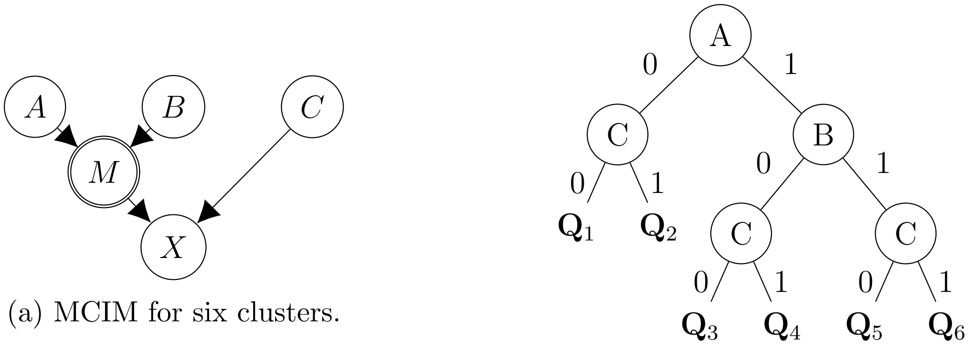
capable of full representing any possible clustering, although as discussed in Section 5.5, this type of representation is itself too complex to provide any computational benefits. As such, only non-trivial mappings are considered in the following.

Theorem 1. *There are clusterings that can be represented fully and exactly using either a non-trivial MCIM or a TCIM.*

Proof. To prove this claim, it is sufficient to provide a single example where this statement holds. Consider a network containing a node X and three parents $\mathbf{U} = \{A, B, C\}$, each with two states. A potential clustering of the CIM for X is as follows:

$$\begin{array}{r|ll}
C_1: & \mathbf{Q}_{X|\{a_0,b_0,c_0\}} & \mathbf{Q}_{X|\{a_0,b_1,c_0\}} \\
C_2: & \mathbf{Q}_{X|\{a_0,b_0,c_1\}} & \mathbf{Q}_{X|\{a_0,b_1,c_1\}} \\
C_3: & \mathbf{Q}_{X|\{a_1,b_0,c_0\}} & \\
C_4: & \mathbf{Q}_{X|\{a_1,b_0,c_1\}} & \\
C_5: & \mathbf{Q}_{X|\{a_1,b_1,c_0\}} & \\
C_6: & \mathbf{Q}_{X|\{a_1,b_1,c_1\}} &
\end{array} .$$

The original CIM consists of $2^3 = 8$ intensity matrices, which are then condensed down to six clusters. Figure 5.20a shows an MCIM that encodes the clustering, while



(a) MCIM for six clusters.

(b) TCIM for six clusters.

Figure 5.20: Compact structures for CIM clustering.

Figure 5.20b shows a TCIM that captures the same clustering. Here, the mapping variable M in the MCIM reduces the four state combinations of $\{A, B\}$ down to three, meaning that node X contains $3 \times 2 = 6$ intensity matrices, one for each of the clusters. Similarly, the tree structure has six leaf nodes corresponding to each cluster, indicating that these representations were in fact able to encode the entire clustering.

□

The shaded area between the circles in Figure 5.19 indicates clusterings that can be captured entirely by either non-trivial MCIMs or TCIMs.

Theorem 2. *There are clusterings that can be represented fully and exactly using a non-trivial MCIM, but not a TCIM.*

Proof. Again, a single example is sufficient. Consider the clustering presented in the previous section, but now assume that B has three states rather than two, resulting in four new matrices that do not belong to any of the previous clusters. This new clustering is shown below.

C_1:	$\mathbf{Q}_{X\|\{a_0,b_0,c_0\}}$	$\mathbf{Q}_{X\|\{a_0,b_1,c_0\}}$
C_2:	$\mathbf{Q}_{X\|\{a_0,b_0,c_1\}}$	$\mathbf{Q}_{X\|\{a_0,b_1,c_1\}}$
C_3:	$\mathbf{Q}_{X\|\{a_1,b_0,c_0\}}$	
C_4:	$\mathbf{Q}_{X\|\{a_1,b_0,c_1\}}$	
C_5:	$\mathbf{Q}_{X\|\{a_1,b_1,c_0\}}$	
C_6:	$\mathbf{Q}_{X\|\{a_1,b_1,c_1\}}$	
C_7:	$\mathbf{Q}_{X\|\{a_0,b_2,c_0\}}$	
C_8:	$\mathbf{Q}_{X\|\{a_0,b_2,c_1\}}$	
C_9:	$\mathbf{Q}_{X\|\{a_1,b_2,c_0\}}$	
C_{10}:	$\mathbf{Q}_{X\|\{a_1,b_2,c_1\}}$	

These clusters cannot be represented by any tree structure. This is because there are three states of B, that are not all equal in any context of A and C. The MCIM framework does not require that all states be equal and provides a method for consolidating a subset of states. In this example, the tree structure from Figure 5.20a is sufficient to describe the above clustering. Here, M maps state combinations $\{a_0, b_0\}$ and $\{a_0, b_1\}$ to a single state, and maps the remaining four state combinations to their own states. This results in a five state mapping variable, in combination with the unmodified two state variable C, resulting in a total of 10 possible state combinations for the parents of variable X. This matches the clustering described in this section where two of the original twelve intensity matrices in the network have been consolidated using the latent variable framework. A TCIM representation would require a full tree with no pruning, necessitating the storage of all twelve of the original intensity matrices.

□

This example suggests that the MCIM framework is better suited for representing CIMs when parent variables have a large number of states. As the number of states grows, it becomes increasingly less likely that all intensity matrices will be equal across all states for a given context. This means that TCIMs are likely to provide little or no benefit in these cases. An MCIM is capable of capturing equivalencies over a subset of the total number of states, which is a more reasonable requirement when there is a large number of states. The left circle in Figure 5.19 represents clusterings that may be represented using non-trivial MCIMs.

Theorem 3. *There are clusterings that can be represented fully and exactly using a TCIM, but not a non-trivial MCIM.*

Proof. In this case, the example starts with the same clustering from the proof for Theorem 1, except that now we assume clusters C_5 and C_6 are now close enough that they can be merged. The resulting cluster set is as follows, where $C_5 \approx C_6$.

$$\begin{array}{r|ll}
C_1\text{:} & \mathbf{Q}_{X|\{a_0,b_0,c_0\}} & \mathbf{Q}_{X|\{a_0,b_1,c_0\}} \\
C_2\text{:} & \mathbf{Q}_{X|\{a_0,b_0,c_1\}} & \mathbf{Q}_{X|\{a_0,b_1,c_1\}} \\
C_3\text{:} & \mathbf{Q}_{X|\{a_1,b_0,c_0\}} & \\
C_4\text{:} & \mathbf{Q}_{X|\{a_1,b_0,c_1\}} & \\
C_5\text{:} & \mathbf{Q}_{X|\{a_1,b_1,c_0\}} & \mathbf{Q}_{X|\{a_1,b_1,c_1\}}
\end{array}$$

This clustering cannot be represented using the MCIM framework. The consolidation of the matrices in clusters C_1 and C_2 can be achieved as described in the previous section, but this does not hold for the matrices in C_5. The issue is that the equivalence observed in C_5 does not hold across all contexts; therefore, they cannot be captured by mapping variables, except in the trivial case where a variable is introduced as a child for all parents A, B and C. Conversely, a TCIM can describe

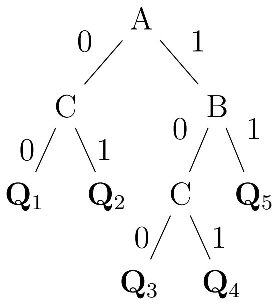


Figure 5.21: Tree-structured CIM

this clustering compactly without issue, as shown in Figure 5.21, where the five leaf nodes correspond to the five clusters, thereby providing a minimal representation.

□

This example suggests that a TCIM may work better for representing clusterings when there are a large number of parent variables. This is because large parent sets make it increasingly less likely that multiple instantiations will be equivalent across *all* contexts. As the number of parents grows, the size of the subject and context sets grows exponentially, likely making MCIMs a poor choice for networks with large parent sets. This shortcoming is not shared by tree structures, which are likely able to exploit context-based independence regardless of the number of parent nodes. The right circle in Figure 5.19 represents clusterings that can be represented using TCIMs.

Theorem 4. *There are clusterings that cannot be represented fully and exactly using either a non-trivial MCIM or a TCIM.*

Proof. The previous two cases may be combined to form an example of a clustering that cannot be represented by either MCIMs or TCIMs. Specifically, a clustering can be produced that contains a three state variable B, as well as a clustering of $\mathbf{Q}_{X|\{a_1,b_1,c_0\}}$ and $\mathbf{Q}_{X|\{a_1,b_1,c_1\}}$. The combined clustering is shown below.

$$\begin{array}{r|ll}
C_1\colon & \mathbf{Q}_{X|\{a_0,b_0,c_0\}} & \mathbf{Q}_{X|\{a_0,b_1,c_0\}} \\
C_2\colon & \mathbf{Q}_{X|\{a_0,b_0,c_1\}} & \mathbf{Q}_{X|\{a_0,b_1,c_1\}} \\
C_3\colon & \mathbf{Q}_{X|\{a_1,b_0,c_0\}} & \\
C_4\colon & \mathbf{Q}_{X|\{a_1,b_0,c_1\}} & \\
C_5\colon & \mathbf{Q}_{X|\{a_1,b_1,c_0\}} & \mathbf{Q}_{X|\{a_1,b_1,c_1\}} \\
C_6\colon & \mathbf{Q}_{X|\{a_0,b_2,c_0\}} & \\
C_7\colon & \mathbf{Q}_{X|\{a_0,b_2,c_1\}} & \\
C_8\colon & \mathbf{Q}_{X|\{a_1,b_2,c_0\}} & \\
C_9\colon & \mathbf{Q}_{X|\{a_1,b_2,c_1\}} &
\end{array}$$

The three state issue from Theorem 2 prevents complete representation of the clustering using a TCIM. Similarly, the combined matrices in cluster C_5 have the same issue seen in Theorem 3, in that the equivalence does not hold across all contexts. For this reason, an MCIM is unable to encode the clustering efficiently, unless all parent variables are included as inputs to the discrete mapping function. An MCIM representation of the clustering requires the storage of 10 intensity matrices, and a TCIM will need to store 11 matrices at the leaves. Although the MCIM appears to be a slightly better choice, neither representation is able to represent the minimal nine intensity matrices defined by the clusters. This demonstrates that clusterings exist that can only partially be captured using MCIMs and TCIMs.

□

The surrounding rectangle in Figure 5.19 represents the entire space of clusterings, and the area outside of the circles is the set of clusterings that cannot be represented with either non-trivial MCIMs or TCIMs.

5.8 Summary

In this chapter we introduced the MCIM and the TCIM: two approaches to representing conditional CTMPs compactly. The MCIM exploits the notion of context-independent equivalence, allowing the conditional CTMP to be factored using a composition of functions. Similarly, the TCIM takes advantage of context-specific independence and uses a tree structure to factor a conditional CTMP. In both cases, the compact structures make use of intensity matrix clustering as a preprocessing step, which allows for approximate compact representations as well as exact. Experiments were conducted that evaluated the capabilities of both the MCIM and TCIM, and it was proved that the classes of conditional CTMPs that can be represented by MCIMs and TCIMs intersect, but are not equivalent. These two compact representations provide a means of combating scalability concerns, especially when data exhibits strong context-independent equivalence or context-specific independence.

When working with models that contain a variable with many parents, the exponential number of intensity matrices can be intractable. This is because CTBNs are only able to factor the parameters using conditional independence between variables, and fail to capture other forms of independencies. The MCIM and TCIM structures presented in this chapter allow additional structure to be imposed that combats the complexity issues. By providing these compact structures, it is possible to model processes that would have otherwise been too large to be represented using CTBNs. The MCIM and TCIM representations will serve as valuable tools for those applying CTBNs to highly interdependent, structured datasets.

CHAPTER SIX

CONTINUOUS TIME DISJUNCTIVE INTERACTION

As discussed in the previous chapter, the driving factor in the complexity of a CTBN is the size of the parent sets for each variable. The last chapter suggested MCIMs and TCIMs as techniques for addressing this issue. These compact representations both work by consolidating equivalent or similar intensity matrices, reducing the total number of parameters required to specify a CTBN. This chapter pursues the same goal of compact representation but takes a distinctly different approach to solving the problem. The concept of disjunctive interaction is introduced for CTBNs, allowing for a substantial reduction in the number of required parameters, so long as assumptions hold regarding parent-child relationships. The techniques presented in this chapter are analogous to the Noisy-OR model in BNs, generalized to allow for non-Boolean variables.

6.1 Background

This section describes disjunctive interaction as it exists in the context of BNs. Disjunctive interaction, also referred to as the Noisy-OR gate or the Noisy-Max gate, is a generalized version of the logical OR gate, capable of capturing the non-determinism in a system with disjunctive interactions. In a traditional OR gate, there are n Boolean inputs $u_1, \ldots, u_n$ and a single Boolean output x. The domain for each u_i and x consists of the states *False* and *True*, which we denote using 0 and 1 respectively. The output x takes on a value of 1 if at least one of the inputs u_i is in state 1.

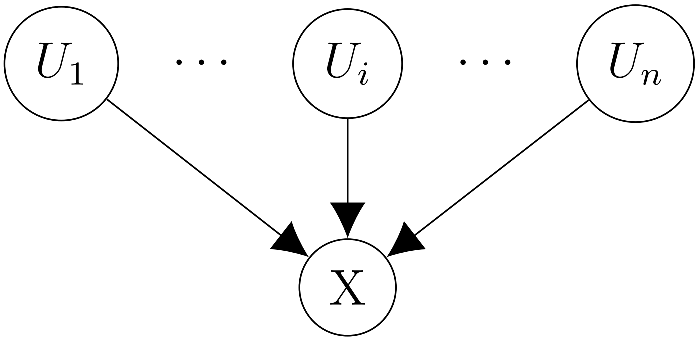


Figure 6.1: Generic Parent-Child Relationship

The Noisy-OR model can also be described generally as an OR gate with inputs $u_1, \ldots, u_n$ and an output of x. The BN structure is shown in Figure 6.1, where the inputs are represented using nodes $U_1, \ldots, U_n$, and the output is the node X. A lowercase u_i is used to refer to an instantiation of node U_i, and x is considered to be an instantiation of X. We use u_i^s to represent the specific instantiation where U_i is in state s. Without loss of generality, let $\tilde{\mathbf{u}}_i$ denote the specific set of parent node instantiations $\{u_0^0, \cdots, u_{i-1}^0, u_i^1, u_{i+1}^0, \cdots, u_n^0\}$, such that only node U_i has a value of 1, while the remaining variables in $\mathbf{U}$ are set to 0.

We say that if a single parent U_i is in state 1, it will "cause" X to be 1 with probability λ_i and 0 with probability $(1 - \lambda_i)$, as shown in the following:

$$P(X = 1|\tilde{\mathbf{u}}_i) = \lambda_i$$
$$P(X = 0|\tilde{\mathbf{u}}_i) = (1 - \lambda_i).$$

The probability that X is in state 0 given its entire parent set $\mathbf{U}$ is calculated as the product of the probabilities $P(X = 0|\tilde{\mathbf{u}}_i)$ for each U_i that is in state 1. We use $\tilde{\mathbf{u}}^+$ to denote the set of indices for parents that are in state 1, as shown below.

$$\begin{aligned} P(X=0|\mathbf{U}) &= \prod_{i \in \tilde{\mathbf{u}}^+} P(X=0|\tilde{\mathbf{u}}_i) \\ &= \prod_{i \in \tilde{\mathbf{u}}^+} (1-\lambda_i). \end{aligned}$$

If all parents $\mathbf{U}$ are in state 0, then there are no causes that can effect an event in X and $P(X=0|\mathbf{U}) = 1.0$. If this behavior is not desired, an additional parent referred to as a leak node can be added to represent unmodeled causes. We can use this to calculate the probability that X is in state 1 by taking 1 minus this probability, as follows:

$$\begin{aligned} P(X=1|\mathbf{U}) &= 1 - P(X=0|\mathbf{U}) \\ &= 1 - \prod_{i \in \tilde{\mathbf{u}}^+} (1-\lambda_i). \end{aligned}$$

As a result, the number of parameters required to populate the CPT is reduced. Rather than being exponential in the number of parents, we need only a linear number of parameters, equating to only the cases where a single parent is in state 1, and the case where all parents are in state 0 when a leak probability is desired. The reduction in the number of required parameters streamlines the process of creating a BN when the underlying system consists of disjunctive interactions. This allows for the representation of systems that would otherwise have been intractably large.

6.2 Continuous Time Disjunctive Interaction

Although the CTBN is representationally powerful, it is incapable of scaling to larger models that contain nodes with a large number of parents. The space complexity of the system primarily depends on the node with the largest number of parents, with all other nodes contributing potentially far less to the space requirements of the model. With this in mind, a generic network containing a single child node and n parent nodes is introduced in this section for demonstrative purposes. This network can be thought of as a standalone CTBN, or as a subnetwork in a larger model. The graph structure for this network is shown in Figure 6.1.

The number of homogeneous intensity matrices required to describe a CIM for node X is as follows:

$$|\mathbf{Q}_X|\mathbf{U}| = \prod_{i=0}^{n} |U_i|. \tag{6.1}$$

It is this exponential number of intensity matrices that prevents model scalability. CTBNs provides a more compact representation of a CTMP by using conditional independence between variables. Similarly, as discussed in Chapter refch:compact, it is possible to exploit special characteristics of the data in a CTBN to provide an even more compact representation. This provides a method for avoiding the exponential space complexity for cases where this data characteristic applies.

Disjunctive interaction between the variables in a CTBN can be used to simplify model representation. Let X be a node in a CTBN, and let $\mathbf{U}$ be the set of parents for that node. We say that there is a disjunctive interaction among the parents of X if the assumptions of *accountability* and *causation independence* hold. We describe these two assumptions as they exist in a temporal context with an arbitrary number of variable states, which varies slightly from the traditional definitions provided by Pearl [92].

The accountability assumption refers to the notion that a transition between the states of X may only occur as a result one of the parents in $\mathbf{U}$. In other words, if each of the parents in $\mathbf{U}$ are expected to cause a zero transition rate from a state x_i to x_j, then the final transition probability should be zero as well since there has been no cause to explain the transition. If it is desired for X to have some rate of transitioning despite the absence of a cause, an additional node can be added synthetically to the parent set of X that represents the notion of unmodeled causes. This synthetic node is referred to as a "leak" node in that it captures all other possibilities and explicitly represents otherwise implicit behavior.

The assumption of causation independence indicates that if a transition event from x_i to x_j occurs, the mechanisms that cause such a transition are independent from one another. This means that any parent $U \in \mathbf{U}$ is sufficient to produce transition rates between the states of a variable X on its own, without the assistance of any other parent. Furthermore, if multiple parents of X are expected to cause a transition from x_i to x_j, then when considered simultaneously, they will cause the transition to occur at a higher rate than when considering the parents individually.

If these assumptions hold, then we say that there is a disjunctive interaction for the parents of X. When this is the case, the node X can be parameterized more concisely than a typical node in a CTBN. The insight that allows for the reduction in parameters is the notion that X depends on each of its parents independently, rather than simultaneously. To formalize this idea, we introduce the notion of a *Disjunctive Conditional Markov Process*, which we formalize in the following definition.

Definition 6.2.1. *A Disjunctive Conditional Markov Process (DCMP) X is an inhomogeneous Markov process whose intensity matrix varies with time, but only as a function of the current value of a discrete conditioning variable U drawn from a set of variables $\widehat{\mathbf{U}}$, where the hat notation indicates that the set of variables exhibits disjunctive interaction. Its intensity matrix, called a Disjunctive Conditional Intensity Matrix (DCIM), is written $\mathbf{Q}_{X|\widehat{\mathbf{U}}}$ and can be viewed as a set of homogeneous intensity matrices $\mathbf{Q}_{X|u}$ – one for each instantiation u to U, for each variable U in $\widehat{\mathbf{U}}$.*

6.2.1 Boolean Disjunctive Interaction

In this section, we restrict our analysis of the model to only consider the case where variable are Boolean. The notion of disjunctive interaction that was first proposed for BNs assumed that all nodes in the network were Boolean, where each state had an explicit meaning [92]. Specifically, state one is considered to be the presence of an event, while state zero can be thought of as the absence of an event. The relationship between a parent and its child can then be interpreted as causal, where an event's occurrence in a parent will cause an event to occur in a child node as well with some probability. This interaction acts as an OR gate in that one or more parents are needed to cause an event in the child node. Furthermore, if no events have occurred in the parent nodes, then no event will occur in the child. The model is referred to as the Noisy-OR gate because there is a non-zero probability that an event fails to occur in the child node, despite an event occurring in a parent.

This section explores disjunctive interaction in CTBNs for the special case where a child node and all of its parents have exactly two states. First, note that the CIM for node X consists of 2^n homogeneous intensity matrices, where n is the number of parents for node X. Each of these homogeneous intensity matrices has four entries, equating to two unique rates. An example of such an intensity matrix is shown below.

$$\mathbf{Q}_{X|\mathbf{u}} = \begin{array}{c} \\ 0 \\ 1 \end{array}\!\!\begin{array}{c} \begin{array}{cc} \quad 0 & \qquad 1 \end{array} \\ \begin{pmatrix} -\lambda_{\mathbf{u}} & \lambda_{\mathbf{u}} \\ \mu_{\mathbf{u}} & -\mu_{\mathbf{u}} \end{pmatrix} \end{array}$$

The parameters in these matrices describe the rate at which the variable is expected to transition from one state to another. Using this, the probability density function f and the cumulative distribution function F can be defined for transitioning from state x_0 to state x_1.

$$f_{\lambda_{\mathbf{u}}}(t) = \lambda_{\mathbf{u}} \exp(-\lambda_{\mathbf{u}} t)$$
$$F_{\lambda_{\mathbf{u}}}(t) = 1 - \exp(-\lambda_{\mathbf{u}} t)$$

The cumulative distribution function $F_{\lambda_{\mathbf{u}}}(t)$ defines the probability of transitioning from state x_0 to state x_1 by time t. This means that the probability that the process has remained in state x_0 until time t is $\exp(-\lambda_{\mathbf{u}} t)$.

To replicate the event causation interpretation from the BN literature, restrictions can be applied to the rates of the intensity matrices for X. First, it is assumed that once an event occurs, the process cannot revert back to an absence of an event. This requirement can be handled by forcing each rate $\mu_{\mathbf{u}}$ to equal 0.0. This guarantees that state x_1 is an absorbing state, meaning that an event will only occur once and stay in that state for the remainder of the process. The second restriction that must be applied is that $\lambda_{\mathbf{u}} = 0.0$ when each $u \in \mathbf{u}$ is in state u_0. This ensures that if no event has occurred in the parent nodes, then there is no cause to explain an event in X, and the probability that $X = x_1$ is 0.0. Finally, the initial distribution for X is defined as $P = (1.0, 0.0)$, which forces X to start deterministically in state x_0 at the starting time.

The DCIM model may now be used for the described Boolean CTBN. Rather than requiring 2^n intensity matrices, a DCIM needs only $2n$. This corresponds to the two separate intensity matrices required for each state of a parent node, which is necessary for all n parents. Using the available $2n$ intensity matrices in the DCIM, it is possible to compute any intensity matrix in the CIM according to the parent state assignment $\mathbf{u}$. The probability that multiple causes fail to produce an event in X is the product of the probabilities that each cause fails to produce an event independently. For more information, refer to Pearl's work on Noisy-OR [92]. Using this property, along with the cumulative distribution function, an equation can be derived that defines the rates of a CIM in terms of the rates in a DCIM.

$$\begin{aligned}
P(X(t) = x_1) = &\quad 1 - P(X(t) = x_0) && \text{(6.2a)}\\
1 - \exp(-\lambda_{\mathbf{u}} t) = &\quad 1 - \prod_{u \in \mathbf{u}} \exp(-\lambda_u t) && \text{(6.2b)}\\
\exp(-\lambda_{\mathbf{u}} t) = &\quad \prod_{u \in \mathbf{u}} \exp(-\lambda_u t) && \text{(6.2c)}\\
-\lambda_{\mathbf{u}} t = &\quad \sum_{u \in \mathbf{u}} -\lambda_u t && \text{(6.2d)}\\
\lambda_{\mathbf{u}} t = &\quad t \sum_{u \in \mathbf{u}} \lambda_u && \text{(6.2e)}\\
\lambda_{\mathbf{u}} = &\quad \sum_{u \in \mathbf{u}} \lambda_u && \text{(6.2f)}
\end{aligned}$$

The derivation starts on Line 6.2a with a basic equality showing that the probability of X being in each of the two states sums to 1.0. Next, Line 6.2b shows the cumulative distribution function using a rate $\lambda_{\mathbf{u}}$ for a CIM is shown on the left. An alternative representation is shown on the right side of the equation, where the probability of remaining in state x_1 is replaced with the product of probabilities for remaining in state x_1 given each parent individually. This produces an equality that relates the rate in a CIM to the rates in a DCIM. Line 6.2c subtracts 1 from both sides and multiplies the resulting terms by -1. Line 6.2d takes the log of both sides, which also has the effect of changing the product to a sum. In Line 6.2e, -1 is multiplied by each side, and t is pulled out of the summation. Finally, note that the t value on the left side of the equation is equal to the t value on the right, indicating that this equality holds for all times t. As a result, the t on each side of the equation cancels, resulting in the equality shown in Line 6.2f.

This derivation indicates that a rate in a CIM can be computed by summing the rates contributed by individual parents. This validates the formulation presented in this chapter by demonstrating its equivalence to previous disjunctive interaction as it was proposed originally for BNs. Furthermore, if the rate assumptions made in this Boolean special case hold, then the computation of CIM rates can be simplified. First, it is known immediately that the rate of transitioning from x_1 to x_0 is always 0.0, meaning that no summation needs to be performed for these entries in the CIM. Next, it is assumed that if a parent is in state u_0, then it will not cause X to transition to state x_1. With the knowledge that these rates are equal to 0.0, the summation shown in Equation 6.2f may be simplified. Rather than summing over the entire set of parent instantiations $\mathbf{u}$, the sum can instead consider only those parents where $u = u_1$. This excludes the cases where $u = u_0$, which have a known contributing rate of 0.0. The Boolean rate may be computed more efficiently than a general CIM rate due to the potential reduction in the size of the summation.

6.2.2 Generalized Disjunctive Interaction

The previous section described disjunctive interaction in CTBNs for the case where all variables contain exactly two states. This section lifts that assumption to allow both the child and parent variables to incorporate any number of states. The approach that we take to generalizing disjunctive interaction is similar to that of Srinivas [**?**]. Without loss of generality, let X be a node in a CTBN with n parents, and assume that disjunctive interaction holds. A DCIM can be defined in terms of the more traditional CIM data structure by using a conceptual model. For example, Figure 6.2 shows n separate instances of X, one for each parent $U \in \mathbf{U}$. Although the actual CTBN structure contains only a single instance of X, this figure serves as an aid for conceptualizing DCIMs. Intuitively this structure conforms to the

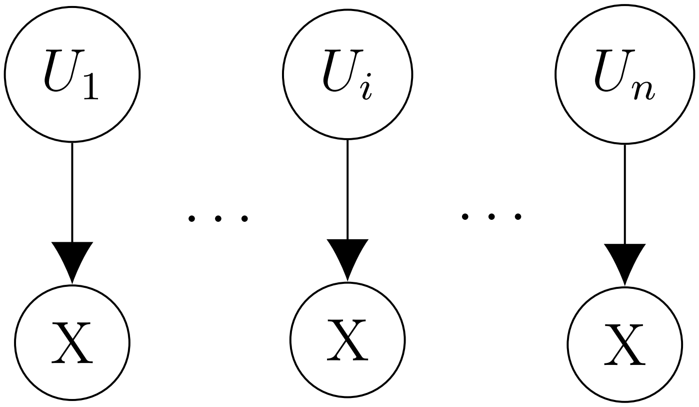


Figure 6.2: Conceptual disjunctive network structure.

idea of disjunctive interaction in that each of the parents are now able to influence the behavior of X independently without the interference of another parent. Using traditional CIM parameterizations for each of these disjoint instances of X, each $\mathbf{Q}_{X|U}$ requires $|U|$ homogeneous intensity matrices corresponding to the states of each parent variable U. The DCIM for a node X with parents $\widehat{\mathbf{U}}$ can be thought of as the union of the CIMs required for each X in the conceptual model. This is shown formally in the following equation, where each intensity matrix $\mathbf{Q}_{X|U}$ is conditioned on only a single variable U.

$$\mathbf{Q}_{X|\widehat{\mathbf{U}}} = \bigcup_{U \in \mathbf{U}} \mathbf{Q}_{X|U}. \tag{6.3}$$

For a more concrete example, refer back to the drug effect network in Figure 2.3. The `JointPain` variable was modeled as a conditional Markov process, dependent on both `Barometer` and `Concentration`. By assuming disjunctive interaction between the causes of the `JointPain` variable, it is assumed that the barometric pressure will affect joint pain without the assistance of drug concentration, and vice versa. In other words, a specific combination of barometric pressure and drug concentration will not have a unique influence on joint pain other than the existing contribution made by `Barometer` and `Concentration` individually. If this assumption holds, then

`JointPain` can instead be modeled using a DCMP. Recall from our example that `Barometer` consists of three states while `Concentration` has two. Then the set of homogeneous intensity matrices that make up the DCIM for `JointPain` is as follows:

$$\mathbf{Q}_{J|B,C} = \{\mathbf{Q}_{J|b0}, \mathbf{Q}_{J|b1}, \mathbf{Q}_{J|b2}, \mathbf{Q}_{J|c0}, \mathbf{Q}_{J|c1}\}.$$

Note that there are five intensity matrices in this set, compared to the six intensity matrices necessary for the original `JointPain` CIM. This is because a DCIM considers the state of each parent on an individual basis, rather than using state assignments to all parents simultaneously. As a result, the number of homogeneous matrices required to specify a DCIM are the summation of the number of states for each parent variable, as shown here.

$$|\mathbf{Q}_{X|\widehat{\mathbf{U}}}| = \sum_{i=0}^{n} |U_i| \tag{6.4}$$

This follows directly from Equation 6.3, which constructs the set of intensity matrices using a union operation. A viable interpretation of disjunctive interaction is that each of the parents acts as an independent cause of a transition in the child. This means that identifying the next transition in state X reduces to the task of taking the minimum of a set of independent random variables that are exponentially distributed with rates defined for each of the parents. Each of these random variables is a transition event, and the earliest of these events will be exponentially distributed with a rate equal to the sum of the parent contributions. This is analogous to how a diagonal entry in a row of an intensity matrix defines the earliest transition time for all other states by summing the remaining columns.

Comparing Equations 6.1 and 6.4, we see that the number of intensity matrices required to specify a DCIM is a summation, while a CIM is a product over the same terms. This change from a product to a summation is the cause for the reduction in the number of parameters when using DCIMs. For instance, in the joint pain example where there are two parents with 2 and 3 states, a CIM will require $3 * 2 = 6$ homogeneous intensity matrices, while a DCIM will require only $3 + 2 = 5$. As we consider models with a larger number of parents, or parents with a larger number of states, the savings provided by the DCIM representation becomes increasingly evident.

6.3 Converting DCIMs to CIMs

Although DCIMs are capable of representing disjunctive interaction concisely, existing algorithms designed for CTBN learning and inference are dependent on the CIM framework. To ensure DCIMs are compatible with these algorithms, we require a method for converting a DCIM to a CIM. The concern however, is that if a DCIM is simply replaced with a CIM, the representational advantages afforded by disjunctive interactions are lost. Instead, we take advantage of the fact that algorithms typically only use a small subset of the parameters in a CIM at one time. To facilitate access to specific parts of a CIM, DCIMs can be used to derive only the information that is necessary. This allows algorithms access to a CIM implicitly, despite the fact that no explicit representation exists.

To derive a CIM from a DCIM, we make use of the properties of the underlying disjunctive interaction. As a reminder, each parent U of a variable X is considered sufficient to explain the behavior of X if the accountability and causal independence assumptions hold. As a result, each parent U in some state u will cause X to transition from state x_i to x_j with a rate of $\mathbf{Q}_{X|u}[i, j]$. When considering every parent $U \in \mathbf{U}$, we are left with $n = |\mathbf{U}|$ transition events, each of which are independent exponentially distributed random variables. The rate at which we expect X to transition from x_i

to x_j given a full state instantiation $\mathbf{u}$ to its parents $\mathbf{U}$ is therefore the minimum of the transition times obtained from each parent individually. Given that the events are independent and exponentially distributed, the rate of the minimum transition time for a full instantiation of the parents is as follows.

$$\mathbf{Q}_{X|\mathbf{u}}[i,j] = \sum_{u\in\mathbf{u}} \mathbf{Q}_{X|u}[i,j] \tag{6.5}$$

The property that allows the individual transition rates to be summed is the same property that ensures that the diagonal entry in the row of an intensity matrix encodes the sojourn time for that state. Recall that the diagonal entry in the row of an intensity matrix is the negative sum of the remaining entries for the row, and that the absolute value of the diagonal represents the rate at which the variable will transition to any of the other states. This is directly analogous to how a combination of parents cause a transition event in a child with a rate equal to the sum of their individual effects. Each parent can be thought of as contributing to the overall transition rate, and due to the accountability assumption discussed in the previous section, the entirety of a transition rate must be accounted for by these parent contributions.

Using Equation 6.5, each rate $q_{i,j}$ for a homogeneous intensity matrix in a CIM can be computed using the rates in a DCIM. This applies to any of the entries in an intensity matrix for a CIM, including the negative entries along the diagonal. Given the summation applies to each of the individual entries of a matrix, entire intensity matrices in a CIM can be computed at once using matrix addition. Specifically, $\mathbf{Q}_{X|\mathbf{u}} = \sum_{u\in\mathbf{u}} \mathbf{Q}_{X|u}$, where $\mathbf{Q}_{X|\mathbf{u}}$ is a homogeneous intensity matrix in the CIM $\mathbf{Q}_{X|\mathbf{U}}$, and $\mathbf{Q}_{X|u}$ is a homogeneous intensity matrix from the DCIM $\mathbf{Q}_{X|\widehat{\mathbf{U}}}$. Note that while summing the matrices as a whole may be advantageous in some cases, it will likely be more efficient to sum the individual rates when only a subset of an intensity matrix is required for an algorithm to operate.

To better conceptualize how CIM rates are generated dynamically from rates in a DCIM, we can define a CIM alternatively in terms of functions rather than statically stored transition rates. Consider an arbitrary homogeneous intensity matrix $\mathbf{Q}_{X|\mathbf{u}}$ from the CIM $\mathbf{Q}_{X|\mathbf{U}}$. Each of of the entries in the matrix can be replaced with a call to the following function f_X which is taken directly from Equation 6.5, and returns an intensity corresponding to the indices provided as input.

$$f(X, \mathbf{U}, i, j) = \sum_{u \in \mathbf{u}} \mathbf{Q}_{X|u}[i, j] \tag{6.6}$$

The key concept is that an intensity for a position in the matrix does not exist until it is computed by the function. Rather than storing the rates directly, a reference to a function that enables the rate to be computed is provided. After a rate is computed, the value can be discarded and recomputed at a later time if necessary. Here is the CIM with function entries.

$$\mathbf{Q}_{X|\mathbf{u}} = \begin{array}{c} \\ x_1 \\ x_2 \\ \vdots \\ x_n \end{array} \begin{array}{c} \begin{array}{cccc} x_1 & x_2 & \cdots & x_n \end{array} \\ \begin{pmatrix} f(X, \mathbf{u}, 1, 1) & f(X, \mathbf{u}, 1, 2) & \cdots & f(X, \mathbf{u}, 1, n) \\ f(X, \mathbf{u}, 2, 1) & f(X, \mathbf{u}, 2, 2) & \cdots & f(X, \mathbf{u}, 2, n) \\ \vdots & \vdots & \ddots & \vdots \\ f(X, \mathbf{u}, n, 1) & f(X, \mathbf{u}, n, 2) & \cdots & f(X, \mathbf{u}, n, n) \end{pmatrix} \end{array}$$

This representation provides a means of representing a larger set of parameters with a smaller set of parameters and a mapping function that exploits the disjunctive interaction in the model.

6.4 Approximate Inference with Disjunctive Interaction

Importance sampling can be used to achieve inference over a CTBN that contains nodes that are parameterized using DCIMs. Trajectories σ can be broken into segments $\sigma[i]$ that are partitioned based on when variables transition between states. The likelihood of a trajectory is decomposable by time, and is therefore calculated by multiplying the likelihood contributions for each segment of the trajectory. The likelihood for a trajectory is as follows:

$$L'(\sigma) = \prod_{u}\prod_{x}\left(q_{x|\mathbf{u}}^{M[x|\mathbf{u}]}\exp(-q_{x|\mathbf{u}}T[x|\mathbf{u}])\right)\prod_{x'\neq x}\theta_{xx'|\mathbf{u}}^{M[x,x'|\mathbf{u}]}$$

where $T[X|\mathbf{u}]$ is the amount of time spent in state x, $M[x,x'|\mathbf{u}]$ is the number of transitions from x to x', and $M[x|\mathbf{u}]$ is the total number of transitions from x to any other state. These sufficient statistics are particularly simple to obtain from the trajectory segment. $T[X|\mathbf{u}]$ is simply the duration of the segment, and $M[x|\mathbf{u}]$ is guaranteed to be zero since there are no transitions in the segment by construction. Given this, the likelihood can be rewritten specifically for a trajectory segment.

$$\tilde{L}'(\sigma) = \prod_{\mathbf{u}}\prod_{x}(q_{x|\mathbf{u}}^{0}\exp(-q_{x|\mathbf{u}}(t_e - t))\prod_{x'\neq x}\theta_{xx'|\mathbf{u}}^{0} \tag{6.7}$$

$$= \prod_{\mathbf{u}}\prod_{x}(\exp(-q_{x|\mathbf{u}}(t_e - t)) \tag{6.8}$$

Importance sampling works by sampling from a proposal distribution P' that guarantees the resulting trajectories conform to evidence, rather than the target distribution $P_{\mathcal{N}}$. This is achieved by sampling from truncated exponential distributions that force the necessary state transitions to occur prior to periods of evidence. The sampled trajectories are then weighted to account for their being drawn from the proposal distribution. The calculation of the weight for a sample is shown below, where **e** is the set of evidence indicating known states for specified time durations.

$$w(\sigma) = \frac{P_{\mathcal{N}}(\sigma, \mathbf{e})}{P'(\sigma)}$$

In order to compute likelihoods in the presence of disjunctive interaction, the sampling process must be adjusted slightly. With the introduction of DCIMs, the necessary rate parameters from Equation 6.7 are no longer readily available, and must be computed as they are required. The importance sampling algorithm can be extended to allow for disjunctive interaction by calling a function to obtain rates for transition times, as well as the multinomial distributions that define the relative likelihood of entering each state in the event of a transition. These rates for transition times and multinomial distributions may be computed using Algorithms 6.1 and 6.2 respectively, which either compute the intensities in the event that the node uses a DCIM representation, or draw the intensities directly from the corresponding intensity matrix if the node is using a standard CIM representation.

The pseudocode for importance sampling was originally presented by Fan *et al.*, which demonstrates how the likelihood and weight equations are built into the process of continuous time sampling [38,40]. A modified version of this procedure is shown in Algorithm 6.3, where asterisks to the right of lines indicate changes that were made to the original code. These changes include calls to either the `Get-Rate` procedure or the `Get-Multinomial` procedure, which compute the necessary rates on the fly using the summation from Equation 6.5. This differs from the original importance sampling algorithm which obtained these rates directly from the intensity matrices that were available. Algorithm 6.4 contains a subroutine used by importance sampling to assign

Algorithm 6.1: Get-Rate($x, \mathbf{u}$)

Input: Current state x of the node and instantiation $\mathbf{u}$ to parent variables.
Output: The rate λ for transitioning away from state x.

```
$q_{x|\mathbf{u}} \leftarrow 0$
if $X$ uses DCIM then
    for $u \in \mathbf{u}$ do
        $q_{x|\mathbf{u}} \leftarrow q_{x|\mathbf{u}} + \mathbf{Q}_{X|u}[x, x]$
    end for
else
    $q_{x|\mathbf{u}} \leftarrow \mathbf{Q}_{X|\mathbf{u}}[x, x]$
end if
return $q_{x|\mathbf{u}}$
```

Algorithm 6.2: Get-Multinomial($x, \mathbf{u}$)

Input: Current state x of the node and instantiation $\mathbf{u}$ to parent variables.
Output: The discrete distribution θ for transitioning to each state $x' \neq x$.

```
$\theta_{x|\mathbf{u}} \leftarrow \{0, \cdots, 0\}$                    // Vector of $|X|$ zeros
if $X$ uses DCIM then
    for $s \in X$, where $s \neq x$ do
        for $u \in \mathbf{u}$ do
            $\theta_{x|\mathbf{u}}[s] \leftarrow \theta_{x|\mathbf{u}}[s] + (\mathbf{Q}_{X|u}[x, s]/\mathbf{Q}_{X|u}[x, x])$
        end for
    end for
else
    for $s \in X$, where $s \neq x$ do
        $\theta_{x|\mathbf{u}}[s] \leftarrow \theta_{x|\mathbf{u}}[s] + (\mathbf{Q}_{X|\mathbf{u}}[x, s]/\mathbf{Q}_{X|\mathbf{u}}[x, x])$
    end for
end if
return $\theta_{x|\mathbf{u}}$
```

states to undefined variables. Algorithm 6.5 provides the updated pseudocode for the `Update-Weight` procedure, which is a subroutine of the importance sampling algorithm.

Other inference algorithms can be modified in a similar fashion, where the `Get-Rate` and `Get-Multinomial` procedures are called as rates from CIMs are needed. Disjunctive interactions allow for a more compact representation of CTBNs, regardless of the choice of inference algorithm. Furthermore, it is quite likely that large portions of a CIM will never need to be computed during the inference process. Consider sample-based inference, which essentially performs a random walk through state transitions. As the number of homogeneous intensity matrices in a CIM increases, it becomes increasingly less likely that a sample path will use every rate. This relates to the curse of dimensionality in that samples will not be sufficient to cover the entirety of a CIM. For this reason, learning and storing every possible transition rate for a CIM is often unnecessary, and may not be possible when considering memory limitations. This further emphasizes the importance of the DCIM representation and the role it plays during inference.

Note that the expected computational complexity of the algorithm has not been altered, but rather has been adapted to work with DCIMs using the same architecture designed for CIMs. The intent of the importance sampling algorithm presented here is to demonstrate how existing algorithms are able to work with the more compact DCIM representation without the need for significant alterations. Despite the fact that the theoretical bounds for computational complexity are not improved, the algorithm adapted to work with disjunctive interaction tends to result in smaller computation times when the number of parents is large, which is discussed in greater detail in Section 6.5.2. Future work will focus on developing inference algorithms that

Algorithm 6.3: DCIM-Importance-Sample($\mathbf{X}, \mathbf{e}, t_e$)

Input: CTBN over variables $\mathbf{X}$, possibly empty evidence set $\mathbf{e}$, and end time t_e.
Output: A complete trajectory σ over variables in $\mathbf{X}$ during the interval $[0, t_e)$.

```
$t \leftarrow 0$, $\sigma \leftarrow \emptyset$, $w \leftarrow 1$
for each variable $X \in \mathbf{X}$ do
    if $\mathbf{e}_X^{val}(0)$ defined then
        set $x(0) \leftarrow \mathbf{e}_X^{val}(0)$, and then set $w \leftarrow w \cdot \theta^{\mathcal{B}}_{x(0)|\mathbf{pa}_{\mathcal{B}}(0)}$
    else
        draw state $x(0)$ according to $\theta^{\mathcal{B}}_{X|\mathbf{pa}_{\mathcal{B}}(X)}$
    end if
end for

DCIM-Sample-Transition-Times($\mathbf{X}, \mathbf{e}, t, t_e$)

if $Time(X) \geq t_{end}$ then
    $w \leftarrow$ Update-Weight($X, w, t, t_{end}$)
    return $(\sigma, w)$
else
    $w \leftarrow$ Update-Weight($X, w, t, Time(X)$)
end if
Update $t \leftarrow Time(X)$

if $\mathbf{e}_X^{end}(t) \neq t$ or $\mathbf{e}_X^{val}(t)$ is defined then
    if $\mathbf{e}_X^{val}(t)$ is defined then
        set $x(t) \leftarrow \mathbf{e}_X^{val}(t)$
    else
        draw $x(t)$ from discrete distribution Get-Multinomial($x(t), \mathbf{u}_X(t)$)      *
    end if
    Add $\langle X \leftarrow x(t), t \rangle$ to $\sigma$
    Undefine $Time(X)$ and $Time(Y)$ for all variables $Y$ for which $X \in \mathbf{U}_Y$
else
    Undefine $Time(X)$
end if
```

Algorithm 6.4: DCIM-Sample-Transition-Times($\mathbf{X}, \mathbf{e}, t, t_e$)

Input: CTBN over variables $\mathbf{X}$, evidence set $\mathbf{e}$, current time t, and end time t_e.
Result: Assigns transition times to any undefined variables.

```
for each $X \in \mathbf{X}$ such that $Time(X)$ is undefined do
    if $\mathbf{e}_X^{val}(t)$ is defined then
        set $\Delta t \leftarrow \mathbf{e}_X^{end}(t) - t$
    else if $\mathbf{e}_X^{val}(t_e)$ is defined where $t_e = \mathbf{e}_X^{time}(t)$, $x(t) \neq \mathbf{e}_X^{val}(t_e)$ then
        draw $\Delta t$ from exponential with rate $\texttt{Get-Rate}(x(t), \mathbf{u}_X(t))$          *
    else
        draw $\Delta t$ from an exponential with parameter $\texttt{Get-Rate}(x(t), \mathbf{u}_X(t))$          *
    end if
    Define $Time(X) \leftarrow t + \Delta t$
end for
Let $X \leftarrow \arg\min_{X \in \mathbf{X}}[Time(X)]$
```

Algorithm 6.5: Update-Weight(Y, w, t_1, t_2)

Input: A variable Y, an existing weight value w, and start-end times t_1 and t_2.
Output: An updated weight w.

```
for each $X \in \mathbf{X}$ such that $\mathbf{e}_X^{val}(t)$ is defined for $t \in [t_1, t_2)$ do
    $w \leftarrow w \cdot \tilde{L}_X(x[t_1 : t_2])$
end for

for each $X \in \mathbf{X}$ such that $\mathbf{e}_X^{val}(t_e)$ is defined,
where $t_e = \mathbf{e}_X^{time}(t_1)$, and $x(t_1) \neq \mathbf{e}_X^{val}(t_e)$ do
    if $X = Y$ then
        $w \leftarrow w \cdot (1 - \exp(-\texttt{GetRate}(x(t_1), \mathbf{u}_X(t_1)) \cdot (t_e - t_1)))$
    else
        $w \leftarrow w \cdot \frac{1-\exp(-\texttt{GetRate}(x(t_1),\mathbf{u}_X(t_1))\cdot(t_e-t_1))}{1-\exp(-\texttt{GetRate}(x(t_1),\mathbf{u}_X(t_1))\cdot(t_e-t_2))}$          *
    end if
end for

return $w$
```

use the DCIM representation directly, rather than first translating the parameters to a CIM compatible form.

Finally, it is worth noting that this process is analogous to inference algorithms in BNs, except that rather than summing rates, BN algorithms multiply disjunctive conditional probabilities. The reason for this equivalence is the CTBN's reliance on the exponential distribution, where summing transition intensities is the temporal equivalent to multiplying fixed probabilities. Excluding algorithms such as *quickscore* which make use of specific network structures, inference in BNs in the presence of disjunctive interaction works in much the same way as described here, where the probabilities in a conditional probability table are computed at runtime as necessary.

6.5 Experiments

6.5.1 Expectation Comparisons

In this section, two experiments are presented that are intended to validate and explore the behavior of the proposed DCIM parameterization. The first experiment restricts the number of states for both the child and parent nodes to be Boolean, while the second experiment relaxes this restriction. These experiments compare the query results of a network parameterized using DCIMs with the expected transition probabilities. These experiments are designed to further justify the formulation of disjunctive interaction in CTBNs by supporting the theoretical results already provided. In all cases, the modified importance sampling algorithm presented in Section 6.4 was used, which performs on-the-fly calculations of the necessary rates using information encoded in DCIMs.

6.5.1.1 Binary State Nodes

For the first experiment, we constructed a network with four parents A, B, C and D, and a single child node X. The network structure

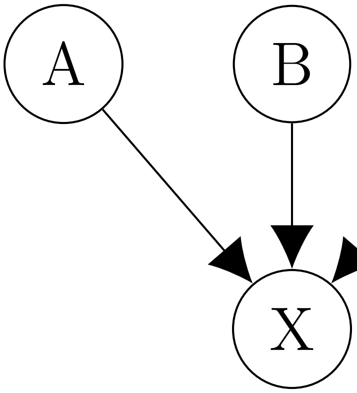


Figure 6.3: The network structure for the CTBN used in the expectation comparison experiments. Here, X is parameterized using DCIMs.

is shown in Figure 6.3. All nodes in the network are assumed to have two states, 0 and 1. Each of the parent nodes are parameterized with an initial distribution of $(0.5, 0.5)$ and a single intensity matrix containing a rate of 1.0 for transitioning to either state 0 or 1. The child node X is parameterized with an initial distribution of $(1.0, 0.0)$, and uses the disjunctive interaction model. This means that rather than using a CIM that would require a total of 16 intensity matrices, a DCIM is used to define transition behavior, consisting of only eight. The rate of X transitioning to state 1 given an instantiation of its parents is a random number drawn uniformly from the range $[0.0, 1.0]$. These rates are generated using a random number generator with a fixed seed to ensure that parameterization is consistent over numerous experiment runs. For each intensity matrix in the DCIM, the rate of transitioning from state 1 to state 0 is 0.0. The initial distribution for X ensures that the process starts deterministically in state 0, and the zero valued transition rates from state 1 to state 0 produce an absorbing state in state 1. This configuration ensures only a single transition from state 0 to state 1 will occur, at which point the process will remain in state 1 indefinitely. An example of the initial distribution for X and an intensity matrix in the DCIM for X is shown below. Here, λ_1 is a random value between 0.0

and 1.0, representing the transition intensity into state 1 from state 0.

$$P_X = [1.0, 0.0] \qquad\qquad \mathbf{Q}_{X|\mathbf{U}} = \begin{pmatrix} -\lambda_1 & \lambda_1 \\ 0.0 & 0.0 \end{pmatrix}$$

Node X has four parents with two states each, meaning that there are $2^4 = 16$ possible unique instantiations to these parents. For the sake of this experiment, ten of these instantiations were considered, randomly chosen to ensure a representative cross-section. For each instantiation of the parents, evidence was assigned to the model to guarantee that the parents conform to their corresponding states for the interval of time $t = [0.0, 2.0)$. The probability distribution over the states of X was then queried at 100 evenly spaced timesteps over this time interval. Next, the expected probabilities for the states of X were obtained by retrieving the probability of X having transitioned to state x_1 from an exponential distribution with a rate equal to the sum of the rates contributed by the corresponding intensity matrices in the DCIM.

To ensure that the inference algorithm adapted to work with DCIMs behaves as anticipated, we compare the results to the expected probabilities through time. Let $\Delta[x_1]$ be the mean difference between the probability obtained by running inference and the expected probability that was retrieved manually.

$$\Delta[x_1] = \Big(\sum_t \big|P(X(t) = x_1) - E[X(t) = x_1]\big|\Big)/100 \tag{6.9}$$

Here the summation accounts for each of the 100 evenly spaced intervals of $t \in [0.0, 2.0)$. This average difference serves as a metric for capturing how closely the inference algorithm is able to track the expected value over time. Table 6.1 shows $\Delta[x_1]$ for each of the ten parent instantiations considered for this experiment.

	$\Delta[x_1]$
$P(X\|a_1, b_0, c_0, d_1)$	0.0001
$P(X\|a_0, b_0, c_1, d_1)$	0.0008
$P(X\|a_0, b_1, c_0, d_0)$	0.0003
$P(X\|a_0, b_1, c_1, d_1)$	0.0002
$P(X\|a_0, b_0, c_0, d_0)$	0.0002
$P(X\|a_1, b_1, c_0, d_1)$	0.0008
$P(X\|a_1, b_0, c_0, d_0)$	0.0003
$P(X\|a_1, b_1, c_1, d_1)$	0.0002
$P(X\|a_1, b_0, c_1, d_0)$	0.0010
$P(X\|a_0, b_0, c_0, d_1)$	0.0006

Table 6.1: Average differences between expected probabilities and probabilities queried from a Boolean state network containing a DCIM-specified node.

Note that the observed values of $\Delta[x_1]$ are generally very small, indicating that inference closely matches the expected values throughout the entire time period $[0.0, 2.0)$. A paired equivalence test with a 0.005 region of similarity and a significance level of 0.05 was used to compare the queried result to the expected probability. For all cases, it was found that the probability obtained by querying the DCIM parameterized CTBN was statistically equivalent to the corresponding expected probability. What error does exist can be explained by our use of an approximate inference algorithm.

6.5.1.2 General State Nodes The previous experiment investigated the behavior of disjunctive interaction when a network contains Boolean state nodes only. This restricts the number of intensity matrices required to specify the child node, and mandates that each of these intensity matrices is of size 2×2. This Boolean restriction is now lifted, allowing nodes with any arbitrary number of states. This experiment makes use of the same network structure shown in Figure 6.3, but in this case we let $[A] = 2$, $[B] = 5$, $[C] = 3$, and $[D] = 10$. The number of states for the child node is also increased, such that $[X] = 4$.

With these added states, each intensity matrix for X now contains $4 \times 4 = 16$ entries rather than 4. In addition, a CIM would require $2 \times 5 \times 3 \times 10 = 300$ of these intensity matrices. This is drastically reduced by switching to a disjunctive representation, in that a DCIM requires only $2 + 5 + 3 + 10 = 20$ intensity matrices. This is a reduction of 280 intensity matrices, which, when compared to the reduction of eight matrices observed in the Boolean experiment, demonstrates the increasing utility of the disjunctive representation as the number of states increase.

Just as before, parent nodes are parameterized such that the initial probability for starting in each state is distributed evenly across all options, and all states transition to all other states with a rate of 1.0. The child node X is set to start deterministically in state x_0 at time $t = 0$, and may then transition to some state $x_i \neq x_0$ with a rate λ_i drawn from a uniform distribution over $[0.0, 1.0]$. Here again, these random values are drawn using a seeded random number generator to ensure consistency throughout the experiment. The initial distribution and an intensity matrix in the DCIM for node X is shown below, where Λ is the sum of the randomly drawn rates λ_1, λ_2 and λ_3.

$$P_X = [1.0, 0.0, 0.0, 0.0] \qquad\qquad \mathbf{Q}_{X|\mathbf{U}} = \begin{pmatrix} -\Lambda & \lambda_1 & \lambda_2 & \lambda_3 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{pmatrix}$$

Unlike the Boolean case, node X is now able to transition to several different states from its initial state x_0. To compute the expectation for each of these transitions, we again draw from an exponential distribution with a rate obtained by summing the contributing rates in the corresponding intensity matrices in the DCIM. This results in three exponential distributions, each of which can be thought

	$\Delta[x_1]$	$\Delta[x_2]$	$\Delta[x_3]$
$P(X\|a_0, b_0, c_1, d_0)$	0.0019	0.0024	0.0006
$P(X\|a_1, b_4, c_0, d_6)$	0.0004	0.0005	0.0011
$P(X\|a_1, b_2, c_0, d_8)$	0.0016	0.0024	0.0008
$P(X\|a_1, b_3, c_0, d_5)$	0.0008	0.0003	0.0010
$P(X\|a_0, b_1, c_2, d_7)$	0.0011	0.0017	0.0025
$P(X\|a_0, b_2, c_1, d_7)$	0.0012	0.0014	0.0027
$P(X\|a_0, b_1, c_1, d_2)$	0.0006	0.0013	0.0006
$P(X\|a_0, b_0, c_0, d_1)$	0.0073	0.0103	0.0028
$P(X\|a_0, b_1, c_2, d_5)$	0.0014	0.0010	0.0029
$P(X\|a_0, b_3, c_2, d_4)$	0.0013	0.0030	0.0017

Table 6.2: Average differences between expected and queried probabilities for for transitioning to states x_1, x_2 and x_3. X is parameterized using a DCIM.

of as "racing" one another to transition X into any one of the states x_1, x_2, or x_3. To account for the possibility that X may instead transition to another state first, the probabilities obtained from each exponential distribution in weighted by its rate as a fraction of the total sum of the transition rates.

Once more, ten different instantiations of the parent nodes were considered for this experiment, corresponding to evidence that was applied for individual inference runs. The difference between the expected probability and the queried probability for each state x_1, x_2 and x_3 is computed over 100 uniformly spaced time points in the range $[0.0, 2.0)$. Given that there are now three possible states for X to transition to, Δ can now be viewed as a vector of errors, indexed by each state of the variable X. The errors in this vector Δ are shown for the ten parent instantiations in Table 6.2.

The errors shown in Table 6.2 are generally larger than those observed in Table 6.1, but ultimately these errors still amount to fractions of a percent. Using the same paired equivalence test as the previous experiment, it was found that there was again no significant difference between the probabilities obtained by inference and the expected probability for each state. This points to the use of an approximate inference algorithm as the source of error, and the increase in error from the Boolean case is likely due to the increased model complexity. This experiment shows that disjunctive interaction behaves as expected, even in the case of non-Boolean variables.

6.5.2 Scalability

The previous experiments focused on validating the disjunctive formulation presented in this chapter. In this section, we present a series of experiments designed to investigate disjunctive interaction further. More specifically, we run inference over increasingly complex models in an attempt to gauge how well the DCIM approach scales.

For these experiments, a network with a single child node X, and n parent nodes $\mathbf{U}$ is used. All nodes in the network are assumed to have three states, meaning that a CIM for X will require 3^n, intensity matrices containing 9 entries each, while a DCIM will need only $3 \cdot n$. Each parent is parameterized using an initial distribution and intensity matrix as shown below:

$$P = \left[\frac{1}{3}, \frac{1}{3}, \frac{1}{3}\right] \qquad \mathbf{Q}_U = \begin{pmatrix} -\Lambda_0 & \lambda_{0,1} & \lambda_{0,2} \\ \lambda_{1,0} & -\Lambda_1 & \lambda_{1,2} \\ \lambda_{2,0} & \lambda_{2,1} & -\Lambda_2 \end{pmatrix}$$

where each $\lambda_{i,j}$ is a rate that was drawn from a uniform distribution over the range $[0.0, 1.0]$ and $\Lambda_i = \sum_{j \neq i} \lambda_{i,j}$. This is very similar to the parameterization of parent nodes in the previous expectation experiments, except that transitions make occur between states at varying rates.

Parameterization of the child node X is also modified from the one presented previously. In the expectation comparisons, it was required that X deterministically started in a single state and that all other states were absorbing. This ensured that X transitioned only once during the entire process, thereby allowing for the manual computation of state expectations. In this experiment we lift that restriction and allow X to be fully general. The initial distribution allows for an equal probability of starting in each of the three states, and all rates in the intensity matrices for the DCIM were drawn uniformly from the range $[0.0, 1.0]$. In other words, X is parameterized in exactly the same fashion as the parent nodes, except that rates are drawn to populate all $3 \cdot n$ intensity matrices, rather than just one.

For all scalability experiments, the number of parents is varied from one to twelve. The baseline run uses the modified inference algorithm presented in Section 6.4, which generates $100,000$ trajectories over the interval of time $[0.0, 2.0)$. Recall that generating these trajectories requires the use of CIM rates, which are not directly available for node X given the DCIM parameterization. Instead, CIM rates are computed dynamically as they are needed by the algorithm. For the purpose of this experiment, a recording is kept of each dynamically generated rate in the CIM for X throughout the inference process. The total number of unique rates that were computed is then compared to the total number of rates in the CIM, which can be obtained directly as 9×3^n. This provides a method for determining the portion of the CIM that was actually required to perform inference.

6.5.2.1 Number of Samples To determine how model complexity affects the portion of the CIM that was computed, we consider this portion as a function of the number of parent nodes in the graph. To investigate how the number of samples used in importance sampling impact these results, the number of samples was also varied. In addition to the baseline that makes use of $100,000$ samples, three other runs of the inference algorithm were considered that make use of $10,000$, $1,000$ and

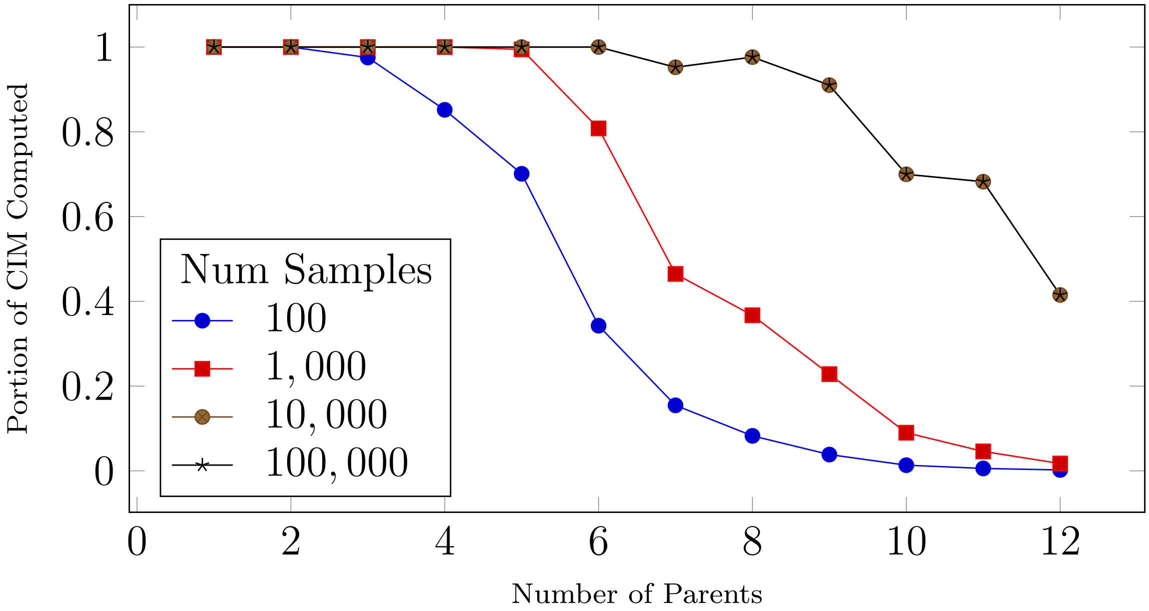


Figure 6.4: Plot of the portion of a CIM that is used during inference as a function of the number of parents in the graph. Each series represents inference that uses a different number of samples.

100 samples. Figure 6.4 plots the portion of the CIM computed during the four different inference runs as a function of the number of parents in the graph. Note that there is no perceptible change between inference that uses $10,000$ samples and one that uses $100,000$. As a result, the line for the $100,000$ sample is obscured in the figure.

The first notable feature of the plot is that the portion of the CIM that is computed decreases as the number of parents increase, regardless of the number of samples used during inference. This is due to the exponential increase in the number of intensity matrices that comprise the underlying CIM. When there is only one parent, there are three intensity matrices comprising the CIM, one for each state of the parent node. Throughout the sample generation process, it is very likely that all rates in these CIMs will be needed eventually, which is why 100% of the CIM is computed. As the number of parent nodes increase, so does the number of intensity matrices, resulting in a large number of rates for the CIM. At 12 parents, there are 3^{12} intensity matrices in a CIM for X, for a total of approximately 1.6 million rates. This makes it extremely unlikely that every rate will be used when sampling during inference, which explains the reduction in the portion of the CIM that is computed.

It should also be noted that the computed portion of the CIM decreases more quickly with fewer samples. This is because fewer samples implies fewer chances to generate unique rates of the CIM. In theory, as the number of samples approaches infinity, the portion of the CIM that is computed will approach 100%. Unfortunately as the number of parents approaches infinity, the portion of the CIM that is computed approaches 0%, creating competing influences. Furthermore, an increase in the number of samples produces only a small increase in the likelihood of computing the remaining rates, especially considering that some state instantiations to the parents may be very unlikely, suggesting a mixing problem. Conversely, the increased number of parameters is exponential in the number of parents, meaning that the influence imposed by the network structure will dominate the impact of the number of samples used during inference. In practice, as parent sets become large, only a small portion of the CIM will be used during inference, regardless of the number of samples chosen. This can be observed in Figure 6.4, where the increase from $10,000$ samples to $100,000$ samples has no perceivable impact on the portion of the CIM that was computed.

6.5.2.2 Time Period The next experiment investigates the impact that time has on the portion of the CIM that is computed. In all previous experiments, inference was computed over the interval of time $t = [0.0, 2.0)$. To determine the impact that this interval has on the portion of the CIM computed, the end time t_e is now varied. Additional endtimes of 1.5, 1.0, and 0.5 are considered, each of which are added as

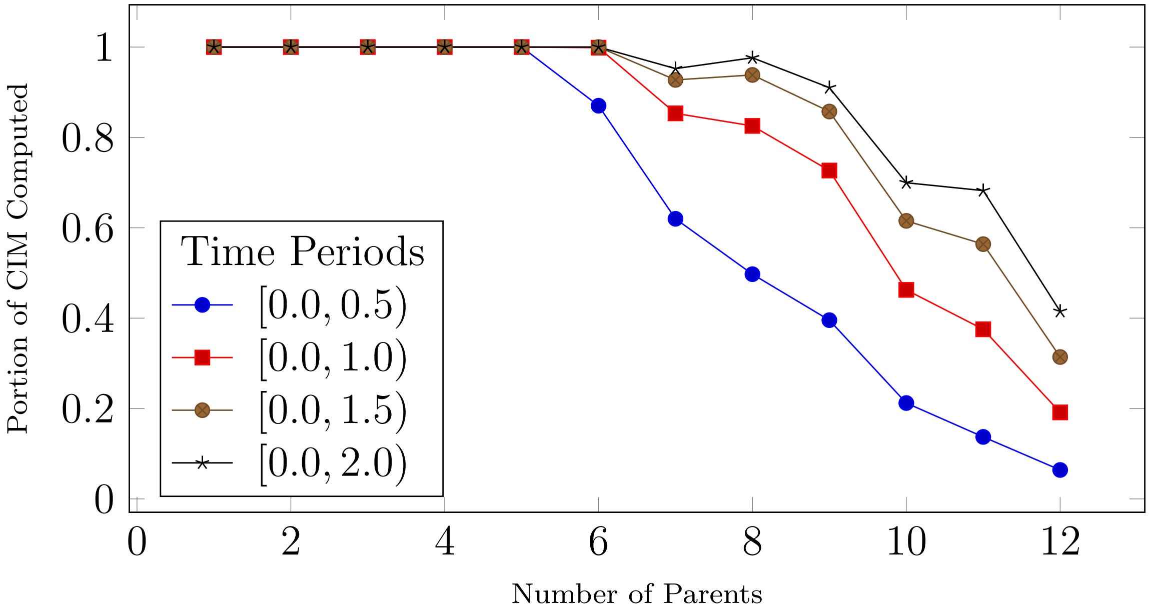


Figure 6.5: Plot of the portion of a CIM that is used during inference as a function of the number of parents in the graph. Each series represents a separate run of inference over a varying lengths of time.

separate series in the plot shown in Figure 6.5. All instances use a fixed 100, 000 of samples when performing inference.

First note that the series where inference is run until time $t = 2.0$ matches the series in Figure 6.4 with 100, 000 samples, since the experimental setup in both cases is identical. These can be thought of as the baseline case. Next note that as the end time t_e decreases, the decrease in the portion of the CIM computed occurs more rapidly. The reason for this is that earlier end times imply shorter time periods in general, which decreases the size of the sampled trajectories. These smaller trajectories contain fewer transitions, meaning fewer rate computations to accommodate these transition events. Just as before, however, ultimately the number of parents is more important than the size of the time period, in that the exponential increase in parameters cannot be matched by a linear increase in trajectory size. For a large number of parents, even very long trajectories will not require computation of the entire CIM.

6.5.2.3 Magnitude of Transition Rates In the final scalability experiment, the magnitudes of transition rates used when parameterizing nodes are altered. In previous experiments, each transition rate λ in an intensity matrix was drawn from a uniform distribution over the range $[0.0, 1.0]$. In this experiment, the range for the distribution is altered such that the maximum magnitude of a rate is limited to values 0.5, 0.25 and 0.1. These three new parameterization ranges, in addition to the baseline case where the maximum is 1.0, are each used to parameterize different models. As before, the portion of the CIM for node X that was computed during inference is recorded as a function of the number of parents, and the four different cases are shown as the four series in Figure 6.6. Although the rates are drawn from different distributions for each case, the process is still achieved using a seeded random number generator to guarantee consistency between runs. In this experiment, the number of samples used during inference is fixed at $100,000$, and the time period over which inference is computed is $[0.0, 2.0)$.

The baseline series where rates are constrained to a magnitude of 1.0 and below is equivalent to the $[0.0, 2.0)$ series from Figure 6.5, and the $100,000$ series from Figure 6.4. Here again, a decrease in the rates at which transitions occur results in a more rapid decrease in the portion of the CIM computed as the number of parents increase. Smaller rate values result in longer times between transitions. Given that trajectories encode transitions that occur over a fixed length of time, longer times between transition events imply fewer transitions overall. This decreases the probability of covering the entire CIM during the sampling process.

In each of the scalability experiments, it was shown that as the number of parent nodes increase, the portion of the underlying CIM that is actually used decreases. This behavior is exhibited regardless of attributes such as the number of samples used during inference, the time period considered, or the magnitude of the rates used

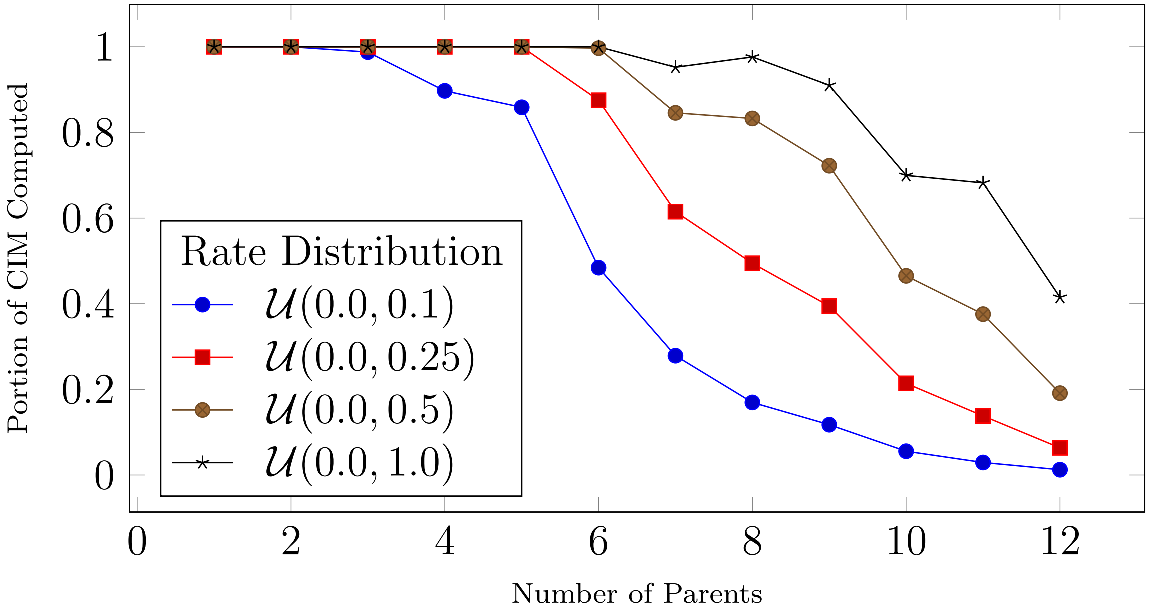


Figure 6.6: Plot of the portion of a CIM that is used during inference as a function of the number of parents in the graph. Each series represents a different model parameterized with varying magnitudes of transition intensities.

to parameterize intensity matrices. Despite amplifying or lessening the effect, changes to the model or inference algorithm do not significantly impact the overall trend that a CIM cannot be covered when the number of parents becomes very large. The issue is that the number of possible state instantiations to the parent nodes is exponential in size, making it difficult to cover the space with even a large number of samples. This relates to the curse of dimensionality that notes the difficulty in sufficiently covering a space with many dimensions, and therefore extremely large volume.

When sampling in CTBNs, it is unlikely to encounter every state instantiation of parents, meaning that only a subset of the CIM is accessed. Given that only a subset of the CIM is in use during the inference process, storing the information that will never be used is wasteful and may limit the size of the models that can be represented. If the disjunctive interaction assumption holds, then a DCIM provides a much more compact representation that allows for the computation of portions of the CIM only as needed.

For instance, consider the case where a node has twelve parents, as shown by the right-most points in Figure 6.6. For these examples, the portion of the CIM that was actually used during inference runs was, at most, approximately 40%. Using a DCIM formulation, it is unnecessary to store all $3^{12} = 531441$ intensity matrices required for a CIM directly. Instead, $3 \times 12 = 36$ intensity matrices are stored for the DCIM, and the rates required during inference are computed dynamically.

We conclude the experiment section by informally noting the observed runtime performances. We acknowledge that CPU time is largely dependent on factors such as implementation and hardware; therefore, we provide this information only as a rough guide. Using a machine with modest hardware specifications, inference using the DCIM formulation for all experiments took seconds to complete for small numbers of parents, and up to several minutes for instances with 10, 11 and 12 parents. This was a massive improvement when compared to running the same experiments using a CIM formulation, and models with larger numbers of parents were unable to load due to memory restrictions. Improved hardware or optimized implementations may help alleviate this problem, but ultimately the exponential increase in model size will make representation and inference infeasible when using CIMs in scenarios with large numbers of parents.

6.6 Summary

In this chapter, disjunctive interaction was described in the context of CTBNs. The disjunctive conditional intensity matrix was introduced as a compact means of parameterizing nodes with disjunctive interaction. This DCIM parameterization was shown to reduce the number of required intensity matrices when compared to a standard CIM representation. Specifically, the number of intensity matrices required to specify the disjunctive model is linear in the number of parents rather than exponential. This chapter also provided a method to compute unspecified rates in a CIM using a DCIM defined over the same set of variables, allowing for dynamic computation of specific regions in a CIM as necessary. Disjunctive interaction was then described in more detail as it works with Boolean state variables, and the theory presented here was related back to the BN literature. Finally, a modified version of the importance sampling inference algorithm for CTBNs was presented which made use of the DCIM formulation and the dynamic rate computation process. This demonstrated the ability to use the DCIM formulation with existing algorithms provided in the literature on CTBNs.

The introduction of disjunctive interaction represents a major step toward improved scalability in CTBNs, especially in domains with causal relationships between variables. By providing a parameterization scheme that is linear in the number of parents rather than exponential, it is possible to represent models that would have otherwise been intractable. While disjunctive interaction works to achieve the same goal as the compact representations presented in the previous chapter, DCIMs are able to encode mathematical relationships between individual transition rates rather than describing similarities in the intensity matrices as a whole. This provides a compact model for describing CTBNs that can applied to different scenarios exhibiting causal relationships. Just as with the MCIM and TCIM representations, disjunctive interaction addresses the issue of scalability in CTBNs, and expands the set of processes that can be modeled using this framework.

CHAPTER SEVEN

APPLICATIONS

Up until now, this dissertation has presented techniques for improving representation or scalability in CTBNs. In either case, the underlying motivation behind our research is to support the adoption of CTBNs in real-world domains, either by extending the class of representable problems, or by managing complexity when working with large systems. In this chapter, we consider the specific domain of prognostics and health management (PHM), and present novel techniques for deriving the structure and parameters of a CTBN using a relatively small amount of diagnostic and reliability information that is already commonly available for most systems. This avoids the need to employ traditional CTBN learning algorithms, which are computationally more complex and typically require an abundance of data that may not be available for a system of interest. Furthermore, this chapter demonstrates how the CTDN framework presented in Chapter 4 can be used for performance based logistics (PBL), a task of increasing importance to large, contract-driven organizations. This chapter not only provides a guideline for applying CTBNs to PHM and PBL, but also serves as a demonstration of how the research in previous chapters can be applied in a real-world setting.

7.1 Background

This section starts by providing an overview of some additional notation. It then provides background information on D-matrices and fault trees, which are the

Table 7.1: Table of Notation

F_i	$\triangleq$	The i^{th} fault in the network
T_i	$\triangleq$	The i^{th} test in the network
$\mathbf{F}_i$	$\triangleq$	The vector of faults monitored by T_i (the parent set of T_i)
λ_i	$\triangleq$	Failure rate for fault F_i
μ_i	$\triangleq$	Repair rate for fault F_i
ND_i	$\triangleq$	Non-detect probability for test T_i
FA_i	$\triangleq$	False alarm probability for test T_i
$ND_{i,j}$	$\triangleq$	Non-detect probability for test-fault relationship T_i and $\mathbf{F}_i[j]$
$FA_{i,j}$	$\triangleq$	False alarm probability for test-fault relationship T_i and $\mathbf{F}_i[j]$

types of models that we leverage to construct prognostic CTBNs. Any related work relating to CTBN applications is covered in Chapter 2.

7.1.1 Notation

Before going into any technical detail, we first describe some additional notation in this chapter. Capital letters F_i and T_j are used to denote faults and tests respectively, where the subscript represents the index of the component in the system. A bold $\mathbf{F}_i$ is a vector of faults that are detected by a test T_i, therefore $\mathbf{F}_i[j]$ signifies the j^{th} fault detected by T_i. The subscripted Greek letters λ_i and μ_i are used to describe the failure and repair rates respectively for fault F_i. Similarly, ND_i and FA_i are the non-detect and false alarm probabilities respectively associated with test T_i, which describe the probabilities of the test failing to do its job. Symbols $ND_{i,j}$ and $FA_{i,j}$ also describe non-detect and false alarm probabilities, but do so for the specific test T_i and fault $\mathbf{F}_i[j]$ pair. Rather than describing the probability of complete test failure, $ND_{i,j}$ and $FA_{i,j}$ capture the notion that a test may fail to detect an individual fault. These symbols and their meanings are summarized in Table 7.1.

7.1.2 D-matrix

To perform system diagnosis and prognosis, we require certain information about the current state of a unit under test (UUT). This information is collected via tests performed on the UUT, and the outcome of these tests indicate the presence or absence of faults in the system [123]. In a complex system, this interplay of tests and faults can make diagnosis and maintenance difficult since a single fault may be monitored by multiple tests, and a single test may monitor multiple faults. To manage this complexity, the relationships between the faults and tests can be represented explicitly and compactly in an adjacency matrix that we call a D-matrix. Within this matrix, the columns correspond to tests and the rows correspond to the potential failures observed by the tests. The D-matrix, has been adopted by a number of modeling tools used to perform fault diagnosis, and the aerospace community is one of the major communities using these tools [119].

Constructing a D-matrix involves assembling the diagnostic signatures for every potential fault within the UUT. Given a set of faults $\mathbf{F}$ and a set of tests $\mathbf{T}$, we can define the function $eval(F_i, T_j)$ as follows:

$$eval(F_i, T_j) = \begin{cases} 1, & \text{if } T_j \text{ detects } F_i \\ 0, & \text{otherwise.} \end{cases}$$

For some fault F_i, when we apply this function to each $T_j \in \mathbf{T}$, the result is the diagnostic signature $\mathbf{F}_i = [eval(F_i, T_1), \ldots, eval(F_i, T_{|\mathbf{T}|})]$ for that fault [119, 129]. This diagnostic signature is constructed for each $F_i \in \mathbf{F}$. Each signature $\mathbf{F}_i$ forms the i^{th} row of the D-matrix.

The relationships between tests and faults, described by a D-matrix, provide information about how to diagnose the UUT, but they do not offer the complete picture. The logical relationships between the tests and faults are codified in the D-matrix, but probabilistic information is not captured by this format [129]. Similarly, the D-matrix does not provide information on fault-to-fault or test-to-test relationships; although, test-to-test relationships can be recovered through a process called "logical closure" [129]. For these reasons, while the D-matrix is used to assist in producing a network structure as detailed in Section 7.2.1 of this chapter, additional information is required to parameterize the model used to perform diagnosis and prognosis on the UUT. An example of a simple D-matrix is shown here, where test T_1 monitors both of the faults F_1 and F_2, while test T_2 monitors only F_1.

$$D_S = \begin{array}{c c} & \begin{array}{cc} F_1 & F_2 \end{array} \\ \begin{array}{c} T_1 \\ T_2 \end{array} & \begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix} \end{array}$$

7.1.3 Fault Trees

Fault tree analysis (FTA) is a powerful and well-established technique for evaluating system design in a reliability context. In a number of critical domains, fault trees encode knowledge about the system in a manner that is intuitive and easy to interpret. A fault tree provides a graphical representation that depicts the ways in which the failure of one or more system components can lead to system failure. A fault tree is a directed acyclic graph (DAG) consisting of a set of events and a set of logic gates. Interior nodes represent effects, while the leaf nodes are faults. Leaf states propagate upward through the tree to the root, modeling how faults can contribute to the various possible system failures. Each interior effect receives input from its children, which are either faults or other effects. Input to each effect is passed through a logic gate such as the AND gate or the OR gate.

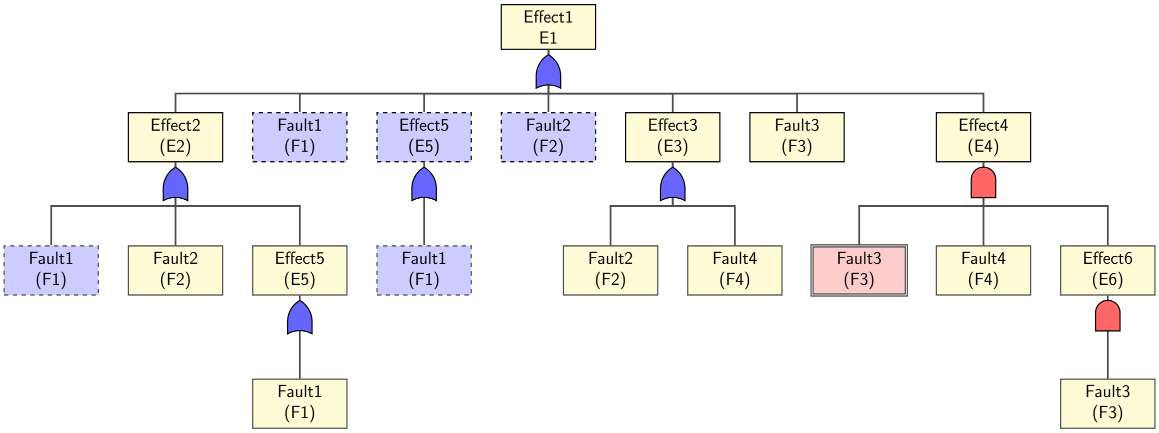


Figure 7.1: Example fault tree.

Some fault tree representations may allow for redundancies in the system that can cause the same component to appear in multiple parts of the tree thus violating the tree constraint. These redundancies can also be introduced when the system of interest is not represented easily as a polytree. The process of converting the system to a corresponding polytree representation often involves duplicating portions of the tree and placing them appropriately to avoid cycles. While these redundant components do provide valid information about the system, in many cases the information they provide is already encoded at another location in the tree. Furthermore, when these redundant components are removed from the fault tree with a valid pruning process, the resulting fault tree is easier to store, update and analyze. Fortunately, the pruned tree remains functionally equivalent to the original fault tree. Figure 7.1 shows an example of the fault tree models used in this chapter. We discuss the dashed and colored nodes in more detail in Section 7.3.1.

A related diagnostic model is the Fault Isolation Manual (FIM), which often plays a significant role in the maintenance of large systems [123]. FIMs are derived from decision trees and are typically referred to as a fault tree. Despite the name, these FIM-based fault trees are different from fault trees that we use in this chapter. The internal nodes of the FIM correspond to a series of tests, and the leaves correspond to a fault. To use the FIM, the user first performs the test specified by the root of the tree and follows the branch specified by the test outcome until reaching the leaf node corresponding to the diagnosed fault. In this way, an FIM enables the isolation of individual faults or ambiguity groups through the application of a structured sequence of tests, and can be used to derive BNs [129].

7.2 Deriving CTBNs from D-matrices

This section describes how a CTBN model can be derived from a D-matrix and related reliability and measurement data. Section 7.2.1 focuses on deriving the network structure using the dependence information encoded by the D-matrix. Section 7.2.2 describes how the nodes in the derived CTBN can be computed from failure rates, as well as false alarm and non-detect rates for tests.

7.2.1 Network Structure

The non-zero entries in a D-matrix represent a dependence relationship between a test and a fault. Similarly, the directed edges in a CTBN indicate dependence between variables. This observation can be used to derive the network structure for a CTBN from a D-matrix. Let D be an $m \times n$ D-matrix, and let G be a graph structure for a CTBN that initially contains no nodes. To start, m fault nodes are added to G, each corresponding to a row in matrix D. Then, n test nodes are added, corresponding to the columns in D. Note that fault and test nodes are behaviorally identical to any standard CTBN node and are described as faults and tests for semantic purposes only. Finally, for each entry in the D-matrix where $eval(F_i, T_j) = 1$, an arc is inserted from node F_i to node T_j. Each of these arcs indicate that the behavior of a test depends on the state of a fault at some point in time, which is equivalent to the information originally encoded by the D-matrix.

The resulting network G has $m+n$ nodes and m_d edges, where m_d is equal to the number of nonzero entries in the matrix. These directed edges are inserted exclusively from a fault node to a test node, with no edges from tests to faults, or between two test or fault nodes. As a result, the graph G is bipartite, with one layer consisting of the m faults and the other layer consisting of the n tests. Typically the variables contain relatively few states, meaning that the initial probability distributions and intensity matrices for each node are small. Instead, the total number of intensity matrices for each node is the primary factor driving complexity. Each fault node has no parent by construction, meaning that only a single intensity matrix is required to parameterize each of these nodes. The test nodes, however, may have any number of fault nodes in its parent set. As a result, a CIM for a test node will have $|\mathbf{Q}_{T_j}| = \prod_{F_i \in \mathbf{Pa}(T_j)} |F_i|$. In the worst case, a test may monitor all faults in the system, resulting in an upper bound on the space complexity equal to $O(c^m)$, where c is some small constant value that serves as a upper bound on the number of states for each fault.

7.2.2 Parameterization

The previous section described a procedure for producing a CTBN graph structure from a D-matrix, which consists of nodes and directed edges. While this network structure indicates the presence or absence of fault-test dependence, it does not quantify the strength of such a dependence. Furthermore, the matrix alone does not incorporate any knowledge about prior failure rates. The specifics of how one variable influences another, as well as default failure transition behavior, is captured by the parameterization of each node. This section shows how the constructed models can be parameterized directly by using reliability information. We assume Boolean states for both fault and test nodes, although the same principles apply to non-Boolean variables as well, so long as there exists reliability information to support the procedures.

7.2.2.1 Parameterizing Fault Nodes To begin, the fault nodes in the network can be parameterized, independent of the test nodes. Fault nodes have no parents in the network by construction, which is advantageous in terms of parameterization in that it associates each fault node with a single unconditional intensity matrix. The intensity matrix describes exponential transition distributions and provides expected transition probabilities between the states. In the Boolean case, this means that the CIM defines the probability distributions associated with transitioning to a failing state and transitioning back to a non-failing state. Typically these failure and repair rates are available in some form via either historical data or domain knowledge. Formally, a failure rate λ indicates the rate at which a failure will occur given that no failure currently exists. This failure rate may be available directly, or may be provided indirectly in the form of mean time between failures (MTBF). The mean time between failure is a metric more frequently used in the reliability literature, and is defined as $MTBF = 1/\lambda$.

A repair rate μ indicates the rate at which a fault will transition back to having no failure, and depending on expected repair policy is often set to zero. Note that if no repair policy is implemented, then MTBF is often referred to simply as the mean time to failure (MTTF), implying that the failure state is absorbing. Intensity matrices are nothing more than an organized collection of transition rates, so parameterization of each matrix can be achieved by inserting the rates into the correct locations. The failure rate λ is assigned to the entry that describes the transition from non-failure to failure, and the repair rate μ is assigned to the reverse direction. As required by the CIM definition, diagonal values are set to be the negative sum of the remaining entries in the row. Thus, for a fault F_i,

$$\mathbf{Q}_{F_i} = \begin{array}{c c} & \begin{array}{cc} f_i^0 & f_i^1 \end{array} \\ \begin{array}{c} f_i^0 \\ f_i^1 \end{array} & \begin{pmatrix} -\lambda_i & \lambda_i \\ \mu_i & -\mu_i \end{pmatrix} \end{array}. \tag{7.1}$$

7.2.2.2 Parameterizing Test Nodes With the fault nodes parameterized, the only remaining task is to parameterize the test nodes in the network. Specifically, a probability distribution for each test node must be determined given the faults that they monitor. Parameterizing test nodes is a more complex task than parameterizing the fault nodes, in that test nodes have an arbitrary number n of parents in the graph, meaning that the number of intensity matrices required to specify a test node's CIM will be exponential with respect to n. As a result, there are potentially many parameters in the model, and the need for a fast parameterization procedure is especially important in this context. Fortunately the relationships described by the D-matrices are inherently causal, making them suitable for the DCIM parameterization described in the previous chapter.

The intensity matrices for a test node's CIM may be derived using false alarm (FA) and non-detect (ND) likelihoods for each test. Just as with failure rates, FA and ND values are readily available for most systems of interest, obtained either from historical data or expert knowledge. The relation between FA/ND rates and the probability distribution for a test can be modeled using a line failure model, as described in detail by Perreault *et al.* [102]. The resulting probability distribution is given as:

$$
\begin{aligned}
P(T_i = 0|\mathbf{F}_i) = (ND_i)P(p_i = 1|\mathbf{F}_i) \qquad (7.2)\\
+ (1 - FA_i)P(p_i = 0|\mathbf{F}_i), \qquad (7.3)
\end{aligned}
$$

where T_i is a test in the network, $\mathbf{F}_i$ is the set of parent fault nodes for T_i, and ND_i and FA_i are non-detect and false alarm values defined for test T_i. These rates describe the likelihood that a test either fails to detect a fault despite fault input (non-detect event), or indicates the presence of a fault despite valid input (false alarm event). This may occur due to noise in the environment, or due to a failure in the test itself. The equation also incorporates two probabilities $P(p_i = 0|\mathbf{F}_i)$ and $P(p_i = 1|\mathbf{F}_i)$, as defined below.

$$
\begin{aligned}
P(p_i = 0|\mathbf{F}_i) &= \prod_{\{f_{i,j} \in \mathbf{F}_i | f_{i,j}=0\}} (1 - FA_{i,j}) \cdot \prod_{\{f_{i,j} \in \mathbf{F}_i | f_{i,j}=1\}} (ND_{i,j})\\
P(p_i = 1|\mathbf{F}_i) &= (1 - P(p_i = 0|\mathbf{F}_i))
\end{aligned}
$$

Here, $ND_{i,j}$ and $FA_{i,j}$ represent the non-detect and false alarm rates for test T_i, specifically with respect to the j^{th} fault in the parent set of T_i. These fault-dependent FA/ND rates may be caused by imperfect or noisy input data received from one of the components monitored by the test. The entire probability distribution $P(p_i|\mathbf{F}_i)$ can be thought of as the probability that the true state of faults $\mathbf{F}_i$ are received correctly or incorrectly by test node T_i.

While this representation is useful as a generalization, it is often the case that information is collected at a larger granularity, and specific false alarm and non-detect rates are not provided with respect to each fault. Given the previous equation, the number of non-detect and false alarm values necessary is equal to the number of edges in the CTBN network, plus the number of total tests: $O(|E| + |V|)$. Although this number is quite manageable from a computational standpoint, it may be that no data exists to describe the non-detect and false alarm values for specific fault-test relationships. Instead, the fault-test specific values $ND_{i,j}$ and $FA_{i,j}$ are all assumed to be zero, and any false alarm or non-detect likelihoods are summarized by the ND_i and FA_i values for test T_i. In this way, the false alarm and non-detect rates are assumed to be constant across all faults.

With this assumption, the number of user-specified parameters is reduced drastically. The only non-detect and false alarm values that need to be specified are for the test nodes: $O(|V|)$. Given this assumption, and assuming a test will fail given any failure of the monitored components, then the test's output will follow a deterministic OR gate. The intensity matrix for the case when no fault nodes are failing is shown in Equation 7.4, while the intensity matrix for the cases with failures is given by Equation 7.5. Using these equations, each intensity matrix in a CIM $\mathbf{Q}_{T_i}$ can be constructed, resulting in a procedure for parameterizing each test node using only the available false alarm and non-detect values.

$$Q_{\{T_i|(\wedge_{F\in\mathbf{F}_i}F=0)\}} = \begin{array}{c c} & \begin{array}{cc} t_i^0 & t_i^1 \end{array} \\ \begin{array}{c} t_i^0 \\ t_i^1 \end{array} & \begin{pmatrix} -(1-FA_i)^{-1} & (1-FA_i)^{-1} \\ (FA_i)^{-1} & -(FA_i)^{-1} \end{pmatrix} \end{array} \tag{7.4}$$

$$Q_{\{T_i|(\vee_{F\in\mathbf{F}_i}F=1)\}} = \begin{array}{c c} & \begin{array}{cc} t_i^0 & t_i^1 \end{array} \\ \begin{array}{c} t_i^0 \\ t_i^1 \end{array} & \begin{pmatrix} -(ND_i)^{-1} & (ND_i)^{-1} \\ (1-ND_i)^{-1} & -(1-ND_i)^{-1} \end{pmatrix} \end{array} \tag{7.5}$$

7.3 Deriving CTBNs from Fault Trees

Section 7.2 described how the network structure and parameters for a CTBN might be derived using a D-matrix and reliability information. The result is a model that describes potential faults and the tests that monitor these faults. Although this is useful from a diagnostic perspective, it fails to account for the effects that the faults might produce over time. While it is certainly beneficial to identify likely faults in a system, it is the risks that are of primary interest in most applications, with fault behavior serving as a means to an end. In this section, a method for deriving CTBNs from fault trees is provided, resulting in a network with faults and effects. An explanation of how this model relates to the D-matrix-produced model is then provided, and a method is given for merging the two networks.

7.3.1 Pruning Process

Recall that redundant information in the fault tree may result in an unnecessarily complex CTBN model. In this section, we present a pruning process that alleviates this problem through the elimination of redundant branches of the fault tree. To motivate this pruning process, we refer back to Figure 7.1. Note the effect E_2 on the second level of the tree. Due to the OR gate, this effect will have a value of 1 if and only if at least one of F_1, F_2 or E_5 are 1. In other words, we assume that E_2 is 0 unless one of the following three rules applies:

$$(\mathbf{E2.1}) \qquad F_1 = 1 \rightarrow E_2 = 1$$

$$(\mathbf{E2.2}) \qquad F_2 = 1 \rightarrow E_2 = 1$$

$$(\mathbf{E2.3}) \qquad E_5 = 1 \rightarrow E_2 = 1.$$

Now we focus on the subeffect E_5 in this input set. E_5 is determined by another OR gate with a single fault F_1, as an input. This means E_5 is determined by only a single rule:

$$(\mathbf{E5.1}) \qquad F_1 = 1 \rightarrow E_5 = 1.$$

We can derive implied rules by taking advantage of the transitive property. In this case, we can combine rules ($\mathbf{E5.1}$) and ($\mathbf{E2.3}$) to derive the rule $F_1 = 1 \rightarrow E_2 = 1$. This rule is implicitly followed by E_2, but note that the rule is also explicitly specified by rule ($\mathbf{E2.1}$). We say that effect E_5 already accounts for F_1, since $E_2 = 1$ if $F_1 = 1$, regardless if F_1 is a direct descendant or not. For this reason, it is redundant to list F_1 as a direct descendant of E_2. Although this representation may be advantageous in some situations where fault trees are used, the complexity of a CTBN model is driven by the number of dependencies in the model. For this reason, we wish to remove the extraneous dependencies that are already implicitly encoded by the fault tree.

As shown, fault F_1 under effect E_2 is unnecessary and can therefore be pruned from the fault tree while still retaining the semantic meaning encoded by the fault tree. In general, any direct descendant of an effect can be pruned if it is already accounted for by another child. In the case of an OR gate, inputs that are also OR gates will account for all of their children by producing implied rules via the transitive property. We refer back to Figure 7.1. Fault F_1 under E_1 can be pruned due to the chained rule $F_1 = 1 \rightarrow E_5 = 1 \rightarrow E_2 = 1 \rightarrow E_1 = 1$. This rule can be simplified to the more basic rule of $F_1 = 1 \rightarrow E_1 = 1$, which makes the direct descendant F_1 unnecessary for the top level effect E_1. We say that F_1 is already accounted for by E_2 via E_5. By this same logic, we can prune F_2 and E_5 from E_1 since they are already accounted for by E_3 and E_2 respectively. The nodes that can be pruned in Figure 7.1 using the rules implied by OR gates are shown with dashed lines.

A similar idea can also be applied to nested AND gates in a fault tree. Consider effect E_4 in Figure 7.1. The output of E_4 is determined by the rule $F_3 \wedge F_4 \wedge E_6 \rightarrow E_4$. Effect E_6 is in turn determined by the rule $F_3 \rightarrow E_6$. To obtain the implied rules for E_4, we can simply replace node E_6 with the logical expression that determines it. This results in the rule $F_3 \wedge F_4 \wedge (F_3) \rightarrow E_4$. Here we can see that F_3 is redundant in the logical expression. We say that effect E_6 accounts for F_3, and therefore E_4 does not require F_3 as a direct descendant. The pruned F_3 is highlighted in Figure 7.1 with a double outline. Note that in more complex cases where subeffects have more than one child, implied rules may involve lengthy chains of ANDed variables.

In general, components of a fault tree are accounted for by descendants so long as they chain through the same type of gate. For example, consider the component F_3 that we pruned under E_4. We cannot remove the instance of F_3 under E_1, since E_4 is determined by an AND gate, and E_1 is determined by an OR gate. This is because $F_3 = 1$ does not imply $E_1 = 1$ if the direct descendant is removed. If $F_3 = 1$, E_4 may still be 0 due to the requirement on F_4. The same logic can be applied in the reverse direction where an OR gate is nested below an AND gate. For this reason, the pruning process only applies when chaining through gates of the same type. We can think of the gates as partitioning segments of the tree that allow for implied rules. Algorithm 7.6 provides an outline of the process required to prune redundant fault tree components.

7.3.2 Network Structure

Recall that the structure of a CTBN takes the form of a directed graph $G = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V}$ is a set of nodes and $\mathbf{E}$ is a set of edges. The structure can be derived directly using a fault tree as follows. First the set of nodes $\mathbf{V}$ is obtained by extracting the variables directly from the faults and effects in the fault tree. Note that a fault or

Algorithm 7.6: Prune-Fault-Tree(node)

Input: The root node of the tree X.

```
$visited \leftarrow$ new list()
for each $Y$ in $GetChildrenOf$(X) do
  subnodes = PruneTree($Y$)
  if IsFault($Y$) or GatesEqual($X$, $Y$) then
    visited.Union(subnodes)
  end if
end for
for each $Y$ in $GetChildrenOf$(X) do
  if $Y \in visited$ then
    RemoveChild($X$, $Y$)
  else
    visited.Add($Y$)
  end if
end for
return visited
```

effect may occur in multiple locations of the fault tree, but the corresponding node in the CTBN will occur only once. To insure that no duplicates are added to the set **V**, a list can be maintained for variables that have already been added when iterating through the fault tree components.

Next, edges are inserted between the nodes to form the set **E**. These can be obtained directly from the structure of the fault tree. First note that faults only occur as leaves in the fault tree. This means that faults occur on their own accord and do not depend on any other modeled variables. Thus the corresponding nodes in the CTBN have no parents. Next we consider the effect nodes in the fault tree, whose states are determined by its inputs. We represent dependence on the inputs by adding an edge from the nodes corresponding to each input u_i to the node corresponding to the effect. The resulting CTBN has the same structure as the fault-tree after it has been pruned, but is generally shown reversed with the faults on top. The pruning process may result in substantially fewer edge insertions, and given that the space complexity of a node is exponential with respect to the number of parents, pruning is a critical preprocessing step.

7.3.3 Parameterization

The task of parameterizing a CTBN involves populating the initial distribution and the rates for the CIMs associated with each node in the network. In general, this set of parameters can be quite large. Let n_X be the number of states in the domain of variable X. Then the number of parameters required to populate all the associated IMs for node X is $(n_X(n_X - 1)) \cdot \prod_{A \in Pa(X)} n_A$. In system reliability, the prior probability that each component starts in a failing state is often known and is based on the failure rate. In this part of our model derivation we assume that all variables start in a working state with a probability of 1.0. For our purposes, we can assume the variables are Boolean, consisting of a failing state and a non-failing state. In this case, each node X requires $2^{(|Pa(X)|+1)}$ rates, which can make identifying parameters for even relatively small models a difficult task. Just as in Section 7.2, the goal is to reduce the number of rates required to parameterize a node by taking advantage of common reliability information and by exploiting behavioral information obtained from the underlying fault tree. The fault nodes in the network are identical to those obtained from the D-matrix; therefore, the same parameterization process from Section 7.2.2.1 can be employed. Parameterization of the remaining effect nodes are described throughout the remainder of this section, broken out by the logical gate that determines the effect's behavior. Specifically, Section 7.3.3.1 describes effects that use a logical AND gate, while Section 7.3.3.2 provides a parameterization scheme for effects following an OR gate.

7.3.3.1 Parameterization of AND Effects Although we wish to model state transitions that occur over time, eventually it is expected that the behavior of an effect node in a CTBN matches the behavior of the corresponding static logic gate. Let F_X be the discrete function corresponding to the gate from which variable X was derived. Then it is expected that $\lim_{t\to\infty} P(X(t) = F_X(Pa(X))) = 1.0$. This behavior is guaranteed for each CIM $Q_{X|Pa(X)}$ by forcing a non-zero transition rate *to* the state produced by $F_X(Pa(X))$, and a zero rate *from* the state produced by $F_X(Pa(X))$. This results in an absorbing state that will eventually be reached, at which point the variable will remain in this state until a change in at least one of the parents occurs.

In the case of a variable created from an AND gate, F_X becomes the logical AND function. Here F_X is expected to produce a value of 1 if and only if all inputs are 1. To ensure the desired behavior in the CTBN, the CIM is parameterized for a node X where all parents are 1 as follows:

$$Q_{X|Pa(X)} = \begin{array}{c c} & \begin{array}{cc} x^0 & x^1 \end{array} \\ \begin{array}{c} x^0 \\ x^1 \end{array} & \begin{pmatrix} -\lambda_X & \lambda_X \\ 0 & 0 \end{pmatrix} \end{array},$$

when $F_X(Pa(X)) = 1$ (all ones).

Next, we turn our attention toward the remaining cases where not all parents of the node are 1. In this case, the function F_X produces a value of 0, so we wish to describe the time it takes to transition back to a state of 0 for the node in the CTBN.

To achieve this, the CIM is parameterized as:

$$Q_{X|Pa(X)} = \begin{array}{c c} & \begin{array}{cc} x^0 & x^1 \end{array} \\ \begin{array}{c} x^0 \\ x^1 \end{array} & \begin{pmatrix} 0 & 0 \\ \mu_{X|Pa(X)} & -\mu_{X|Pa(X)} \end{pmatrix} \end{array},$$

when $F_X(Pa(X)) = 0$ (not all ones).

Note that it is required to specify a rate for transitioning back to 0 for each of the instances where not all parents are in state 1. Although this specificity may be necessary in some cases, it may be possible to simplify the parameterization process by assuming the rate is the same, regardless of the parent set. This means that X transitions back to 0 at the same rate, so long as F_X evaluates to 0. Given this, a variable X derived from an AND node in a fault tree can be specified using only two parameters. λ_X defines the time it takes to transition to 1 in the event that all parents are in state 1, and μ_X is used in all other cases to indicate when X will transition back to 0.

7.3.3.2 Parameterization of OR Effects When a variable X is created from an OR gate, F_X is defined as the logical OR function. In this case, F_X is expected to produce a value of 0 if and only if all the parents of node X are 0. Again, it is guaranteed that this CTBN node eventually reaches state 0 by parameterizing the intensity matrix such that state 0 is an absorbing state:

$$Q_{X|Pa(X)} = \begin{array}{c c} & \begin{array}{cc} x^0 & x^1 \end{array} \\ \begin{array}{c} x^0 \\ x^1 \end{array} & \begin{pmatrix} 0 & 0 \\ \mu_X & -\mu_X \end{pmatrix} \end{array},$$

when $F_X(Pa(X)) = 0$ (all zeroes).

Next, consider the cases where F_X evaluates to 1. Since F_X is a logical OR gate, this occurs whenever at least a single parent is in state 1. In this case, node X is parameterized such that the intensity matrices in the CIM eventually transition to state 1:

$$Q_{X|Pa(X)} = \begin{array}{c c} & \begin{array}{cc} x^0 & x^1 \end{array} \\ \begin{array}{c} x^0 \\ x^1 \end{array} & \begin{pmatrix} -\lambda_{X|Pa(X)} & \lambda_{X|Pa(X)} \\ 0 & 0 \end{pmatrix} \end{array},$$

when $F_X(Pa(X)) = 1$ (not all zeroes).

Here again it is required to specify a rate parameter for every possible state instantiation of the parents where at least one parent is in state 1. This means that $2^{|Pa(X)|}$ parameters are required for node X. As in the case of the AND node in the previous section, this number can be reduced by making a simplifying assumption. Given the causal interpretation of the model, the concept of disjunctive interaction from Chapter 6 may be employed. This allows for specification of the rates λ_{X_i} for only the cases where a single parent is in state 1. This reduces the number of required parameters to be linear in the number of parents rather than exponential. The remainder of the parameters for the CIMs where multiple parents are in state 1 are accounted for by disjunctive interaction.

7.3.4 Merging Derived Models

Sections 7.3.2 and 7.3.3 provide a method for deriving the structure and parameters for a CTBN automatically using a fault tree and a relatively small number of rate parameters. The resulting network consists of a set of fault nodes, as well as a set of effect nodes that depend on these faults. The constructed CTBN supports queries about the expected behavior of effects over time, given information that might be know about the faults in the system. This section briefly describes how to merge this model with a CTBN produced from D-matrices as described earlier in Section 7.2. The key observation that makes this merge process possible is that given a D-matrix D_S and a fault tree T_S that both describe a single system S, the set of faults is the same for both diagnostic models. A CTBN that models system S as a whole can be obtained by merging the two CTBNs derived from D_S and T_S.

Let $CTBN_D$ be a CTBN constructed from a D-matrix D_S, and let $CTBN_F$ be a CTBN derived from a fault tree F_S. Furthermore, let $\mathbf{T}$ be the set of test nodes in $CTBN_D$, $\mathbf{E}$ be the set of effect nodes in $CTBN_F$, and $\mathbf{F}$ the set of faults contained in both CTBNs. A new model $CTBN_S$ can be obtained for system S by combining both networks to obtain a single CTBN with nodes $\mathbf{T}$, $\mathbf{E}$, and $\mathbf{F}$. The nodes in the sets $\mathbf{T}$ and $\mathbf{E}$ ultimately have the same parents and parameters as they did in the original networks. The only difference is that the nodes in the set $\mathbf{F}$ in the combined CTBN have more children than they did prior to being merged, but this does not affect their behavior or parameterization. Furthermore, fault nodes in $CTBN_D$ and $CTBN_F$ are parameterized using the same method, meaning that there is no conflict when consolidating the faults from the individual CTBNs.

7.4 Vehicle System Demonstration

This section works through an example intended to demonstrate how the material presented in this chapter is used in a practical application. In this example, a model is constructed from reliability data describing the uptime for a military vehicle, which going forward is formally denoted as system S. Reliability and cost information is based on real-world data for the High Mobility Multipurpose Wheeled Vehicle (HMMWV) and is an extension of the vehicle model presented in Chapter 4.

Table 7.2: Summary of Faults

Faults	Full Name	MTBF	MTTR	Repair Costs (USD)
AI	Air System	900	5	250
AL	Alternator	1300	15	900
AX	Axles	1800	12	2000
BR	Brakes	1200	5	950
CO	Cooling System	2100	6	200
EL	Electronics	1700	20	600
FU	Fuel System	800	16	1200
IG	Ignition System	600	13	300
PR	Compression System	1100	35	9000
PW1	Power Source 1	400	3	250
PW2	Power Source 2	200	3	250
SU	Suspension System	1500	8	850
TR	Transmission	2500	30	6500
WT	Wheels/Tires	700	3	700

The intent is that by using this real-world example, the applicability of CTBNs to PHM and other domains will become more clear.

The major components for the extended vehicle model are enumerated in Table 7.2. The identifier is provided by the first column, while the full descriptive name is shown in the second column. The third and fourth columns provides the mean time between failure and mean time to repair in hours. The last column shows the average repair cost in US dollars. Notice that the air system (AI), alternator (AL), fuel system (FU), ignition system (IG), and compression system (PR), and power sources one and two (PW1/PW2) are all new components that were not modeled in the original vehicle model from Chapter 4. In total, there are now 14 components, representing the major subsystems in the vehicle.

The tests that have been assigned to monitor the faults from Table 7.2 are shown in Table 7.3. The first two columns show the identifier and full name for the test. The third column gives a brief description providing more information about what the test

Table 7.3: Summary of Tests

ID	Full Name	Description	FA	ND
AF	Air Filter Check	Checks the state of the air filter.	0.07	0.1
AM	Axle Movement	Check for excessive axle movement in a suspended vehicle.	0.04	0.03
BC	BCM Test	Checks health of the body control module.	0.02	0.02
BT	Bounce Test	Check number of bounces to recover from suspension depression.	0.05	0.06
BT1	Battery Test 1	Tests the battery's ability to hold a charge.	0.02	0.01
BT2	Battery Test 2	Tests the battery's ability to hold a charge.	0.02	0.01
CP	Coolant Pressure Test	Tests pressure of the cooling system.	0.02	0.01
DE	Decelerometer	Simple test of deceleration.	0.09	0.05
DY	Dynamometer	Measures for torque and power.	0.04	0.03
FC	Fluid Check	Check for proper fluid levels, color, etc.	0.04	0.02
FP	Fuel Pump Test	Tests the fuel pump.	0.02	0.02
FT	Fuse Tests	Tests if fuses are operational.	0.01	0.02
GC	Groove Cracking	Check for cracks in tire grooves.	0.01	0.01
GT	Gauge Test	Tests gauges.	0.02	0.05
LT	Leakdown Test	Tests cylinder pressures.	0.02	0.01
PWT	Power Test	Tests for the presence of vehicle power.	0.01	0.01
RS	RPM Shift Test	Tests for proper gear shift behavior based on RPM thresholds.	0.04	0.05
RT	Roller Test	Comprehensive break test (static vehicles).	0.01	0.01
SP	Spark Plug Test	Tests each spark plug for ignition.	0.02	0.05
TT	Timing Test	Tests distributor, timing belt, etc.	0.01	0.01
TW	Tread Wear	Check tire tread depth at various points.	0.03	0.02
VM	Voltage Meter	Performs a voltage test on the alternator.	0.03	0.08

does. Finally, the last two columns show the false alarm and non-detect rates for the tests. These rates represent the overall expected false alarm and non-detect rates, and it is assumed that these rates are approximately the same for each fault monitored by the tests. Note that some tests are inherently more effective than others. For instance, although both tests check the brakes component, the Decelerometer (DE) test has relatively high false alarm and non-detect rates, while the Roller Test (RT) is highly effective and hardly ever produces erroneous results. This means that RT is more effective than DE with respect to failure probabilities, although it is likely a more costly in terms of time and money, which may be an important factor in some scenarios. There are a total of 22 tests, each of which monitor a subset of the components listed in Table 7.2.

7.4.1 Structure Derivation

The specifics of which components are monitored by which tests is encoded in the D-matrix D_S shown in Figure 7.2. The rows are labeled with the test identifiers, and the columns are labeled with the component identifiers. As dictated by the number of tests and components, the dimensions of the matrix are 22×14, for a total of 308 entries. As is often the case, the D-matrix is sparse, with only 29 of the entries being non-zero. In other words, there are 29 fault-to-test relationships out of a possible 308 in the event that all tests monitor all components. The dependence information in D_S, in conjunction with the reliability information in Tables 7.2 and 7.3, are sufficient to derive a bipartite CTBN network as described in Section 7.2. The resulting model $CTBN_D$ contains 22 test nodes, 14 fault nodes, and 29 edges between them.

Next, consider the fault tree F_S shown in Figure 7.3, which encodes the relationships between the vehicle faults and possible events. The top level effect is the `LossOfCrew` event, which indicates the event where one or more crew members of

$$D_S = \begin{array}{c|cccccccccccccc|}
 & AI & AL & AX & BR & CO & EL & FU & IG & PR & PW1 & PW2 & SU & TR & WT \\
AF & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
AM & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
BC & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
BT & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\
BT1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\
BT2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\
CP & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
DE & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
DY & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
FC & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\
FP & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
FT & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
GC & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\
GT & 0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
LT & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\
PWT & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\
RS & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\
RT & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
SP & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
TT & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\
TW & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 1 \\
VM & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{array}$$

Figure 7.2: D-matrix for the vehicle model. The rows represent tests while the columns represent potential faults. Nonzero entries in the matrix indicate that the test associated with the row monitors the fault associated with the column.

the vehicle perish. This effect may occur if either the LossOfChassis event occurs, or if the LossOfVehicle event occurs. The LossOfChassis event describes a situation where the chassis of a vehicle is no longer operational, due to either a failure in the brakes, wheels/tires, axle, or suspension subsystems. This chassis failure may cause a loss of crew if the failure is catastrophic and the damage to the vehicle is passed along to the crew. The LossOfVehicle event occurs if any of the three main subsystems fail, as captured by the LossOfElectrical, LossOfChassis, and LossOfPowerTrain events. In a military setting, an inoperable vehicle may result in a loss of crew due to the crew's dependence on vehicle functionality for mission operations. The LossOfPowerTrain event depends on the transmission and coolant system components, as well as the LossOfEngine event, which is broken down by the compression, ignition, air, and fuel components. The LossOfElectrical event depends on the alternator and electrical components, along with the LossOfPower event, which can occur if there is a failure in both of the two power sources. The redundancy in the design of the power source subsystem is the reason behind the AND gate for the LossOfPower event rather than the OR gate that is followed by other events in the fault tree. Note that each of the leaf nodes are faults from Table 7.2, while the internal nodes represent the effects or risks in the vehicle system.

Before constructing a CTBN from the fault tree, the pruning process is applied to eliminate redundant information. For the fault tree in Figure 7.3, there is only one branch that is removed, denoted with blue dashed nodes. The reasoning is that the LossOfChassis subsystem that feeds into the LossOfCrew event is already accounted for as input to the LossOfVehicle event. More formally, the branch may be pruned because LossOfChassis occurs at a lower level in the tree, and each of the events of interest follow the OR gate function. The same concept could be applied to AND gates if there was repetition in the LossOfPower branch. Although only one branch

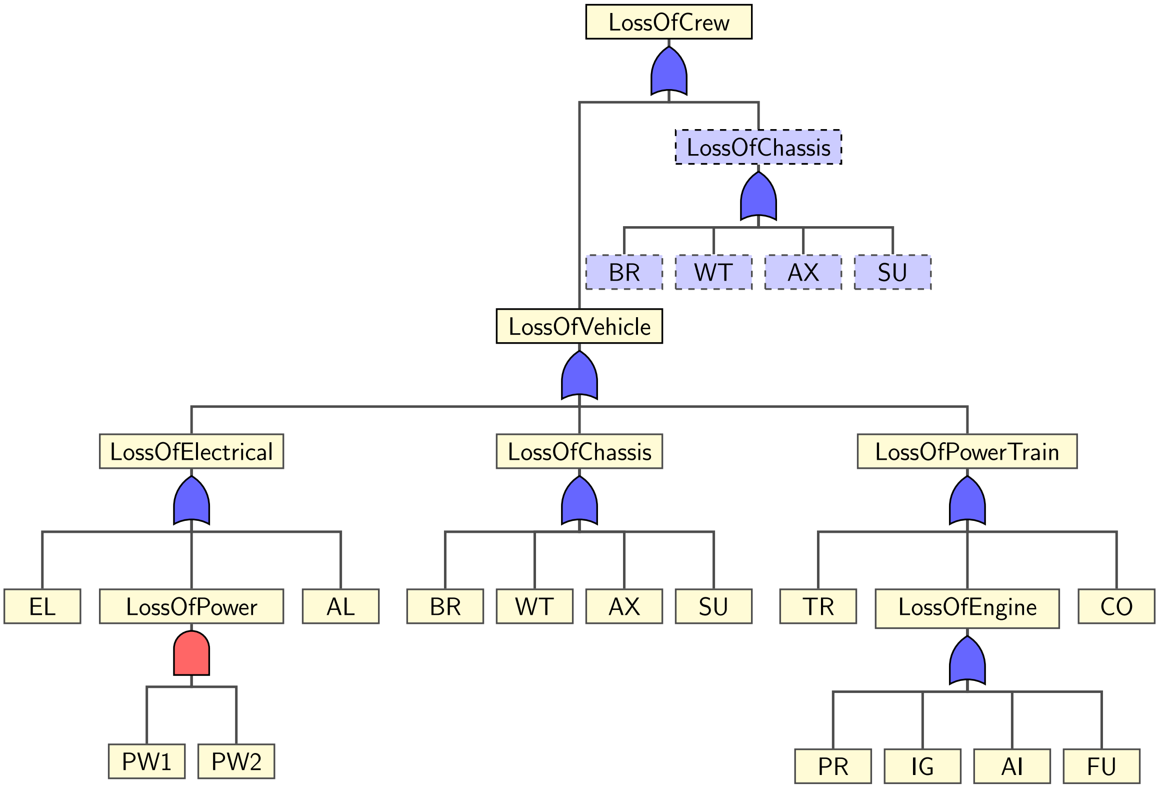


Figure 7.3: Fault tree for the vehicle model.

is pruned for the given structure, removing the `LossOfChassis` event also removes the brakes, wheels/tires, axle, and suspension subsystems below the event, resulting in a savings of five nodes in total.

Using the fault tree, the network structure can be obtained for a CTBN using the procedure described in Section 7.3. Starting at the top, `LossOfCrew` is added to the network as an effect node. The effect `LossOfVehicle` is then added to the network, and an edge is inserted from the new `LossOfVehicle` node to the `LossOfCrew` node. Note that `LossOfChassis` is not yet inserted since that branch was pruned during the preprocessing step. Next, the `LossOfElectrical`, `LossOfChassis`, and `LossOfPowerTrain` effects are added to the network, and edges are inserted so that these three new nodes form the parent set for `LossOfVehicle`. This process continues until the leaf fault nodes are reached, which are inserted into the network with empty parent sets. The produced network $CTBN_F$ contains 7 effect nodes, 14 fault nodes, and 20 edges.

Since both networks $CTBN_D$ and $CTBN_F$ were derived from the same vehicle system S, they share a common set of fault nodes and can be merged using the process described in Section 7.3.4. Figure 7.4 shows the final network structure for the merged CTBN obtained from the vehicle system D-matrix and fault tree. As expected, this network is comprised of 7 effect nodes, 14 fault nodes, and 22 test nodes, matching the node counts from the individual networks $CTBN_D$ and $CTBN_F$. Here, effect nodes are represented as ovals, fault nodes are circular, and test nodes are octagonal. Furthermore, the network has a total of 49 edges, which is the sum of the edge counts for each of the original networks. Note that all test nodes have at least one fault node as a parent, indicating that the behavior of the test depends on one or more faults. Similarly, the effect nodes have at least one parent, which is either a fault or

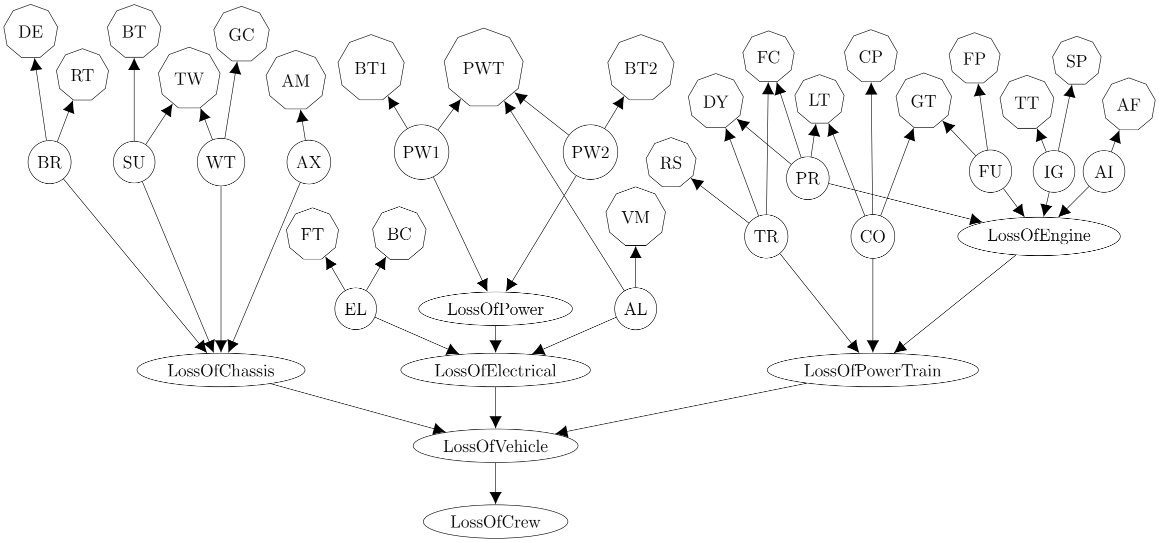


Figure 7.4: Vehicle model.

another effect. None of the fault nodes have any parents in the graph, which is by construction based on the D-matrix and fault tree information.

To better understand the network construction process, consider the power subsystem in the center of the network in Figure 7.4. The `LossOfPower` effect has two parents: `PW1` and `PW2`. This comes from the two leaf nodes that act as inputs to the `LossOfPower` node in the fault tree from Figure 7.3. Note that the type of gate does not affect the structure derivation process, only the parameterization. `PW1` has two parents, `BT1` and `PWT`, which correspond to the two non-zero entries in the `PW1` column of the D-matrix D_S. Alternatively, the faults that a test monitors can be looked up by using the rows in the D-matrix. For instance, it can be seen that test `PWT` monitors faults `AL`, `PW1` and `PW2`, corresponding to the three children for `PWT` in Figure 7.4. Each node or edge in the network can be traced back to information obtained from either the fault tree or the D-matrix for the vehicle system.

7.4.2 Parameterization

With the network structure for the vehicle model in place, the only remaining task is to parameterize the nodes in the CTBN. This involves specifying rates for each entry in each of the intensity matrices in the CIM for each node. While this is a complex task when employing traditional learning algorithms, the process is made relatively simple by taking advantage of reliability information. To demonstrate how parameterization is achieved, this section shows the parameters for example nodes in the network corresponding to each of the different node types.

First, consider the fault node PR, representing engine compression. Table 7.2 shows that the mean time between failure for the PR fault is 1100, while the mean time to repair is 35. As such, the failure rate is $1/1100 \approx 0.001$, and the repair rate is $1/35 \approx 0.029$. Assuming state pr^0 is non-failing, and pr^1 is failing, then the intensity matrix for node PR is shown below. Since PR is a fault node, it has no parents by construction, and its CIM contains only this single matrix.

$$\mathbf{Q}_{PR} = \begin{array}{c} \\ pr^0 \\ pr^1 \end{array} \begin{array}{c} \begin{array}{cc} pr^0 & pr^1 \end{array} \\ \begin{pmatrix} -0.001 & 0.001 \\ 0.029 & -0.029 \end{pmatrix} \end{array}$$

Next, consider the LT test node, representing the leakdown test. According to Table 7.3, the false alarm rate for test LT is 0.02, and the non-detect rate is 0.01. Equation 7.4 shows how the false alarm rate can be used to derive the intensity matrix $\mathbf{Q}_{\{LT|pr^0,co^0\}}$. Specifically, the transition rate from state lt^1 to lt^0 is $FA^{-1} = 50$, and the transition rate from lt^0 to lt^1 is $(1 - FA)^{-1} = 1.02$. Similarly, Equation 7.5 shows how the non-detect rate is used to produce intensity matrices $\mathbf{Q}_{\{LT|pr^0,co^1\}}$ and $\mathbf{Q}_{\{LT|pr^1,co^0\}}$. Both of these matrices are identical due to the assumption that the false alarm and non-detect rates for the LT test are equivalent regardless of the fault being monitored. The derived intensity matrices are shown below, which together fully specify the CIM for test node LT.

$$\mathbf{Q}_{LT} = \left\{ \begin{array}{ll} \mathbf{Q}_{\{LT|pr^0,co^1\}} = \begin{array}{cc} & \begin{array}{cc} lt^0 & lt^1 \end{array} \\ \begin{array}{c} lt^0 \\ lt^1 \end{array} & \begin{pmatrix} -100 & 100 \\ 1.01 & -1.01 \end{pmatrix} \end{array} & \mathbf{Q}_{\{LT|pr^1,co^0\}} = \begin{array}{cc} & \begin{array}{cc} lt^0 & lt^1 \end{array} \\ \begin{array}{c} lt^0 \\ lt^1 \end{array} & \begin{pmatrix} -100 & 100 \\ 1.01 & -1.01 \end{pmatrix} \end{array} \\ \mathbf{Q}_{\{LT|pr^0,co^0\}} = \begin{array}{cc} & \begin{array}{cc} lt^0 & lt^1 \end{array} \\ \begin{array}{c} lt^0 \\ lt^1 \end{array} & \begin{pmatrix} -1.02 & 1.02 \\ 50 & -50 \end{pmatrix} \end{array} & \end{array} \right\}$$

These three intensity matrices cover the case where all of the monitored faults are in a non-failing state, as well as each of the cases where a single parent is in a failing state. The remaining parent state instantiation with multiple failures is implicitly covered by the disjunctive interaction parameterization, as introduced in Chapter 6. By making use of the disjunctive interaction, the last remaining intensity matrix $\mathbf{Q}_{\{LT|pr^1,co^1\}}$ is generated on the fly using the existing parameters.

Finally, consider the `LossOfElectrical` effect node. This effect is dictated by an OR gate of inputs, meaning that just as with the test node parameterization, disjunctive interaction can be employed to reduce the number of intensity matrices required. Assume that the time it takes for `LossOfElectrical` to take effect is exponentially distributed with a rate of 5 when the `EL` fault has occurred, with a rate of 35 when `LossOfPower` occurs, and a rate of 1.5 when the `AL` fault occurs. Furthermore, it is assumed that if there is no input to cause the effect, then `LossOfElectrical` is guaranteed not to occur. Then the intensity matrices for the case where no faults have occurred and where one of each of the three faults have occurred is listed below. For conciseness, `LOE` is used as shorthand to represent the `LossOfElectrical`, and `LOP` stands for `LossOfPower`. Just as before, the remaining unspecified intensity matrices for the CIM can be derived automatically using disjunctive interaction.

$$\mathbf{Q}_{LOE} = \left\{ \begin{array}{cc} \mathbf{Q}_{\{LOE|el^0al^0lop^0\}} = \begin{array}{c|cc} & loe^0 & loe^1 \\ \hline loe^0 & 0 & 0 \\ loe^1 & \infty & -\infty \end{array} & \mathbf{Q}_{\{LOE|el^1al^0lop^0\}} = \begin{array}{c|cc} & loe^0 & loe^1 \\ \hline loe^0 & -5 & 5 \\ loe^1 & 0 & 0 \end{array} \\ \mathbf{Q}_{\{LOE|el^0al^1lop^0\}} = \begin{array}{c|cc} & loe^0 & loe^1 \\ \hline loe^0 & -1.5 & 1.5 \\ loe^1 & 0 & 0 \end{array} & \mathbf{Q}_{\{LOE|el^0al^0lop^1\}} = \begin{array}{c|cc} & loe^0 & loe^1 \\ \hline loe^0 & -35 & 35 \\ loe^1 & 0 & 0 \end{array} \end{array} \right\}$$

Although this section covered parameterization of only three of the 43 nodes in the vehicle network, the same principles can be applied to the remaining 40 nodes. Methods exist for parameterizing faults, tests, and effects. In each case, methods make use of available reliability data, and when nodes behave according to an OR input, disjunctive interaction is employed to reduce the required number of parameters.

7.5 Usage and Decision Making

In the previous section, a prognostic model was created to describe a military vehicle using automated derivation procedures and reliability data. The derived CTBN contains nodes representing tests, faults, and effects related to one another with edges and parameterized using transition rates derived from reliability data. This model can use state information collected about tests to perform diagnostics and prognostics. More specifically, evidence is applied to the test nodes indicating whether or not the test passed or failed, and at what time. Queries can then be made to obtain probability distributions over the states of faults and effects at any point in time. These probability distributions can identify the most likely effects and how probable they are.

Traditionally, approaches to PHM use domain experts to synthesize the probability information in order to gauge the implication of effect likelihoods and determine the next course of action. The concern is that the analysis and decision making can be informal and difficult to justify. Instead, a more ideal approach is to build the concept of utility and decision making into the model directly, and rely on standard query techniques to quantify the quality of the system. To accomplish this, we use the CTDN framework proposed in Chapter 4, which can be used to model potential actions and their associated outcomes. This allows for an estimated evaluation of an action before it is enacted in the real world, providing a more mathematically rigorous method for decision making in the presence of uncertainty. The remainder of this section describes how the CTDN framework can be used to decide between different operational or design choices.

7.5.1 Scenarios

In some cases, it may be useful to construct multiple variations of the same model. We refer to these model variations as "scenarios.", which represent maintenance actions or different system modes of operation. For instance, one option may be to perform preventative maintenance on the vehicle at time $t = 500$ hours and replace the wheels/tires prior to a failure event for the component. This results in two possible scenarios, one in which no preventative action is taken, and one where the `WT` component is replaced. In the replacement scenario, there is a guaranteed downtime for the system in order to perform the maintenance operation, as well as a cost associated with that operation. With that said, the preventative action will reduce the likelihood that the component fails on its own, may prevent a more drastic `LossOfVehicle` event, and may even prevent a `LossOfCrew` event in an extreme case. Another example of a scenario may be to change the operational mode of the vehicle.

If the cooling system has a high probability of failing in the near future, rather than continue operating the vehicle as normal, a more conservative approach could be taken that reduces speed and conserves power consumption. This would reduce the likelihood of failure for the component but may have other associated costs.

There are several options when modeling scenarios. A naïve approach is to duplicate the base model and make the necessary changes to the network structure or parameters to account for the difference in the new scenario. The result is two networks that vary in some shape or form. Queries can then be performed over the two networks simultaneously, and the results may be compared. Although this works from a technical standpoint, scenarios often differ by very little, meaning that the various networks will have many redundancies. This may be a concern when there are many scenarios or when the models are very large. Furthermore, in the event that a change must be made to the model, this change must be propagated through all of the copied networks, causing a consistency concern.

An alternative approach is to use the decision nodes in a CTDN. In this case, the states of the decision node represent scenarios that may be "selected." After inserting the decision node, any nodes that are changed as part of the scenario are added to the child set of that node. The child node's parameters are then modified to account for the new scenario.

See Figure 7.5 for an example of how a decision node might be added to the vehicle network from Figure 7.4. Here, only the cooling subsystem is shown, as the remainder of the vehicle network remains unaltered. The alteration to the network is shown in dashed lines. The only change is that the cooling node `CO` now has an additional parent `Operation`, depicted as a rectangular decision node and representing the operational modes of the vehicle. For the purposes of this example, there are two operational modes as discussed earlier: standard or conservative. This

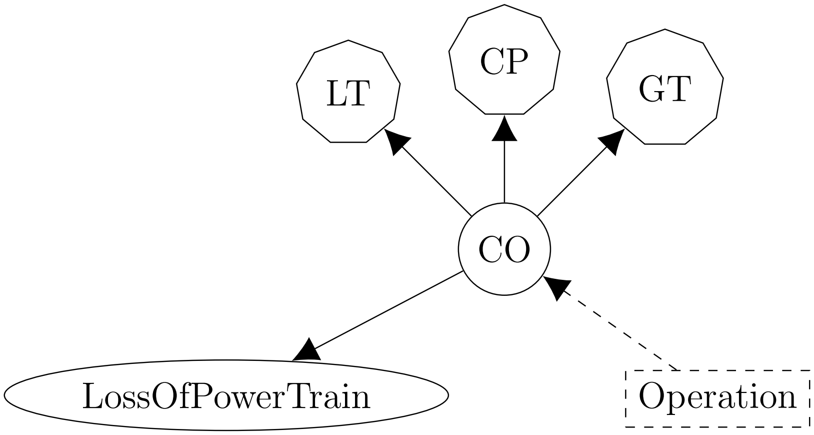


Figure 7.5: The cooling subsystem with the inclusion of a decision node that encodes different system operational decisions.

violates the constraint imposed previously that each fault has no parents. Essentially this means that the components of the vehicle no longer behave independently of external factors but instead are influenced by decisions.

The model must now be reparameterized to account for this change to the structure of the network. The fault `CO` originally had a CIM with only a single intensity matrix $\mathbf{Q}_{CO}$, which assumed standard operation of the vehicle. The new CIM now contains two matrices, one for each state of its new parent: $\mathbf{Q}'_{CO|Operation} = \{\mathbf{Q}'_{CO|standard}, \mathbf{Q}'_{CO|conservative}\}$. Since the original model assumed standard operation, the transition behavior in that case is already known and $\mathbf{Q}'_{CO|standard} = \mathbf{Q}_{CO}$. The remaining intensity matrix $\mathbf{Q}'_{CO|conservative}$ is then defined to describe the new cooling system behavior given a conservative operational mode. For this scenario, the failure rate will likely decrease with the conservative driving style, while the repair rate will remain the same. As discussed in the Chapter 4, parameterization of the decision node itself is unimportant so long as all states are reachable at all times.

Now that operational decisions have been modeled, utility must be defined for the CTDN using performance functions. Let F be a performance function that assigns value for every hour spent in the operational or partially operational state, and no value when in a non-operational state.

$$F = \begin{cases} 10t, & \text{if } LOV = false \wedge Operation = standard \\ 5t, & \text{if } LOV = false \wedge Operation = conservative \\ 0, & \text{otherwise} \end{cases}$$

Here `LOV` is shorthand for `LossOfVehicle`. To model this function directly using the model itself, a new utility node can be added to the network as a child of the `LossOfVehicle` and `Operation` nodes. This utility node has three states that are entered deterministically based on the three conditions in F, each of which are assigned their corresponding value from F.

The vehicle model has now been extended to use a CTDN representation that describes the possible scenarios and utility values. To use this model, inference can be run over the network for each possible decision, and the results can be compared. In this case, that means fixing the `Operation` node to a standard state and running inference and then repeating the process after setting the `Operation` node to a conservative state. This results in two output values for the utility function. The task of making a decision then reduces to maximizing the expected utility over the possible scenarios. Let $\mathbf{A} = \{\{Operation = standard\}, \{Operation = conservative\}\}$ be the set of possible actions or scenarios. Then the optimization problem is summarized by the following maximization:

$$\operatorname*{argmax}_{A \in \mathbf{A}} F(A).$$

Note that computing $F(A)$ requires that evidence be set in the CTDN according to the state assignments in A, and inference is run to obtain the expected value for performance function F when the network is in this state.

In this vehicle model, running the vehicle in a conservative state is less valuable than standard operation, but the decrease in the failure rate for the cooling system may be substantial enough to prevent a total loss of vehicle event. It might therefore make more sense to take the partial value that is available rather than risk losing the vehicle entirely and receiving no value at all. Additional, more complex performance functions may also be defined as necessary, which may change the outlook of the decision process. In the event that there are multiple performance functions, multi-objective optimization can be employed to obtain a Pareto frontier of non-dominated scenarios. This subset of superior scenarios can then be presented to a domain expert who can choose the scenario that best meets the needs of the application. By building these functions into the model directly using the CTDN framework, it is possible not only to support diagnostics and prognostics over faults and risks, but also to compare scenarios using metrics that are relevant to the system's application.

7.5.2 Performance Based Logistics

In addition to supporting decisions for runtime operations, CTDN models can be used in the design stage as well. This is especially useful when working under the guidelines of performance based logistics (PBL), which is a contracting strategy that aims to improve operational effectiveness in large organizations like the DoD. Unlike other approaches that contract for resources, PBL contracts for performance according to a variety of prespecified metrics [124]. These metrics are related directly to system performance, such as reliability, maintainability, and supportability [65]. In short, PBL contracts purchase an open-ended solution from a contractor that must meet performance criteria, which requires an objective method for evaluating such performance values.

By building a prognostic CTBN during the development cycle, accurate predictions of system behavior can be made. Furthermore, performance functions can be constructed and built into the model that relate to the objectives laid out in the PBL contract. Finally, different design alternatives can be added to the model using CTDN decision nodes. This allows design choices to be evaluated with respect to the contract requirements prior to any physical implementation, saving time and money for the contractor, and improving the overall solution for the agency.

To demonstrate this idea, refer again to the vehicle model from Figure 7.4. During the design phase, there may be a choice between two different axle systems. Table 7.2 shows that the AX component has an MTBF of 1800, an MTTR of 12, and a repair cost of 2000. Now assume there is a second axle system AX' with an MTBF of 2100, an MTTR of 20, and a repair cost of 2500. Essentially, the idea is that this new axle design is more complex, resulting in higher reliability and therefore a longer mean time to failure. Unfortunately this additional complexity in the design increases the mean repair time, as well as the cost to fix the component. The question that the contractor is then left with is which design alternative should be chosen to best meet the contract requirements.

The correct choice between these axle design alternatives is not immediately evident and depends on many factors, including how these changes to reliability impact risks and what performance functions are most important for the contract. To approach this using the CTDN framework, the vehicle network can be modified to incorporate this decision. Figure 7.6 shows the axle subsystem of the network, with the new decision node `D-AX` highlighted using a dashed line. To parameterize the `AX` node, an intensity matrix is derived once for each design alternative, and paired with the corresponding state in the `D-AX` decision node. Once again, the parameters in the decision node are largely unimportant since the state will be

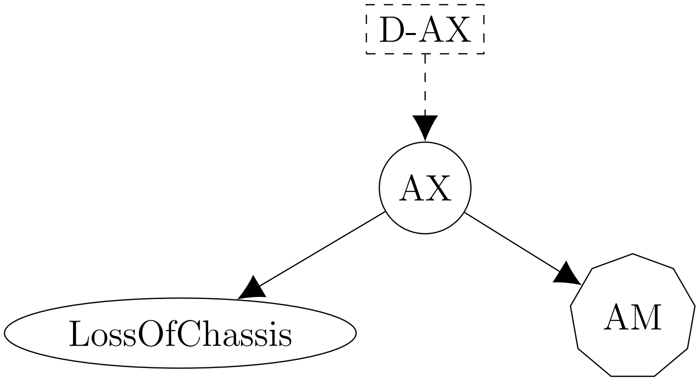


Figure 7.6: The axle subsystem before and after the inclusion of a decision node that encodes different design decisions.

assigned deterministically. With this model, inference can be run once for each design alternative, allowing the performance values to be obtained for each objective. Alternatives who's performances do not meet the contract requirements can be eliminated, and a design choice can be obtained from the remaining valid solutions.

In addition to parameter changes, decision nodes can support structure changes to the graph as well. For example, there may be two design alternatives regarding the power subsystem, where one option includes a redundant power source, while the other alternative omits the secondary battery. Figure 7.7 shows the power subsystem from the vehicle model with the decision node `D-PW2` added as a parent to the `LossOfPower` event. Assuming `PW1` and `PW2` are both Boolean, then the CIM for `LossOfPower` originally consists of four intensity matrices. These four intensity matrices are mapped to the first state of `D-PW2`, and four new intensity matrices are mapped to the second state. With these four new matrices, `LossOfPower` is parameterized according to `PW1` alone as if `PW2` did not exist. As such, when `D-PW2` is in the first state (redundant design), `LossOfPower` is dependent on both `PW1` and `PW2`. When `D-PW2` is in the second state (no redundancy), `LossOfPower` is dependent on only `PW1`, and the edge between `PW2` and `LossOfPower` is effectively severed.

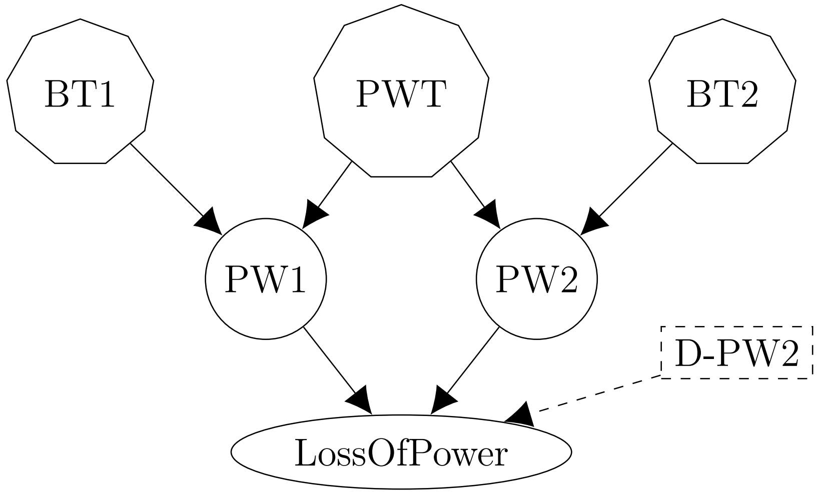


Figure 7.7: The power subsystem before and after the inclusion of a decision node that encodes the decision to support multiple power sources.

These models provide a mathematically sound and inexpensive method for evaluating design choices with respect to contract requirements prior to implementation. In addition, upon fabrication of the system, the model can then be used for diagnostics and prognostics. This is achieved by fixing the decision nodes representing design choices to the actual alternative that was chosen. Going forward, these models can still be used to identify fault and effect likelihoods, as well as to evaluate specified performance functions. Additional decision nodes can be inserted to model possible operational decisions, which can serve as mitigation strategies that have the potential to minimize performance loss.

7.6 Conclusion

In this chapter, we discussed PHM as a specific application for CTBNs. To reduce the barrier to entry when building these models for system prognostics, we have introduced derivation procedures that help to construct CTBNs using commonly available diagnostic and reliability information. We show how both structure and parameters can be obtained using this data, which eliminates the need to apply expensive learning algorithms that depend on often exorbitant amounts of data. In the event that reliability information defines failure or repair rates that are non-exponential, the techniques discussed in Chapter 3 may be used to compute an approximating phase-type distribution, and embed this distribution into the resulting CTBN. The derivation procedures were applied to an extended version of the vehicle model, demonstrating the applicability of these techniques to real-world systems. It is often the case in this domain that system relationships are causal, and the vehicle network is no exception. For this reason, the disjunctive interaction model from Chapter 6 is applied as part of the parameterization procedures, providing a substantial reduction in both the model size and the amount of necessary data. For cases where a node has many parents that were derived from an AND gate in the original fault tree, disjunctive interaction cannot be applied. Instead, the compact representations proposed in Chapter 5 may serve as an alternative approach to managing scalability.

In addition to detailing processes for CTBN construction, this chapter also discussed how the resulting models can be used to perform diagnostics and prognostics by applying evidence corresponding to test results. Furthermore, methods for representing and reasoning over operational decisions were presented. Specifically, we argued for the use of the CTDN framework presented in Chapter 4, which allows for the representation of various scenarios and utilities, which can then be optimized to obtain a mathematically supported best decision during runtime. Finally, we showed that the CTDN framework was not only useful for making operational decisions, but could also be applied to design time decisions as well.

The CTBN derivation algorithms we presented in this chapter provide an automated procedure for constructing models without the need for excessive amounts of historical data. Furthermore, the vehicle network construction and experimentation serves as a tutorial to those seeking to apply the CTBN to PHM tasks. Finally, by providing a framework for optimizing over various design decisions, CTDNs can support tasks like PBL, and can continue to serve as prognostic models capable of describing runtime behavior after fabrication. This lowers the barrier to entry required to apply CTBNs to prognostics and diagnostics, and provides a guideline for how domain-specific algorithms and models may be developed to better suit CTBNs to real-world problems.

CHAPTER EIGHT

CONCLUSION

The contributions in this dissertation share the common goal of making CTBNs more accessible in real-world domains. These contributions can be separated into two distinct categories: those focusing on the representational capabilities of CTBNs, and those concerned with scalability issues. By providing methods for addressing these issues, we hope to reduce the barrier to entry when applying CTBNs in practice. In this final chapter, we provide a brief summary of the contributions presented in this dissertation, and end with directions for future work.

8.1 Contributions

First, Chapter 3 helped to improve the representational capabilities of CTBNs by describing a method for modeling non-parametric transition distributions. It was shown that general-purpose optimization algorithms can be used to minimize the KL-divergence of a phase-type distribution from a target parametric distribution, so long as it is positive and continuous. We then formalized the process for embedding any general phase-type distributions into the intensity matrices of a CTBN. These two contributions provide an automated procedure for specifying non-exponential parametric distributions within the CTBN framework, thereby allowing processes with more complex transition behavior to be modeled.

Next, Chapter 4 continued with the goal of improved representational power by introducing the continuous time decision network. The CTDN was defined formally, and consists of three node types: chance nodes, utility nodes, and decision nodes.

This framework supports the representation of decision problems over processes that change in continuous time. We demonstrated how such a model might be constructed and showed how the framework can be used to evaluate utility values. Optimization algorithms were used to identify the actions that maximize utility in the system, thereby assisting in the decision making process, even in the presence of uncertainty. Just as with the phase-type approximations, the newly introduced CTDN helps in representing additional problems in practical domains.

Chapter 5 shifts its attention to the issue of scalability. Two distinct contributions were presented in this chapter, each corresponding to a different compact representation of the CIMs in a CTBN. The MCIM used compositions of functions to consolidate intensity matrices, thereby reducing the total number of required parameters. The TCIM representation also consolidated intensity matrices but used decision trees rather than discrete functions to encode independences In both cases, the compact representations rely on groupings of identical or similar intensity matrices, which is achieved by applying a clustering algorithm to the CIM. By reducing the number of parameters, scalability concerns may be alleviated, allowing larger processes to be modeled.

Chapter 6 presents disjunctive interaction for CTBNs, which is another approach to managing scalability when working with large models. We showed how the space complexity of a node can be reduced from exponential to linear in the number of parents, so long as the interaction between the parents is disjunctive. We also showed the special case where variables are Boolean, and related this work to the Noisy-OR model in Bayesian Networks. This parameterization can provide a substantial reduction in model size, especially for applications where variables exhibit causal relationships.

Finally, Chapter 7 tied these contributions together by providing an extended example of how CTBNs can be applied to the practical domain of reliability and prognostics for a vehicle model. The modeling process makes use of the CTDN framework to represent decision problems, but also incorporates disjunctive interaction to manage the complexity of the parameterization. Furthermore, this chapter introduces novel methods for deriving the structure and parameters of a CTBN using existing reliability data. This demonstration, as well as the domain-specific techniques presented in the chapter, act as a guideline when applying CTBNs to the field of PHM.

8.2 Future Work

For our future work in non-exponential parametric distributions, we would like to apply our PT distribution learning procedure to a wider array of parametric distributions. It may be possible to estimate the quality of a phase-type approximation based on the underlying characteristics of the target distribution. For instance, phase-type distributions are only capable of approximating continuous distributions, so it is likely that approximations of excessively sharp distributions will be poor.

For Continuous Time Decision Networks, we would like to investigate automated methods for constraining the search space over the trajectory of decision variables. The experiments reported in Chapter 4 made use of domain knowledge to reduce the set of necessary evaluations to a tractable size. We intend to evaluate the effectiveness of gradient based methods, as well as population-based optimization methods like particle swarm optimization to perform a directed search over all possible decisions. This would provide a locally optimal decision while avoiding the need to evaluate every possible decision assignment. We also hope to further refine the CTDN framework by developing learning and inference algorithms that are specifically tailored to work efficiently with this representation.

Concerning the MCIM and TCIM compact representations discussed in Chapter 5, as well as disjunctive interaction described in Chapter 6, we would like to look into hybrid compact structures. As proven, there are conditional CTMPs that cannot be factored fully using either the MCIM or TCIM representation. By combining these approaches, and possibly other structured representations for CTMPs such as Kronecker algebra, it may be possible to achieve even more effective representations. Furthermore, we hope to adapt existing inference algorithms to work on the compact representations directly, potentially allowing for even more efficient inference. By working with the the compact representations directly, improvements may be achievable to the inference time complexity. As is the case in the Bayesian Network literature, it may be that efficient inference algorithms that exploit compact representations may only exist for special cases. If this is the case, it will be necessary to identify when inference can be performed efficiently in terms of network structure and evidence application. The experiments in Chapter 6 also indicated that with large models, specific state combinations become unlikely, presenting a mixing problem. Although this does not influence the compact representations presented in this dissertation, it may negatively influence the quality of approximate inference algorithms. In the future we hope to further study and identify approaches for addressing this issue.

We hope to continue refining the CTBN models used for diagnostics and prognostics, and we intend to investigate other domains where CTBNs may be underutilized. By providing domain-specific algorithms and modeling procedures, the burden placed on researchers and engineers to integrate CTBNs into their current operations will be reduced. We also hope to formalize the process for representing scenarios using decision nodes. This will involve mapping parametric and structure changes to the discrete states of the decision node, and algorithmically updating the network accordingly. Above all, we hope that the contributions presented in this dissertation help to guide CTBNs toward these research aims, and ultimately toward more prominent real-world applications.

REFERENCES CITED

[1] O. O. Aalen. Phase–type distributions in survival analysis. *Scandinavian Journal of Statistics*, 22(4):447–463, 1995.

[2] E. Acerbi and F. Stella. Continuous time Bayesian networks for gene network reconstruction: a comparative study on time course data. In *Bioinformatics Research and Applications*, pages 176–187. Springer, 2014.

[3] E. Acerbi, E. Viganò, M. Poidinger, A. Mortellaro, T. Zelante, and F. Stella. Continuous time bayesian networks identify prdm1 as a negative regulator of th17 cell differentiation in humans. *Scientific reports*, 6, 2016.

[4] E. Acerbi, T. Zelante, V. Narang, and F. Stella. Gene network inference using continuous time bayesian networks: a comparative study and application to th17 cell differentiation. *BMC bioinformatics*, 15(1):387, 2014.

[5] D. Z. Anderson. *Neural information processing systems.* Springer Science & Business Media, 1988.

[6] W. J. Anderson. *Continuous-time Markov chains: An applications-oriented approach.* Springer Science & Business Media, 2012.

[7] G. A. Baker Jr. and J. L. Gammel. The Padé approximant and some related generalizations. In *The Padé Approximant in Theoretical Physics*, volume 71 of *Mathematics in Science and Engineering.* Academic Press, 1970.

[8] J. Banks. *Handbook of Simulation.* John Wiley & Sons, 1998.

[9] W. Bian and D. Tao. Harmonic mean for subspace selection. In *19th International Conference on Pattern Recognition (ICPR)*, pages 1–4. IEEE, 2008.

[10] C. Boutilier, N. Friedman, M. Goldszmidt, and D. Koller. Context-specific independence in bayesian networks. In *Proceedings of the Twelfth international conference on Uncertainty in artificial intelligence*, pages 115–123. Morgan Kaufmann Publishers Inc., 1996.

[11] P. Buchholz, G. Ciardo, S. Donatelli, and P. Kemper. Complexity of kronecker operations on sparse matrices with applications to the solution of markov models. Technical report, Institute for Computer Applications in Science and Engineering, Hampton, VA United States, 1997.

[12] F. Camci and R. B. Chinnam. Dynamic Bayesian networks for machine diagnostics: hierarchical hidden Markov models vs. competitive learning. In

Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN), volume 3, pages 1752–1757, 2005.

[13] D. Cao. *Novel models and algorithms for systems reliability modeling and optimization*. PhD thesis, Wayne State University, 2011.

[14] E. B. Celikkaya, C. R. Shelton, and W. Lam. Factored filtering of continuous-time systems. In F. G. Cozman and A. Pfeffer, editors, *UAI*, pages 61–68. AUAI Press, 2011.

[15] D. Codecasa. *Continuous Time Bayesian Network Classifiers*. PhD thesis, Ph. D. thesis, Universitá degli Studi di Milano-Bicocca. To appear in BOA (Bicocca Open Archive), 2014.

[16] D. Codecasa and F. Stella. A classification based scoring function for continuous time Bayesian network classifiers. In *International Workshop on New Frontiers in Mining Complex Patterns*, pages 35–50. Springer, 2013.

[17] D. Codecasa and F. Stella. Conditional log-likelihood for continuous time Bayesian network classifiers. In *European Conference on Machine Learning and Principles and Practice of Knowledge Discovery in Databases (ECML-PKDD)*, International Workshop New Frontiers in Mining Complex Patterns (NFMCP), 2013.

[18] D. Codecasa and F. Stella. Learning continuous time Bayesian network classifiers. *International Journal of Approximate Reasoning*, 55(8):1728–1746, 2014.

[19] D. Codecasa and F. Stella. Classification and clustering with continuous time Bayesian network models. *Journal of Intelligent Information Systems*, 45(2):187–220, 2015.

[20] D. Codetta-Raiteri and L. Portinale. Generalized continuous time Bayesian networks and their GSPN semantics. *European Workshop on Probabilistic Graphical Models*, pages 105–112, 2010.

[21] D. Codetta-Raiteri and L. Portinale. A gspn based tool to inference generalized continuous time bayesian networks. In *Proceedings of the 7th International Conference on Performance Evaluation Methodologies and Tools*, pages 316–319. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2013.

[22] D. Codetta-Raiteri and L. Portinale. A petri net based tool for the analysis of generalized continuous time bayesian networks. *Theory and Application of Multi-Formalism Modeling*, page 118, 2013.

[23] D. Codetta-Raiteri and L. Portinale. Modeling and analysis of dependable systems through generalized continuous time bayesian networks. In *Reliability and Maintainability Symposium (RAMS), 2015 Annual*, pages 1–6. IEEE, 2015.

[24] I. Cohn. *Mean Field Variational Approximations in Continuous-Time Markov Processes*. PhD thesis, The Hebrew University, 2009.

[25] I. Cohn, T. El-Hay, N. Friedman, and R. Kupferman. Mean field variational approximation for continuous-time Bayesian networks. In *Proceedings of the 25th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 91–100. AUAI Press, 2009.

[26] A. Cutler and O. I. Cordero-Braña. Minimum Hellinger distance estimation for finite mixture models. *Journal of the American Statistical Association*, 91(436):1716–1723, 1996.

[27] P. Dagum, A. Galper, and E. Horvitz. Dynamic network models for forecasting. In *Proceedings of the eighth international conference on uncertainty in artificial intelligence*, pages 41–48. Morgan Kaufmann Publishers Inc., 1992.

[28] P. Dagum, A. Galper, E. Horvitz, and A. Seiver. Uncertain reasoning and forecasting. *International Journal of Forecasting*, 11(1):73–87, 1995.

[29] J. T. de Oliveira. Statistical choice of univariate extreme models. *Statistical Distribution in Scientific Work: Applications in Physical, Social and Life Sciences*, 6:367–387, 1981.

[30] E. Delage, H. Lee, and A. Y. Ng. A dynamic bayesian network model for autonomous 3d reconstruction from a single indoor image. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2418–2428. IEEE, 2006.

[31] P. M. Djuric, J. H. Kotecha, J. Zhang, Y. Huang, T. Ghirmai, M. F. Bugallo, and J. Miguez. Particle filtering. *IEEE Signal Processing Magazine*, 20(5):19–38, 2003.

[32] N. Dojer, A. Gambin, A. Mizera, B. Wilczyński, and J. Tiuryn. Applying dynamic bayesian networks to perturbed gene expression data. *BMC bioinformatics*, 7(1):249, 2006.

[33] M. Dong and Z. Yang. Dynamic Bayesian network based prognosis in machining processes. *Journal of Shanghai Jiaotong University (Science)*, 13:318–322, 2008.

[34] T. El-Hay, I. Cohn, N. Friedman, and R. Kupferman. Continuous-time belief propagation. In J. Frnkranz and T. Joachims, editors, *International Conference on Machine Learning (ICML)*, pages 343–350, 2010.

[35] T. El-Hay, N. Friedman, and R. Kupferman. Gibbs sampling in factorized continuous-time Markov processes. In *Proceedings of the 24th Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 169–178. AUAI Press, 2008.

[36] K. P. Exarchos, G. Rigas, Y. Goletsis, and D. I. Fotiadis. Towards building a dynamic Bayesian network for monitoring oral cancer progression using time-course gene expression data. In *10th IEEE International Conference on Information Technology and Applications in Biomedicine (ITAB), 2010*, pages 1–4, 2010.

[37] Y. Fan. *Continuous Time Bayesian Network Approximate Inference and Social Network Applications*. PhD thesis, University of California Riverside, 2009.

[38] Y. Fan and C. R. Shelton. Sampling for approximate inference in continuous time Bayesian networks. In *Tenth International Symposium on Artificial Intelligence and Mathematics*, 2008.

[39] Y. Fan and C. R. Shelton. Learning continuous-time social network dynamics. In *Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 161–168. AUAI Press, 2009.

[40] Y. Fan, J. Xu, and C. R. Shelton. Importance sampling for continuous time Bayesian networks. *Journal of Machine Learning Research (JMLR)*, 99:2115–2140, 2010.

[41] V. T. Farewell and R. L. Prentice. A study of distributional shape in life testing. *Technometrics*, 19(1):69–75, 1977.

[42] N. Friedman, M. Linial, I. Nachman, and D. Pe'er. Using bayesian networks to analyze expression data. *Journal of computational biology*, 7(3-4):601–620, 2000.

[43] M. Fujita, P. C. McGeer, and J.-Y. Yang. Multi-terminal binary decision diagrams: An efficient data structure for matrix representation. *Formal methods in system design*, 10(2-3):149–169, 1997.

[44] E. Gatti. *Graphical models for continuous time inference and decision making*. PhD thesis, Università degli Studi di Milano-Bicocca, 2011.

[45] E. Gatti, D. Luciani, and F. Stella. A continuous time bayesian network model for cardiogenic heart failure. *Flexible Services and Manufacturing Journal*, 24(4):496–515, 2012.

[46] A. L. Gibbs and F. E. Su. On choosing and bounding probability metrics. *International Statistical Review*, 70(3):419–435, 2002.

[47] Y. Gong, L. Wang, R. Guo, and S. Lazebnik. Multi-scale orderless pooling of deep convolutional activation features. In *European Conference on Computer Vision*, pages 392–407. Springer, 2014.

[48] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. C. Courville, and Y. Bengio. Maxout networks. *ICML (3)*, 28:1319–1327, 2013.

[49] K. Gopalratnam, H. Kautz, and D. S. Weld. Extending continuous time Bayesian networks. In *Proceedings of the National Conference on Artificial Intelligence*, volume 20, page 981. AAAI Press, 2005.

[50] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Cluster validity methods: Part i. *ACM Sigmod Record*, 31(2):40–45, 2002.

[51] M. Halkidi, Y. Batistakis, and M. Vazirgiannis. Clustering validity checking methods: Part ii. *ACM Sigmod Record*, 31(3):19–27, 2002.

[52] J. Hallgren. Structure learning and mixed radix representation in continuous time bayesian networks. Technical report, Royal Institute of Technology, Stockholm Sweden, 2016.

[53] R. Herbrich, T. Graepel, and B. Murphy. Structure from failure. In *Proceedings of the 2nd USENIX Workshop on Tackling Computer Systems Problems with Machine Learning Techniques*, pages 1–6. USENIX Association, 2007.

[54] N. J. Higham. The scaling and squaring method for the matrix exponential revisited. *SIAM Journal on Matrix Analysis and Applications*, 26(4):1179–1193, 2005.

[55] J. Hoey, R. St-Aubin, A. Hu, and C. Boutilier. Spudd: Stochastic planning using decision diagrams. In *Proceedings of the Fifteenth conference on Uncertainty in artificial intelligence*, pages 279–288. Morgan Kaufmann Publishers Inc., 1999.

[56] J. D. Kalbfleisch and R. L. Prentice. *The Statistical Analysis of Failure Time Data*, chapter Failure Time Models. John Wiley & Sons, 2011.

[57] K. F. Kan and C. R. Shelton. Solving structured continuous-time Markov decision processes. In *Proceedings of 10th International Symposium on Artificial Intelligence and Mathematics*, 2008.

[58] J. Kennedy, R. Eberhart, et al. Particle swarm optimization. In *Proceedings of IEEE International Conference on Neural Networks*, volume 4, pages 1942–1948. Perth, Australia, 1995.

[59] M.-S. Kim, I.-H. Yang, and H.-J. Yu. Maximizing distance between GMMs for speaker verification. In *Fourth International Conference on Natural Computation*, pages 175–178. IEEE, 2008.

[60] U. B. Kjaerulff and A. L. Madsen. Bayesian networks and influence diagrams. *Springer Science+ Business Media*, 200:114, 2008.

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

[62] D. Koller and B. Milch. Multi-agent influence diagrams for representing and solving games. *Games and economic behavior*, 45(1):181–221, 2003.

[63] A. N. Kolmogorov. *Foundations of Probability*. Chelsea Publishing Company, New York, 1933.

[64] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[65] U. D. Kumar, D. Nowicki, J. E. Ramirez-Marquez, and D. Verma. A goal programming model for optimizing reliability, maintainability and supportability under performance based logistics. *International Journal of Reliability, Quality and Safety Engineering*, 14(03):251–261, 2007.

[66] G. Latouche and V. Ramaswami. *Introduction to matrix analytic methods in stochastic modeling*. SIAM, 1999.

[67] P. W. Laud and J. G. Ibrahim. Predictive model selection. *Journal of the Royal Statistical Society. Series B (Methodological)*, 57(1):247–262, 1995.

[68] B. B. Le Cun, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. Handwritten digit recognition with a back-propagation network. In *Advances in neural information processing systems*. Citeseer, 1990.

[69] J. Makhoul. Artificial neural networks. *Investigative radiology*, 25(6):748–750, 1990.

[70] R. T. Marler and J. S. Arora. Survey of multi-objective optimization methods for engineering. *Structural and Multidisciplinary Optimization*, 26(6):369–395, 2004.

[71] K. R. McNaught and A. Zagorecki. Using dynamic Bayesian networks for prognostic modelling to inform maintenance decision making. In *IEEE International Conference on Industrial Engineering and Engineering Management, 2009. IEEM 2009.*, pages 1155–1159, 2009.

[72] K. Medjaher, J. Moya, and N. Zerhouni. Failure prognostic by using dynamic Bayesian networks. *Dependable Control of Discrete Systems*, 1:291–296, 2009.

[73] A. Mesaros, T. Virtanen, and A. Klapuri. Singer identification in polyphonic music using vocal separation and pattern recognition methods. In *ISMIR*, pages 375–378, 2007.

[74] B. Miasojedow and W. Niemiro. Geometric ergodicity of rao and teh's algorithm for markov jump processes and ctbns. *arXiv preprint arXiv:1606.08160*, 2016.

[75] B. Miasojedow, W. Niemiro, J. Noble, and K. Opalski. Metropolis-type algorithms for continuous time Bayesian networks. *arXiv preprint arXiv:1403.4035*, 2014.

[76] M. Mitchell. *An Introduction to Genetic Algorithms.* MIT Press, 1998.

[77] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.

[78] A. Muller, M. Suhner, and B. Iung. Formalisation of a new prognosis model for supporting proactive maintenance implementation on industrial system. *Reliability Engineering & System Safety*, 93(2):234–253, 2008.

[79] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.

[80] B. Ng, A. Pfeffer, and R. Dearden. Continuous time particle filtering. In *International Joint Conference on Artificial Intelligence (IJCAI)*, volume 19, pages 1360–1365, 2005.

[81] U. Nodelman. *Continuous Time Bayesian Networks.* PhD thesis, Stanford University, Stanford, California, 2007.

[82] U. Nodelman and E. Horvitz. Continuous time Bayesian networks for inferring users' presence and activities with extensions for modeling and evaluation. *Microsoft Research*, 2003.

[83] U. Nodelman, D. Koller, and C. R. Shelton. Expectation propagation for continuous time Bayesian networks. In *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence*, pages 431–440. AUAI Press, 2005.

[84] U. Nodelman, C. Shelton, and D. Koller. Continuous time Bayesian networks. In *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 378–387, 2002.

[85] U. Nodelman, C. R. Shelton, and D. Koller. Learning continuous time Bayesian networks. In *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 451–458. Morgan Kaufmann Publishers Inc., 2002.

[86] U. Nodelman, C. R. Shelton, and D. Koller. Expectation maximization and complex duration distributions for continuous time Bayesian networks. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[87] J. R. Norris. *Markov chains*, volume 2. Cambridge university press, 1998.

[88] M. Opper and G. Sanguinetti. Variational inference for Markov jump processes. In *Advances in Neural Information Processing Systems*, pages 1105–1112, 2008.

[89] K. Orphanou, A. Stassopoulou, and E. Keravnou. Dbn-extended: a dynamic bayesian network model extended with temporal abstractions for coronary heart disease prognosis. *IEEE journal of biomedical and health informatics*, 20(3):944–952, 2016.

[90] D. B. Parker. Learning logic. Technical Report TR-47, Sloan School of Management, Massachusetts Institute of Technology, Cambridge, MA United States, 1985.

[91] V. Pavlovic, J. M. Rehg, T.-J. Cham, and K. P. Murphy. A dynamic bayesian network approach to figure tracking using learned dynamic models. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 1, pages 94–101. IEEE, 1999.

[92] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufmann, San Mateo, CA, 1988.

[93] L. Perreault, J. Sheppard, H. King, and L. Sturlaugson. Using continuous-time Bayesian networks for standards-based diagnostics and prognostics. In *IEEE AUTOTESTCON Proceedings 2014*, pages 198–204, 2014.

[94] L. Perreault and J. W. Sheppard. Compact structures for continuous time Bayesian networks. Technical report, Montana State University, 2017.

[95] L. Perreault and J. W. Sheppard. Risk-based prognostics and health management. Technical report, Montana State University, 2017.

[96] L. Perreault, S. Strasser, M. Thornton, and J. Sheppard. A noisy-or model for continuous time Bayesian networks. In *Proceedings of the Twenty-Ninth International Florida Artificial Intelligence Research Society Conference*, 2016.

[97] L. Perreault, M. Thornton, R. Goodman, and J. W. Sheppard. Extending continuous time Bayesian networks for parametric distributions. In *IEEE SSCI*, 2015.

[98] L. Perreault, M. Thornton, and J. W. Sheppard. Deriving prognostic continuous time Bayesian networks from fault trees. In *Annual Conference of the Prognostics and Health Management Society 2016*, 2016.

[99] L. Perreault, M. Thornton, and J. W. Sheppard. Embedding phase-type distributions in continuous time Bayesian networks. *Submitted to the Journal of Artificial Intelligence Research (JAIR)*, 2016.

[100] L. Perreault, M. Thornton, and J. W. Sheppard. Valuation and optimization for performance based logistics using continuous time Bayesian networks. *IEEE AUTOTESTCON*, pages 1–10, 2016.

[101] L. Perreault, M. Thornton, J. W. Sheppard, and J. DeBruycker. Disjunctive interaction in continuous time Bayesian networks. *Submitted to the International Journal of Approximate Reasoning (IJAR)*, 2016.

[102] L. Perreault, M. Thornton, S. Strasser, and J. W. Sheppard. Deriving prognostic continuous time Bayesian networks from D-matrices. In *IEEE AUTOTESTCON Proceedings, 2015*, pages 152–161, 2015.

[103] L. Perreault, M. P. Wittie, and J. Sheppard. Communication-aware distributed pso for dynamic robotic search. In *IEEE Symposium on Swarm Intelligence (SIS), 2014*, pages 1–8. IEEE, 2014.

[104] A. Pfeffer. Asynchronous dynamic Bayesian networks. In *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[105] L. Portinale and D. Codetta-Raiteri. Generalizing continuous time Bayesian networks with immediate nodes. In *Workshop on Graph Structures for Knowledge Representation and Reasoning*, 2009.

[106] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vettering. *Combinatorial Minimization: Method of Simulated Annealing.* Cambridge University Press, 1986.

[107] S. Qiao, C. Tang, H. Jin, T. Long, S. Dai, Y. Ku, and M. Chau. PutMode: prediction of uncertain trajectories in moving objects databases. *Applied Intelligence*, 33(3):370–386, 2010.

[108] H. Raiffa. Decision analysis: introductory lectures on choices under uncertainty. *MD computing: computers in medical practice*, 10(5):312, 1968.

[109] S. Rajarshi and M. Rajarshi. Bathtub distributions: a review. *Communications in Statistics — Theory and Methods*, 17, 1988.

[110] V. Rao and Y. W. Teh. Mcmc for continuous-time discrete-state systems. In *Advances in Neural Information Processing Systems*, pages 701–709, 2012.

[111] V. Rao and Y. W. Teh. Fast MCMC sampling for Markov jump processes and extensions. *Journal of Machine Learning Research (JMLR)*, 14(1):3295–3320, 2013.

[112] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning internal representations by error propagation. Technical report, DTIC Document, 1985.

[113] F. Sahin, M. Ç. Yavuz, Z. Arnavut, and Ö. Uluyol. Fault diagnosis for airplane engines using Bayesian networks and distributed particle swarm optimization. *Parallel Computing*, 33(2):124–143, 2007.

[114] S. Saria, U. Nodelman, and D. Koller. Reasoning at the right time granularity. In *Proceedings of the 23rd Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

[115] R. D. Shachter. Evaluating influence diagrams. *Operations research*, 34(6):871–882, 1986.

[116] R. D. Shachter. Probabilistic inference and influence diagrams. *Operations Research*, 36(4):589–604, 1988.

[117] G. R. Shafer and P. P. Shenoy. Probability propagation. *Annals of Mathematics and Artificial Intelligence*, 2(1-4):327–351, 1990.

[118] C. R. Shelton and G. Ciardo. Tutorial on structured continuous-time Markov processes. *Journal of Artificial Intelligence Research*, 51:725–778, 2014.

[119] J. Sheppard and S. Butcher. A formal analysis of fault diagnosis with D-matrices. *Journal of Electronic Testing*, 23(4):309–322, 2007.

[120] D. Shi, X. Tang, and J. You. An intelligent system based on adaptive CTBN for uncertainty reasoning in sensor networks. *Intelligent Automation & Soft Computing*, 16(3):337–351, 2010.

[121] D. Shi and J. You. Update rules for parameter estimation in continuous time Bayesian network. In *PRICAI 2006: Trends in Artificial Intelligence*, pages 140–149. Springer, 2006.

[122] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep fisher networks for large-scale image classification. In *Advances in neural information processing systems*, pages 163–171, 2013.

[123] W. R. Simpson and J. W. Sheppard. *System Test and Diagnosis.* Kluwer Academic Publishers, Norwell, MA, 1994.

[124] A. Sols, D. Nowick, and D. Verma. Defining the fundamental framework of an effective performance-based logistics (PBL) contract. *Engineering Management Journal*, 19(2):40–50, 2007.

[125] A. Srinivasan, T. Ham, S. Malik, and R. K. Brayton. Algorithms for discrete function manipulation. In *Computer-Aided Design, 1990. ICCAD-90. Digest of Technical Papers., 1990 IEEE International Conference on*, pages 92–95. IEEE, 1990.

[126] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers. Exploiting structural duplication for lifetime reliability enhancement. In *Proceedings of the 32nd Annual International Symposium on Computer Architecture*, pages 520–531. IEEE Computer Society, 2005.

[127] F. Stella and Y. Amer. Continuous time Bayesian network classifiers. *Journal of biomedical informatics*, 45(6):1108–1119, 2012.

[128] W. J. Stewart. *Probability, Markov chains, queues, and simulation: the mathematical basis of performance modeling*. Princeton University Press, 2009.

[129] S. Strasser and J. Sheppard. An empirical evaluation of Bayesian networks derived from fault trees. In *Proceedings of the IEEE Aerospace Conference*, pages 1–13, March 2013.

[130] L. Studer, L. Paulevé, C. Zechner, M. Reumann, M. R. Martínez, and H. Koeppl. Marginalized continuous time bayesian networks for network reconstruction from incomplete observations. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, pages 2051–2057. AAAI Press, 2016.

[131] L. Sturlaugson, L. Perreault, and J. W. Sheppard. Factored performance functions and decision making in continuous time Bayesian networks. *Journal of Applied Logic*, 2016.

[132] L. Sturlaugson and J. W. Sheppard. Factored performance functions with structural representation in continuous time bayesian networks. In *FLAIRS Conference*, 2014.

[133] L. Sturlaugson and J. W. Sheppard. Factored performance functions with structural representation in continuous time Bayesian networks. In *The Twenty-Seventh International FLAIRS Conference*, 2014.

[134] L. Sturlaugson and J. W. Sheppard. The long-run behavior of continuous time bayesian networks. In *UAI*, pages 842–851. Citeseer, 2015.

[135] L. Sturlaugson and J. W. Sheppard. Sensitivity analysis of continuous time bayesian network reliability models. *SIAM/ASA Journal on Uncertainty Quantification*, 3(1):346–369, 2015.

[136] L. Sturlaugson and J. W. Sheppard. Uncertain and negative evidence in continuous time bayesian networks. *International Journal of Approximate Reasoning*, 70:99–122, 2016.

[137] G. J. Szekely and M. L. Rizzo. Hierarchical clustering via joint between-within distances: Extending ward's minimum variance method. *Journal of Classification*, 22(2):151–183, 2005.

[138] S. Villa, M. Rossetti, et al. Learning continuous time Bayesian network classifiers using mapreduce. *Journal of Statistical Software*, 62(3):1–25, 2014.

[139] S. Villa and F. Stella. A continuous time Bayesian network classifier for intraday FX prediction. *Quantitative Finance*, 2014.

[140] S. Villa, F. Stella, K. Martiny, R. Möller, C. Muise, J. C. Beck, S. A. McIlraith, A. Moreo Fernández, A. Esuli, F. Sebastiani, et al. Learning continuous time bayesian networks in non-stationary domains. *Journal of Artificial Intelligence Research*, 57:1–37, 2016.

[141] J. Weiss, S. Natarajan, and D. Page. Multiplicative forests for continuous-time processes. In *Advances in neural information processing systems*, pages 458–466, 2012.

[142] J. C. Weiss, S. Natarajan, and C. D. Page, Jr. Learning when to reject an importance sample. In *AAAI (Late-Breaking Developments)*, volume WS-13-17 of *AAAI Workshops*. AAAI, 2013.

[143] P. Werbos. *Beyond regression: New tools for prediction and analysis in the behavioral sciences*. PhD thesis, Harvard University, Cambridge, MA, 1974.

[144] B. K. Wong, T. A. Bodnovich, and Y. Selvi. A bibliography of neural network business applications research: 1988–september 1994. *Expert Systems*, 12(3):253–261, 1995.

[145] B. K. Wong, V. S. Lai, and J. Lam. A bibliography of neural network business applications research: 1994–1998. *Computers & Operations Research*, 27(11):1045–1076, 2000.

[146] W. A. Woodward, P. Whitney, and P. W. Eslinger. Minimum Hellinger distance estimation of mixture proportions. *Journal of Statistical Planning and Inference*, 48(3):303–319, 1995.

[147] J. Xu. *A Continuous Time Bayesian Network Approach for Intrusion Detection*. PhD thesis, University of California Riverside, 2010.

[148] J. Xu and C. R. Shelton. Continuous Time Bayesian Networks for Host Level Network Intrusion Detection. In W. Daelemans, B. Goethals, and K. Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, volume 5212 of *Lecture Notes in Computer Science*, pages 613–627. Springer Berlin / Heidelberg, 2008.

[149] J. Xu and C. R. Shelton. Intrusion detection using continuous time Bayesian networks. *Journal of Artificial Intelligence Research (JAIR)*, 39(1):745–774, 2010.

[150] S. Yang, T. Khot, K. Kersting, and S. Natarajan. Learning continuous-time bayesian networks in relational domains: A non-parametric approach. In *AAAI*, pages 2265–2271, 2016.

[151] J. S. Yedidia, W. T. Freeman, and Y. Weiss. Constructing free-energy approximations and generalized belief propagation algorithms. *IEEE Transactions on Information Theory*, 51(7):2282–2312, 2005.

[152] L. Yu, H. Chen, J. Zhou, H. Yin, and H.-Z. Huang. Fatigue life prediction of low pressure turbine shaft of turbojet engine. *International Journal of Turbo & Jet-Engines*, 2016.

[153] V. M. Zolotarev. Probability metrics. *Theory of Probability and its Applications*, 28(2):264–287, 1983.

[154] M. Zou and S. D. Conzen. A new dynamic bayesian network (dbn) approach for identifying gene regulatory networks from time course microarray data. *Bioinformatics*, 21(1):71–79, 2005.