EXTENSIONS TO MODELING AND INFERENCE IN CONTINUOUS TIME BAYESIAN NETWORKS

by

Liessman Eric Sturlaugson

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

of

Doctor of Philosophy

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

April, 2014

DEDICATION

To Ginny, my beautiful wife

ACKNOWLEDGEMENTS

I am very grateful to my advisor, Dr. John Sheppard, for welcoming me to Montana State University and introducing me to the wide world of artificial intelligence. He has made me a better (and, one hopes, faster) researcher and writer. He has both challenged and encouraged me when needed through the vicissitudes of research, and for that I am grateful. I thank Dr. Binhai Zhu, Dr. Qing Yang, Dr. Tomáš Gedeon, Dr. Jordy Hendrikx, Dr. John Paxton, Ms. Kathryn Hollenback, Ms. Shelly Shroyer, and Ms. Jeannette Radcliffe for their help during my time at MSU.

I thank Warren and Adam for motivating me as I tried to keep up with them. I thank Natalie, Bethany, and Eve for being patient with me. I thank Stuart for being the best buddy a guy could wish for. I thank my dad, the most insightful person I know. I thank my mom, the most self-less person I know. I thank Tara, Bueze, William, Daniel, Tom, Linda, and Josh. I thank my wife, Ginny, for encouraging me through the late nights with texts, prayers, and brownies.

I thank the members of the Numerical Intelligent Systems Laboratory for providing such an enjoyable work environment. Of those, I especially thank Shane Strasser for convincing me to pursue graduate school, letting me be his roommate and labmate, and joining me in my mild Dr. Pepper addiction.

Above all, I am thankful to my Lord and Savior Jesus Christ, who lets me glorify and enjoy him through computer science.

Funding Acknowledgement

This research was partially supported by Phase II of NAVY STTR N10A-T009 and by Phase I of NASA STTR T13.01-9887.

TABLE OF CONTENTS

TABLE OF CONTENTS – CONTINUED

TABLE OF CONTENTS – CONTINUED

TABLE OF CONTENTS – CONTINUED

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES – CONTINUED

LIST OF ALGORITHMS

ABSTRACT

The continuous time Bayesian network (CTBN) enables reasoning about complex systems in continuous time by representing a system as a factored, finite-state, continuous-time Markov process. The dynamics of the CTBN are described by each node's conditional intensity matrices, determined by the states of the parents in the network. As the CTBN is a relatively new model, many extensions that have been defined with respect to Bayesian networks (BNs) have not yet been extended to CTBNs. This thesis presents five novel extensions to CTBN modeling and inference.

First, we prove several complexity results specific to CTBNs. It is known that exact inference in CTBNs is NP-hard due to the use of a BN for the initial distribution. We prove that exact inference in CTBNs is still NP-hard, even when the initial states are given, and prove that approximate inference in CTBNs, as with BNs, is also NP-hard. Second, we formalize performance functions for the CTBN and show how they can be factored in the same way as the network, even when the performance functions are defined with respect to interaction between multiple nodes. Performance functions extend the model, allowing it to represent complex, user-specified functions of the behaviors of the system. Third, we present a novel method for node marginalization called "node isolation" that approximates a set of conditional intensity matrices with a single unconditional intensity matrix. The method outperforms previous node marginalization techniques in all of our experiments by better describing the long-term behavior of the marginalized nodes. Fourth, using the node isolation method we developed, we show how methods for sensitivity analysis of Markov processes can be applied to the CTBN while exploiting the conditional independence structure of the network. This enables efficient sensitivity analysis to be performed on our CTBN performance functions. Fifth, we formalize both uncertain and negative types of evidence in the context of CTBNs and extend existing inference algorithms to be able to support all combinations of evidence types. We show that these extensions make the CTBN more powerful, versatile, and applicable to real-world domains.

CHAPTER 1

INTRODUCTION

In this chapter, we present the motivation for advancing temporal reasoning, specifically in extending modeling and inference in continuous time Bayesian networks (CTBNs[1]). Temporal reasoning in artificial intelligence is the process of answering queries about the behavior of a system that is changing in time. After a brief survey of other temporal modeling approaches, we contrast these models with the CTBN, showing the CTBN as a distinct model. We then summarize our major contributions and conclude with an overview of each of the subsequent chapters of this dissertation.

1.1 Motivation

Temporal models serve an important role in many fields in which systems are observed to be changing through time. For example, temporal models are used for automatic speech recognition [2] and automatic handwriting recognition [3]. Medical professionals use such models to track the likely progression of a disease and use it to inform their prognoses [4, 5]. Similarly, maintenance engineers use such models to track degradation of a system and predict its most likely time of failure [6, 7]. Chemists use temporal models to represent and reason about how chemical reactions change under different experimental conditions [8]. Biologists and ecologists use temporal models to predict population dynamics [9, 10]. Computer security experts use temporal models to monitor systems and detect trends that indicate security anoma-

[1] Although identically named, the CTBN of [1] is *not* the same model as discussed in this dissertation.

lies, for use in intrusion detection systems [11] and malware propagation analysis [12]. The financial sector relies on temporal models for financial forecasting [13, 14]. The list could go on. In short, temporal models can be applicable to any domain in which change occurs.

After a temporal model is constructed, it can be used to reason about this change, such as the most likely progression of the state of a system. Using the model to reason about the underlying system being modeled is a process called inference. Reasoning about complex systems that are evolving in time is a difficult endeavor [15]. Temporal models often attempt to track multiple variables that are interacting in time. As the number of variables increases and the interactions between variables become more complex, the computations necessary for inference become more difficult. Furthermore, there are different ways of representing time that must be taken into consideration [16]. These choices on the representation of time, such as viewing time as a sequence of discrete timesteps or as a continuous interval, is also a factor driving both the representational power of the model and the complexity of performing inference over the model. The CTBN is one such continuous-time model that attempts to simplify the representation of the system into a set of interdependent subsystems, allowing modelers more control and flexibility over the complexity of inference over the model [17, 18]. The CTBN is an example of a graphical model, in that subsystems are represented with nodes and dependencies between subsystems are represented with arcs. The dynamics of each subsystem are specified by the parameters of the corresponding node in the graph.

The strength of a model lies in its ability to represent the system, represent observations about the system, and reason about the system given those observations. While several inference algorithms have been developed for CTBNs, there is still much room for improvement in the types of inference that the model can support and in the types of observations that the model can represent. This dissertation describes a number of valuable extensions to CTBN modeling and inference that make the model more useful, flexible, and applicable.

1.2 Temporal Modeling

Many temporal models and their variations have been proposed for time-series analysis and forecasting. Early models focused on univariate signal processing. These include the autoregressive integrated moving average (ARIMA) model and its many forms, in which the output of a variable in the model maintains a linear dependence on the variable's previous values [19]. Multivariate models extend from the univariate case to allow interaction between multiple variables through time, rather than a single variable dependent only on itself. Filtering methods, such as Kalman filters [20, 21] and particle filters [22, 23], can represent non-linear dynamical systems with continuous variables in continuous time and can be used for online tracking and prediction. Markov chains have been used extensively for modeling and as a means to reason about discrete-state, discrete-time stochastic processes, while Markov processes extend this idea to represent continuous-time Markov chains [24]. Hidden Markov models (HMMs) [25] and dynamic Bayesian networks (DBNs) [26] build upon Markov chains and are also used extensively for sequences of observations within a Bayesian framework. Recurrent neural networks extend the original neural network formulation to reason about sequences of model input and output in discrete time [27] and continuous time [28]. A relatively recent temporal model, the continuous time Bayesian network, builds upon Markov processes and Bayesian networks (BNs) to represent discrete variables in continuous time [17].

While these represent a broad overview of temporal models, each model is suited for different purposes and each exhibits strengths and weaknesses. Some of the distinguishing features of the CTBN are that it is a multivariate, discrete-state, and continuous-time model.

The ability to handle multivariate systems differentiates the CTBN from univariate models, such as ARIMA models. Because the CTBN is multivariate, it is able to model more complex domains. Univariate temporal models simplify to a single variable that is dependent only on its previous values in time. Multivariate temporal models, on the other hand, allow for interaction between multiple variables through time.

The CTBN is specifically designed for discrete-state systems, which differentiates it from the continuous-state models, such as Kalman filters, particle filters, and other state-space models. The choice between a discrete-state and continuous-state model depends heavily on the application. If the variables to be modeled are already categorical, a discrete-state model would probably be better suited to the problem. For example, the CTBN of [29] was learned from survey data and models an individual through time. This model includes four discrete nodes, consisting of marital status {married, not married}, smoking {smoker, a non-smoker}, employment status {student, employment, unemployment}, and number of children {0, 1, 2+}. These variables are naturally discrete, and constructing some mapping from these discrete states to a continuous space for use in a continuous-state model would not make sense.

Furthermore, the CTBN is a continuous-time model, which differentiates it from Markov chains, HMMs, and DBNs. In some temporal applications, the model only needs to represent sequence, not duration. In this case, the CTBN would probably be less applicable. For other applications, duration is important, such as in reliability modeling when querying mean time to failure. Discrete-time models can incorporate observations and be queried only at pre-determined discrete points in time, while continuous-time models lift this restriction. Discrete-time models could be constructed with finer time granularity, but this is often at the expense of increased computational complexity of reasoning over these models.

The Markov process model bears the closest resemblance to the CTBN. This is because the CTBN is a factored Markov process, in much the same way that a DBN is a factored Markov chain. Through this factorization, the CTBN is able to model more complex systems than a single Markov process before inference becomes intractable.

Analogous to the BN and DBN models, the graphical structure of a CTBN can provide an intuitive interpretation of interactions between variables [30]. Similarly, the parameters are relatively easily understood, consisting of expected times within states and transition probabilities between states. This is in contrast to neural networks, in which the network parameters are difficult to interpret [31, 32]. Thus CTBNs can—and have been—manually constructed by domain experts [33].

Lastly, note that according to the No Free Lunch theorems [34, 35], no particular model can be expected to dominate all other models for all problem domains. That is, while other temporal models will be superior to the CTBN on some problems, the CTBN will be superior on others, and we expect at least some of these to be significant, real-world problems. Therefore, further research and development of the CTBN is a valuable contribution to the state-of-the-art in temporal modeling and reasoning.

1.3 Contributions

This thesis contributes a number of important and valuable advancements to inference and modeling in CTBNs. They are listed as follows.

- We present formal complexity proofs for the NP-hardness of exact and approximate inference in CTBNs. These complexity proofs provide insight into the problem of propagating probabilities through time. They further motivate the development of more efficient approximate inference techniques. They also provide a step toward further results that show provably tractable exact or approximate inference for certain special cases of CTBNs. They inform modelers of important considerations when balancing the complexity of the model with tractable inference. To date, however, almost all of the complexity results relevant to CTBNs have been derived from the NP-hardness of the Bayesian network used for the initial distribution. Instead, we advance the understanding of CTBN inference complexity by considering the process of propagating probabilities through time, which is the primary task for CTBNs.

- We extend the CTBN model by introducing performance function nodes that place user-defined values on the behavior of the model. Whereas the behavior of the model is defined by each node's parameters, performance function nodes offer a formal way to represent, quantify, and reason about more complex behaviors of the network. This allows for higher-level reasoning about the system, moving from inferring probabilities of states through time to estimating nonlinear expected values that users have placed on more complex behaviors of the system. For example, these could be cost/reward values when certain transitions occur or certain states are visited for a certain length of time.

- We show two methods for performing node isolation, an approximate node marginalization technique that uses parent-child relationships in the CTBN to isolate the child node. The first method is a sampling method that estimates parameters for the child node so that all incoming arcs can be removed. This serves as a baseline for the second method, which is a closed-form solution that more efficiently computes these parameters. This allows inference to avoid working with the entire CTBN all at once by isolating the nodes of the CTBN in a top-down manner. Because cycles are allowed in a CTBN, we also show an iterative application of the node isolation method to isolate nodes in a cycle without having to deal with the entire cycle all at once. This approximation reduces the complexity from being exponential in the number of nodes in the cycle *and* the number of parents for all nodes in the cycle to just being exponential in the maximum number of parents of any node in the network.

- We present the first algorithms for sensitivity analysis specific to CTBNs. Sensitivity analysis is another form of inference, one that queries the behavior of the model, given changes in the model parameters rather than traditional observations. It is useful for measuring the robustness of the model and detecting which parameters most influence network performance. We then show how node isolation takes advantage of the factored nature of the CTBN model to be able to more efficiently perform sensitivity analysis.

- We extend CTBN modeling and inference to be able to handle uncertain and negative types of evidence. This makes the CTBN more powerful and versatile, allowing for further representation and reasoning under uncertainty. It allows the inference algorithms to incorporate knowledge about the noisiness and robustness of the evidence. It allows for more varied types of evidence, such as when the state is unknown over an interval of time, but some states can be ruled out. For these types of models, it is important to know not only the most probable answer, but to also know the confidence with which we can trust the most probable answer. The introduction of uncertain and negative evidence into the CTBN is an important extension that had not been previously formulated.

As a newer model than the BN, the CTBN has not yet undergone the extensive development enjoyed by the BN. However, many of the extensions defined for the BN model, while they do not directly translate to the CTBN, are similarly valuable for the CTBN model and inference process. Therefore, we have developed and demonstrate a wide range of extensions to the CTBN model and inference that starts to bridge this gap.

1.4 Organization

This section describes the organization of the remaining chapters of this dissertation and gives a brief overview of the focus of each chapter.

In Chapter 2, we review the background work common to this entire thesis. We start by defining Bayesian networks, the static model with which the CTBN shares its name. From there, we define the DBN, the temporal version of the BN. Then we define discrete-state, continuous-time Markov chains, upon which CTBNs are built. Next, we formally define CTBNs and describe an example network. Then we survey the literature describing the subsequent development of CTBNs, including inference algorithms and model learning algorithms, as well as CTBN extensions and application areas that have been researched to date. Each of the subsequent chapters focuses on a different extension to the CTBN. As such, each chapter begins with additional background work relevant to its own contributions.

In Chapter 3, we start with a review of the theoretical work related to complexity of CTBNs. We extend the theory of CTBNs by proving three new theorems, that exact inference in CTBNs is NP-hard and that approximate inference (both absolute and relative) in CTBNs is also NP-hard. These proofs draw on analogous complexity proofs for BNs, which we briefly review.

In Chapter 4, we extend CTBNs with factored performance functions. Whereas the CTBN model describes the dynamics of a system, the performance functions attach user-defined cost/reward values to system behaviors. To handle values defined over complex interactions between multiple nodes, we show how the performance functions can remain factored by augmenting the CTBN structure.

In Chapter 5, we develop the two node isolation methods and show how they can be used to estimate the probabilities of the nodes of the CTBN evolving through time starting from an initial distribution. We also present experiments for the node isolation methods applied iteratively to cycles and to a larger, real-world network.

In Chapter 6, we extend sensitivity analysis to CTBNs. Whereas traditional inference in CTBNs calculates the probabilities given observations, sensitivity analysis measures how changes in the network parameters affect network performance. We show current work in applying perturbation realization, developed for sensitivity analysis of Markov processes, to the CTBN. For the CTBN, sensitivity analysis can be applied to different subnetworks individually through the node isolation methods.

In Chapter 7, we start by reviewing the types of continuous-time evidence currently being used in CTBNs. The concepts of uncertain evidence and negative evidence, while defined for BNs, had not yet been extended to CTBNs. Furthermore, the temporal nature of the evidence leads to a rich variety of evidence that the CTBN can support. We formalize, categorize, and prove relationships between these types of evidence. The addition of uncertain and negative evidence also necessitates extensions to existing inference algorithms to be able to support them. We extend exact inference and develop a rejection sampling technique as our baseline methods, and show how importance sampling is able to include these generalized types of evidence.

In Chapter 8, we conclude with a summary of our main results and discuss areas for future research.

CHAPTER 2

BACKGROUND WORK

This chapter presents the background work for understanding the CTBN model and how it can be used. We start by reviewing Bayesian networks and Markov processes. Then we formally define the CTBN and present a thorough survey of current literature for CTBN extensions and applications. Because each chapter focuses on different extensions to the CTBN, most chapters include additional background work relevant to their own topic.

2.1 Bayesian Networks

In this section, we introduce Bayesian networks and dynamic Bayesian networks. One of the key concepts behind the Bayesian network model is in its use of conditional independence. We start by defining independence of random variables.

Definition 2.1.1 (Independence)**.** *Two random variables A and B are independent if and only if*

$$P(A, B) = P(A)P(B).$$

In other words, the joint probability of A and B is equal to the product of their marginals.

Note that if A and B are independent, then

$$P(A|B) = \frac{P(A, B)}{P(B)} = \frac{P(A)P(B)}{P(B)} = P(A),$$

which means that evidence about B does not change the distribution of A. Conditioning on other evidence could also form independence between variables, which gives rise to the definition of conditional independence.

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

$$P(A|\mathbf{C}, B) = P(A|\mathbf{C})$$

In other words, the conditional probability of A given $\mathbf{C}$ does not depend on B.

We are now ready to introduce the Bayesian network.

2.1.1 Definition

Bayesian networks are probabilistic graphical models that use nodes and arcs in a directed, acyclic graph to represent a joint probability distribution over a set of variables [36]. The Bayesian network is formally defined as follows.

Definition 2.1.3 (Bayesian Network)**.** *Let $P(\mathbf{X})$ be a joint probability distribution over n variables $X_1, \ldots, X_n \in \mathbf{X}$. A Bayesian network $\mathcal{B}$ is a directed, acyclic graph in which each variable X_i is represented by a node in the graph. Let $\mathbf{Pa}(X_i)$ denote the parents of node X_i in the graph. The graph representation of $\mathcal{B}$ factors the joint probability distribution as:*

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

Without any factorization, the size of the full joint probability distribution is exponential in the number of variables. Bayesian networks attempt to address this problem by offering a more compact representation of the joint probability distribution. By factoring the joint probability distribution into only the relevant variable interactions, represented by the parent-child relationships in the network, the complexity of the network can be managed. Furthermore, the Bayesian network uses the concept of conditional independence to reason about the joint probability distribution without having to represent it explicitly. The conditional independence between nodes in a network can be described in terms of their Markov blankets, defined formally as follows.

Definition 2.1.4 (Markov Blanket)**.** *The Markov blanket for a node A is the set of nodes $MB(A)$ composed of A's parents, A's children, and A's children's other parents. The Markov blanket has the property that, for any other node B,*

$$P(A|MB(A), B) = P(A|MB(A)).$$

That is, given A's Markov blanket, A is conditionally independent from all other nodes in the network.

This means that a query about a Bayesian network's joint probability distribution can be done by reasoning over smaller parts of the network at a time in order to calculate the complete answer to the query.

Example 2.1.1. *Figure 2.1 shows an example of a Bayesian network. The states T and F are shorthand for $True$ and $False$, respectively. Let each node be denoted by its first letter. The Bayesian network factors the joint probability distribution as*

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

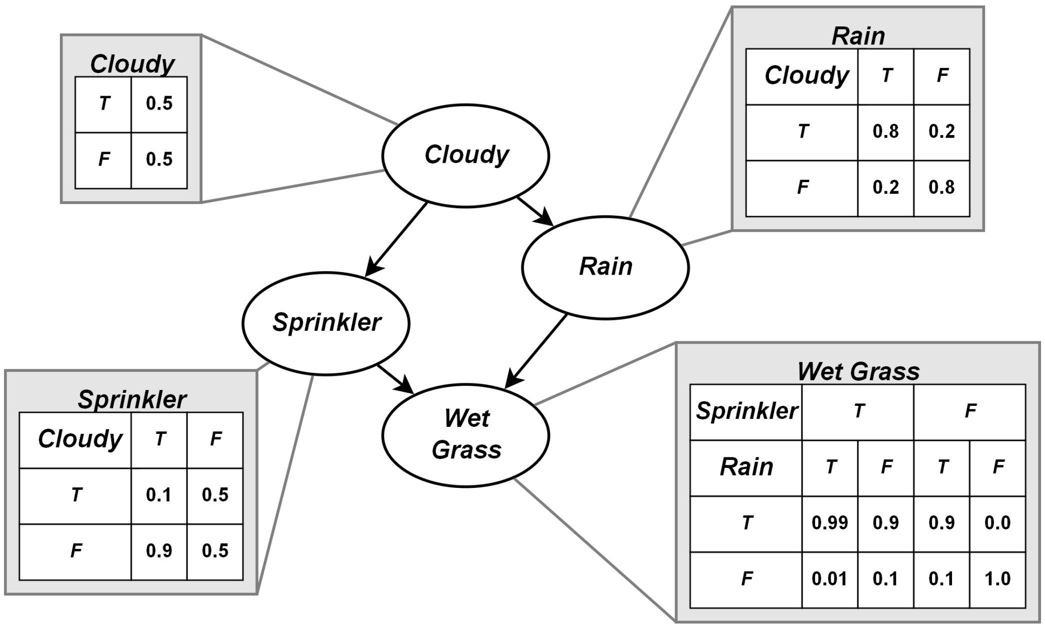


Figure 2.1: Example Bayesian network.

Using this model, we could compute the probability of wet grass given that it is cloudy and the sprinkler is off, $P(W|C = T, S = F)$*. For another example, we could compute the probability that it is cloudy and raining given that the grass is wet,* $P(C, R|W = T)$*.*

2.1.2 Dynamic Bayesian Networks

The Bayesian network defined in Section 2.1.1 is a static model. However, we can introduce the concept of time into the network by assigning discrete timesteps to the nodes to create a dynamic Bayesian network, a temporal version of a BN. The dynamic Bayesian network is defined formally as follows.

Definition 2.1.5 (Dynamic Bayesian Network)**.** *A dynamic Bayesian network (DBN) is a special type of Bayesian network that uses a series of connected timesteps, each of which contains a copy of a regular Bayesian network* $\mathbf{X}_t$ *indexed by time* t*. The probability distribution of a variable at a given timestep can be conditionally*

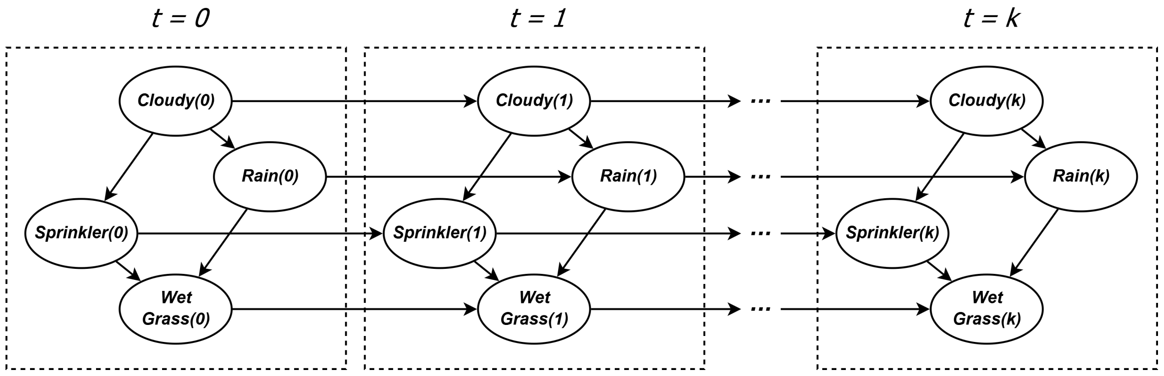


Figure 2.2: Example first-order dynamic Bayesian network

dependent on states of that variable (or even other variables) throughout any number of previous timesteps. In first-order DBNs, the nodes in each timestep are not conditionally dependent on any nodes further back than the immediately previous timestep. Therefore, the joint probability distribution for a first-order DBN of $k + 1$ *timesteps factors as:*

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

Spanning multiple timesteps, the DBN can include any evidence gathered throughout that time and use it to help reason about state probability distributions across different timesteps. Often, the conditional probability tables of the DBN are defined compactly by defining a prior network $\mathbf{X}_0$ and a single temporal network $\mathbf{X}_t$. The temporal network $\mathbf{X}_t$ is then "unrolled" to $\mathbf{X}_1, \mathbf{X}_2, \ldots, \mathbf{X}_k$ for k timesteps.

Example 2.1.2. *Figure 2.2 shows the example Bayesian network transformed into a first-order dynamic Bayesian network. Now we can compute probabilities given evidence through time. Suppose that each timestep represents one day. Now, for example, we could compute the probability of rain on day* i *given that it rained two days ago and that it was cloudy one day ago,* $P(R_i | R_{i-2} = T, C_{i-1} = T)$*. Or we could compute that probability that the grass was wet on some previous day* i *given* *that the sprinkler was not on the day before and that it was cloudy the day after,* $P(W_i|S_{i-1} = F, C_{i+1} = T)$.

The DBN model has been used in many application areas that have sequences of observations. Often these problems are concerned with performing classification only in the current timestep based on prior evidence. However, DBNs can also be unrolled further and forecast future states based on the evolving posterior probabilities. Some predictive tasks that the DBN has been used for include clinical prognostics [37, 38] and mechanical prognostics [39, 40, 41, 42].

2.2 Markov Processes

Although its name attempts to draw parallels between the conditional independence encoded by BNs, the CTBN is functionally a factored Markov process. Therefore, we present the necessary background on Markov processes.

2.2.1 Definition

There are variations and extensions of the Markov process model, but the CTBN model uses the model described here. We refer to a finite-state, continuous-time Markov chain as a Markov process. In a Markov process, a system is comprised of a discrete set of states, and the system probabilistically transitions between these states. The difference between a Markov process and a Markov chain is that each transition occurs after a real-valued, exponentially distributed sojourn time, which is the time it remains in a state before transitioning. The parameters determining the sojourn times and the transition probabilities are encoded in what is called an "intensity matrix." If the intensity matrix is constant throughout the lifetime of the system, we refer to the Markov process as "homogeneous." Formally, we define a Markov process as follows.

Definition 2.2.1 (Markov Process)**.** *A finite-state, continuous-time, homogeneous Markov process X with a state space of size $n = |X|$ is defined by an initial probability distribution P_X^0 over the n states and an $n \times n$ transition intensity matrix*

$$\mathbf{Q}_X = \begin{pmatrix} -q_{1,1} & q_{1,2} & \cdots & q_{1,n} \\ q_{2,1} & -q_{2,2} & \cdots & q_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ q_{n,1} & q_{n,2} & \cdots & -q_{n,n} \end{pmatrix}$$

in which each entry $q_{i,j} \geq 0$, $i \neq j$ gives the transition intensity of the process moving from state i to state j, and each entry $-q_{i,i} = -\sum_j q_{i,j}$ is the parameter for an exponential distribution, determining the sojourn times for the process to remain in state i.

The value $q_{i,i}$ gives the rate at which the system leaves state x_i, while the value $q_{i,j}$ gives the rate at which the system transitions from state x_i to state x_j. Let $X(t)$ denote the state of X at time t. Formally,

$$\begin{aligned} \lim_{\Delta t \to 0} P(X(t+\Delta t) = x_j | X(t) = x_i) &= \lim_{\Delta t \to 0} q_{i,j}\Delta t + O(\Delta t^2), \text{ for } i \neq j, \\ \lim_{\Delta t \to 0} P(X(t+\Delta t) = x_i | X(t) = x_i) &= \lim_{\Delta t \to 0} 1 - q_{i,i}\Delta t + O(\Delta t^2). \end{aligned}$$

With the diagonal entries constrained to be non-positive, the probability density function for the process remaining in state i is given by $|q_{i,i}| \exp(q_{i,i}t)$, with t being the amount of time spent in state i, making the probability of remaining in a state decrease exponentially with respect to time. The expected sojourn time for state i is

$1/|q_{i,i}|$. Each row is constrained to sum to zero, $\sum_j q_{i,j} = 0 \;\forall\; i$, meaning that the transition probabilities from state i can be calculated as $\theta_{i,j} = q_{i,j}/|q_{i,i}| \;\forall\; j,\; i \neq j$. Because the sojourn time uses the exponential distribution, which is "memory-less," the Markov process model exhibits the Markov property, namely, that all future states of the process are independent of all past states of the process given its present state. In other words,

$$P(X(t+\Delta t)|X(t), X(s)) = P(X(t+\Delta t)|X(t)) \text{ for } 0 < s < t < \infty.$$

Note that the model does not specify a particular time unit for the sojourn times. The modeler is responsible for choosing an appropriate time-scale (minutes, hours, days, etc.) for the specific application. The parameters are then set and interpreted accordingly.

Note also that a diagonal parameter can be set to 0, in which case the system never leaves the state once it reaches it. Such a state is called an absorbing state. If at every point in time there is a non-zero probability of reaching every state from any other state at some time in the future, the Markov process is said to be ergodic.

Example 2.2.1. *Suppose we have a Markov process B for modeling barometric pressure. The process has three states, ordered as $\{falling, steady, rising\}$, and defined with*

$$P_B^0 = \{0.3, 0.7, 0\}$$

and

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

where the unit of time is in hours.

2.2.2 Subsystems

One view of the CTBN is a set of interdependent subsystems of a Markov process. A subsystem S defines the behavior of a subset of states of full Markov process X. The intensity matrix $\mathbf{Q}_S$ of the subsystem S is formed from the entries of $\mathbf{Q}_X$ that correspond to the state of S.

In general, the intensity matrix of a subsystem will not describe a closed system, i.e., the process can transition out of the subsystem. In other words, the rows of the subsystem intensity matrix will no longer sum to zero, representing the intensity with which the process is transitioning out of the subsystem. With subsystems, we can consider entrance and exit distributions. An entrance distribution is a probability distribution over the states of S for entering the subsystem, i.e., the probability of entering the subsystem through each state. Similarly, the exit distribution is a probability distribution over the states of S for leaving the subsystem, i.e., the probability of leaving the subsystem from each state.

Example 2.2.2. *Consider the subsystem $\{steady, rising\}$ from the Markov process example for barometric pressure. The intensity matrix for the subsystem is*

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

.

2.2.3 Inference

We can reason over the states and subsystems of the Markov process. As the Markov process is a continuous-time model, we can incorporate observations in real-valued time and can query at real-valued times, rather than being restricted to pre-defined timesteps.

Inference in temporal models can be categorized as either filtering, smoothing, or prediction. Let $0 < t_1 < t < t_2$, where t denotes the current time, i.e., the time at which inference occurs. Let the set of continuous-time evidence over the interval of time $[0, t)$ be denoted as $\sigma_{[0,t)}$. Filtering is reasoning about the current probability distribution given past and possibly present observations, $P(X(t)|\sigma_{[0,t)})$. Filtering only conditions on previous evidence. Smoothing, on the other hand, is reasoning about the past probability distribution given later and possibly earlier observations, $P(X(t_1)|\sigma_{[0,t)})$. In other words, smoothing reasons backward about what likely happened in the past. Prediction is reasoning about the future given past and possibly present observations, $P(X(t_2)|\sigma_{[0,t)})$. In other words, prediction is reasoning forward about what will likely happen in the future given the past and present.

These three types of inference in a Markov process can be achieved using the following forward-backward algorithm [43]. First, consider when there is no evidence. The distribution at any point in time t can be calculated as

$$P(X(t)) = P_X^0 \exp(\boldsymbol{Q}_X t),$$

where the matrix exponential is defined by the power series

$$\exp(\boldsymbol{Q}_X t) = \sum_{n=0}^{\infty} \frac{(\boldsymbol{Q}_X t)^n}{n!}.$$

Note that while the matrix exponential is defined with this equation, there are other ways to compute it that attempt to avoid the computational and numerical difficulties [44].

When continuous-time evidence is added, the matrix exponential is partitioned, resulting in a product of matrices that alternate between matrix exponentials over the segmented intervals and transition matrices between segments. The evidence might be that the process was in a subsystem over some interval of time. Let $X([t_1, t_2))$ denote the state of X over the interval $[t_1, t_2)$. Let σ denote the evidence of the Markov process. Suppose σ is partitioned into N segments, $[t_i, t_{i+1})$, for $i = (0, N-1)$, such that the evidence is constant during each segment. Let $\boldsymbol{Q}_i$ denote the intensity matrix for segment i, meaning that the rows and columns of $\boldsymbol{Q}_i$ that do not conform to the evidence of segment i are zeroed out. Let $\boldsymbol{Q}_{i,j}$ denote the transition probabilities between segments i and j. If a transition is observed on the boundary between segments i and j, the rows and columns of $\boldsymbol{Q}_{i,j}$ are zeroed out except for the transition intensities from non-zero rows in $\boldsymbol{Q}_i$ to non-zero rows in $\boldsymbol{Q}_j$. Otherwise, segments i and j will differ only in what states are becoming observed or unobserved (instead of transitions being observed), in which case $\boldsymbol{Q}_{i,j}$ is the identity matrix. The segmentation of the matrix multiplications can then be defined recursively. Let $\boldsymbol{\alpha}_t$ and $\boldsymbol{\beta}_t$ denote the forward and backward probability vectors, defined as

$$\boldsymbol{\alpha}_t = P(X(t), \sigma_{[0,t]}),$$

$$\boldsymbol{\beta}_t = P(\sigma_{[t,T)} | X(t)).$$

Let $\boldsymbol{\alpha}_0$ be the initial distribution P_X^0 over the states of X, and let $\boldsymbol{\beta}_T$ be a vector of ones. Let $\Delta_{i,j}$ be an $n \times n$ matrix of zeros except for a one in position i, j. The recursive definitions for $\boldsymbol{\alpha}_{t_i}$ and $\boldsymbol{\beta}_{t_j}$ are

$$\begin{aligned}
\boldsymbol{\alpha}_{t_{i+1}} &= \boldsymbol{\alpha}_{t_i} \exp(\boldsymbol{Q}_i(t_{i+1} - t_i))\boldsymbol{Q}_{i,i+1}, \\
\boldsymbol{\beta}_{t_i} &= \boldsymbol{Q}_{i-1,i} \exp(\boldsymbol{Q}_i(t_{i+1} - t_i))\boldsymbol{\beta}_{t_{i+1}}.
\end{aligned}$$

The distribution over state k at time $t \in [t_i, t_{i+1})$ given evidence $\sigma_{[0,T)}$ can be computed as

$$P(X(t) = k|\sigma_{[0,T)}) = \frac{1}{Z}\boldsymbol{\alpha}_{t_i} \exp(\boldsymbol{Q}_i(t - t_i))\Delta_{k,k} \exp(\boldsymbol{Q}_i(t_{i+1} - t))\boldsymbol{\beta}_{t_{i+1}},$$

where Z is the normalizing constant.

Example 2.2.3. *Using the forward-backward algorithm, we can answer queries about the Markov process for barometric pressure. For example, we could compute the distribution at hour* 4.1*, given that the pressure has been steady from hour* 2.2 *through hour* 3.7*, as* $P(B(4.1)|B([2.2, 3.7)) = \mathit{steady})$*. For another example, we could compute the distribution at hour* 5.6*, given that at hour* 4.5 *the pressure was in the* $\{\mathit{steady}, \mathit{rising}\}$ *subsystem, as* $P(B(5.6)|B(4.5) \in \{\mathit{steady}, \mathit{rising}\})$*.*

2.3 Continuous Time Bayesian Networks

With the background in Bayesian networks and Markov processes, we are ready to define the CTBN. The CTBN was first introduced in [17] and then further developed in [18] as a continuous-time probabilistic graphical model.

2.3.1 Definition

The motivation behind CTBNs is to factor a Markov process in much the same way that a BN factors a joint probability distribution. Instead of conditional probabilities, the CTBN uses conditional Markov processes. The CTBN is defined formally as follows.

Definition 2.3.1 (Continuous Time Bayesian Network)**.** *Let* $\mathbf{X}$ *be a set of Markov processes* $\{X_1, X_2, \ldots, X_n\}$*, where each process* X_i *has a finite number of discrete states. Formally, a continuous time Bayesian network* $\mathcal{N} = \langle \mathcal{B}, \mathcal{G} \rangle$ *over* $\mathbf{X}$ *consists of two components. The first is an initial distribution denoted* $P^0_{\mathbf{X}}$ *over* $\mathbf{X}$ *specified as a Bayesian network* $\mathcal{B}$*. This distribution* $P^0_{\mathbf{X}}$ *is only used for determining the initial state of the process. The second is a continuous-time transition model* $\mathcal{G}$*, which describes the evolution of the process from its initial distribution.* $\mathcal{G}$ *is represented as a directed graph with nodes* $X_1, X_2, \ldots, X_n$*. Let* $\mathbf{Pa}(X)$ *denote the set of parents of* X *in* $\mathcal{G}$*, and let* $\mathbf{Ch}(X)$ *denote the set of children of* X *in* $\mathcal{G}$*. Let* $\mathbf{pa}_X$ *denote the set of all combinations of state instantiations to* $\mathbf{Pa}(X)$*, and let* $\langle pa_X \rangle \in \mathbf{pa}_X$*. A set of conditional intensity matrices (CIMs), denoted* $\mathbf{Q}_{X|\mathbf{Pa}(X)}$*, is associated with each* $X \in \mathbf{X}$ *and is comprised of matrices* $\mathbf{Q}_{X|\langle pa_X \rangle}$ $\forall \langle pa_X \rangle \in \mathbf{pa}_X$

Similar to a BN, the CTBN manages the complexity of the network by factoring the Markov process into interdependent subsystems. The conditional independence between nodes in a CTBN can be described in terms of Markov blankets, analogous to those in a BN. The Markov blanket for a CTBN node is still its parents, its children, and its children's other parents, but, in this case, the node is conditionally independent of all other nodes in the network when conditioned on all the states of its Markov blanket *through time*.

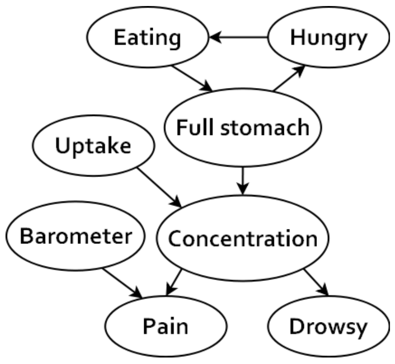


Figure 2.3: Example CTBN.

Example 2.3.1. *Figure 2.3 shows an example CTBN from [17]. The initial distributions and intensity matrices for all the nodes can be found in Appendix B. Each child node has multiple intensity matrices, one for each combination of states of its parent nodes. For example, the matrix denoted* $\mathbf{Q}_{C|u_1,f_0}$ *defines the dynamics of the node Concentration given that the state of Uptake is* u_1 *and that the state of Full stomach is* f_0*.*

This model could be used to answer several interesting queries. For example, what is the expected proportion of time that the patient is in pain while drowsy? Or, given that the patient is initially in pain but that uptake occurred at time t_1 *and that the patient finished eating at time* t_2*, what is the expected amount of time until the patient is not in pain? Or, given that the patient has been in pain from time* t_2 *to* t_3*, what is the expected number of transitions between the concentration levels that occurred during that time period?*

2.3.2 Trajectories

Let t_s and t_e be the start time and end time of an observation, respectively, such that $t_s < t_e$. Let x be a particular state of X. The tuple $\langle t_s, t_e, x\rangle$ represents an observation of the network such that $X(t) = x$ for $t_s \leq t < t_e$. The tuple could be read as "from time t_s to time t_e, the state of X was observed to be x."

Definition 2.3.2 (Trajectory)**.** *A trajectory of X, denoted $\sigma[X]$, is defined as a sequence of observations of X. If $t_s = 0$ for the first observation and if, for every pair of adjacent observations $\langle\langle t_s, t_e, x\rangle, \langle t'_s, t'_e, x'\rangle\rangle$ in $\sigma[X]$, $t_e = t'_s$, and $x \neq x'$, then $\sigma[X]$ is called a complete trajectory of X. Otherwise, $\sigma[X]$ is called a partial trajectory of X.*

A complete trajectory has no "gaps" in the observation. That is, the state of X is known from $t = 0$ until $t = t_e$ of the last observation.

Example 2.3.2. *The complete trajectory*

$$\sigma[X] = \langle\langle 0, 1.5, x_1\rangle, \langle 1.5, 2.7, x_2\rangle, \langle 2.7, 3.1, x_0\rangle\rangle$$

records that X was in state x_1 for time $t = 0$ until time $t = 1.5$. At exactly time $t = 1.5$, X transitioned to x_2 and remained there until time $t = 2.7$. At exactly time $t = 2.7$, X transitioned to x_0 and remained there until time $t = 3.1$, at which point X became unobserved.

Note that the trajectory $\sigma[X]$ is for a single node X in the network. The full set of trajectories $\sigma[X_1] \cup \sigma[X_2] \cup \cdots \cup \sigma[X_n]$ over all of the nodes of a CTBN will be denoted as σ.

Example 2.3.3. *Suppose we have a two-node CTBN, comprised of X and Y, with a dependency $X \rightarrow Y$ such that the behavior of Y changes based on the behavior of X. Furthermore, X and Y are observable (at least at certain times), which we would like to use to reason about their behavior when they are unobserved. Take the*

Algorithm 2.1 Forward sample CTBN.

```
ForwardSample($\mathcal{N}$)
1: for each $X \in \mathcal{N}$
2:     choose $X(0)$ by sampling from $\mathcal{B}$
3: end for
4: $t \leftarrow 0$, $\sigma \leftarrow \emptyset$
5: repeat until termination
6:     $Append(\sigma, \langle X, t \rangle)$
7:     for each $X \in \mathcal{N}$
8:         if $time(X) \neq null$ then continue end for
9:         $\mathbf{A}_X \leftarrow \mathbf{A}_{X|\mathbf{Pa}(X)}$
10:        $i \leftarrow X(t)$
11:        $\Delta t \sim \text{Exponential}(a_{i,i})$
12:        $time(X) \leftarrow t + \Delta t$
13:    end for
14:    $X' \leftarrow \text{argmin}_{X \in \mathcal{N}} (time(X))$
15:    $t \leftarrow time(X')$
16:    $X(t) \sim \text{Multinomial}(\mathbf{A}_{X'}, X(t))$
17:    $time(X') = null$
18:    for each $Y \in \mathbf{Ch}(X')$
19:        $time(Y) = null$
20:    end for
21: end repeat
22: return $\sigma$
```

evidence to be the partial trajectory $\mathbf{e} = \langle\langle 2.1, 3.7, y_0\rangle, \langle 2.5, 2.8, x_1\rangle\rangle$. *We might want to calculate* $P(X(t)|\mathbf{e})$ *for* $0 \leq t < 2.5$ *and* $t \geq 2.8$. *Similarly, we might want to calculate* $P(Y(t)|\mathbf{e})$ *for* $0 \leq t < 2.1$ *and* $t \geq 3.7$. *Because* X *and* Y *are connected, evidence for either node will influence the probabilities of the other.*

2.3.3 Generative Semantics

The CTBN can also be used as a generative model. Algorithm 2.1, adapted from [43], shows the pseudocode for how to create a complete trajectory from a CTBN. The algorithm accepts a CTBN from which to sample and returns a sequence of state transitions and their corresponding transition times. Lines 1-4 choose the starting states of the trajectory and initialize the variables. Lines 5-21 are repeated until the trajectory is of desired length, which could be the total time of the trajectory or the total number of transitions. In line 6, the method $\mathit{Append}(\sigma, \langle X, t\rangle)$ adds a transition, given as a state X at time t to the end of the trajectory σ. Lines 7-13 ensure that all variables have a proposed sojourn time, drawn from an exponential distribution whose parameter depends on the current states of the parents of each node. Line 14 selects the node with the soonest proposed transition time, and line 15 updates the current time of the trajectory. Line 16 chooses the next state for the transitioning node, according to a multinomial distribution whose parameters are derived from the current row of the node's conditional intensity matrix. Finally, lines 17-20 reset the proposed sojourn times for that node and all its children to be re-sampled. The proposed sojourn times of the node's children are reset because their conditional intensity matrices changed when their parent transitioned to a new state.

2.3.4 Amalgamation

While we have shown that the CTBN is able to represent a Markov process as a set of interdependent subsystems, it is also useful to show how the subsystems of a CTBN can be merged together into "supernodes" containing the dynamics of multiple subsystems.

First, we introduce additional notation for specific state instantiations and sets of state instantiations. Let $\langle pa_{X\setminus Y}\rangle$ denote the state instantiation $\langle pa_X\rangle$ excluding any state of Y (this changes $\langle pa_X\rangle$ only if Y is a parent of X). Then $\mathbf{Q}_{X|\langle pa_{X\setminus Y}\rangle,Y}$ is the set of conditional intensity matrices that are dependent on the state instantiation

$\langle pa_{X\backslash Y}\rangle$ and each state of Y,

$$\mathbf{Q}_{X|\langle pa_{X\backslash Y}\rangle,Y} = \{\mathbf{Q}_{X|\langle pa_{X\backslash Y}\rangle,y} | y \in Y\}.$$

Finally, let $\mathbf{pa}_{X\backslash Y}$ denote the set of all combinations of state instantiations to $\mathbf{Pa}(X)$ excluding any state of Y (again, this changes $\mathbf{pa}_X$ only if Y is a parent of X).

The process involves combining sets of conditional intensity matrices from two different nodes, $\mathbf{Q}_{X|\langle pa_{X\backslash Y}\rangle,Y}$ and $\mathbf{Q}_{Y|\langle pa_{Y\backslash X}\rangle,X}$, and forming a new conditional intensity matrix $\mathbf{Q}_{XY|\langle pa_{XY}\rangle}$, where $\langle pa_{XY}\rangle = \langle pa_{X\backslash Y}\rangle \cup \langle pa_{Y\backslash X}\rangle$. That is, the state instantiations for the parents of X and Y are combined, excluding states of X and Y. The states of X and Y are excluded from $\langle pa_{XY}\rangle$ because $\mathbf{Q}_{XY|\langle pa_{XY}\rangle}$ will be defined over all state combinations of X and Y.

Example 2.3.4. *Suppose we have a CTBN with $A \rightarrow X \leftrightarrows Y \leftarrow B$. Combining the conditional intensity matrix sets $\mathbf{Q}_{X|a_0,Y}$ and $\mathbf{Q}_{Y|b_1,X}$ will create conditional intensity matrix $\mathbf{Q}_{XY|a_0,b_1}$.*

Let $x_{i,j}$ be entry i, j of $\mathbf{Q}_{X|\langle pa_X\rangle,y_k}$, and let $y_{k,l}$ be entry k, l of $\mathbf{Q}_{Y|\langle pa_Y\rangle,x_i}$. The combined CIM $\mathbf{Q}_{XY|\langle pa_{XY}\rangle}$ is the matrix defined over the states (x_i, y_k), with the entries populated as follows.

$$(x_{ij}, y_{kl}) = \begin{cases} x_{ij} & \text{if } i \neq j \text{ and } k = l \\ y_{kl} & \text{if } i = j \text{ and } k \neq l \\ x_{ij} + y_{kl} & \text{if } i = j \text{ and } k = l \\ 0 & \text{otherwise} \end{cases} \tag{2.1}$$

The CIM $\mathbf{Q}_{XY|\langle pa_{XY}\rangle}$ defines the simultaneous dynamics of X and Y, given that their parents are in states $\langle pa_{XY}\rangle$. Thus, the state-space of XY is the Cartesian product of the states of X and Y, making $\mathbf{Q}_{XY|\langle pa_{XY}\rangle}$ an $|X||Y| \times |X||Y|$ matrix.

Example 2.3.5. *Assume that we have the CTBN from Example 2.3.4 and that the four conditional intensity matrices are given as follow.*

$$\mathbf{Q}_{X|a_0,y_0} = \begin{array}{c|cc} & x_0 & x_1 \\ \hline x_0 & -1 & 1 \\ x_1 & 2 & -2 \end{array} \qquad \mathbf{Q}_{Y|b_1,x_0} = \begin{array}{c|cc} & y_0 & y_1 \\ \hline y_0 & -3 & 3 \\ y_1 & 4 & -4 \end{array}$$

$$\mathbf{Q}_{X|a_0,y_1} = \begin{array}{c|cc} & x_0 & x_1 \\ \hline x_0 & -5 & 5 \\ x_1 & 6 & -6 \end{array} \qquad \mathbf{Q}_{Y|b_1,x_1} = \begin{array}{c|cc} & y_0 & y_1 \\ \hline y_0 & -7 & 7 \\ y_1 & 8 & -8 \end{array}$$

Combining these intensity matrices, we have

$$\mathbf{Q}_{XY|a_0,b_1} = \begin{array}{c|cccc} & (x_0,y_0) & (x_0,y_1) & (x_1,y_0) & (x_1,y_1) \\ \hline (x_0,y_0) & -4 & 3 & 1 & 0 \\ (x_0,y_1) & 4 & -9 & 0 & 5 \\ (x_1,y_0) & 2 & 0 & -9 & 7 \\ (x_1,y_1) & 0 & 6 & 8 & -14 \end{array}$$

Definition 2.3.3 (Amalgamation)**.** *Amalgamation takes two nodes X and Y and replaces them with node XY, having the set of conditional intensity matrices $\mathbf{Q}_{XY|\mathbf{Pa}(XY)}$ as formed by combining $\mathbf{Q}_{X|\langle pa_{X\setminus Y}\rangle,Y}$ and $\mathbf{Q}_{Y|\langle pa_{Y\setminus X}\rangle,X}$ $\forall\langle pa_{Y\setminus X}\rangle \in \mathbf{pa}_{X\setminus Y}$ and $\forall\langle pa_{X\setminus Y}\rangle \in \mathbf{pa}_{Y\setminus X}$ according to Equation 6.7. Amalgamation can be viewed as*

Algorithm 2.2 Amalgamate two nodes of a CTBN.

```
$Amalgamate(X, Y)$
 1: $\mathbf{Q}_{XY|\mathbf{Pa}(XY)} \leftarrow \emptyset$
 2: for each $\langle pa_{Y \setminus X} \rangle \in \mathbf{pa}_{X \setminus Y}$ and $\langle pa_{X \setminus Y} \rangle \in \mathbf{pa}_{Y \setminus X}$
 3:     $\mathbf{Q}^{XY} \leftarrow \mathbf{0}$
 4:     for $i, j = 1, \ldots, |X|$ and $l, k = 1, \ldots, |Y|$
 5:         $\mathbf{Q}^{X} \leftarrow \mathbf{Q}_{X|\langle pa_{X \setminus Y} \rangle, x_i}$
 6:         $\mathbf{Q}^{Y} \leftarrow \mathbf{Q}_{Y|\langle pa_{Y \setminus X} \rangle, y_k}$
 7:         if $i = j \wedge k = l$
 8:             $q^{XY}_{(i,j),(k,l)} \leftarrow q^{X}_{i,j} + q^{Y}_{k,l}$
 9:         else if $i = j \wedge k \neq l$
10:             $q^{XY}_{(i,j),(k,l)} \leftarrow q^{Y}_{k,l}$
11:         else if $i \neq j \wedge k = l$
12:             $q^{XY}_{(i,j),(k,l)} \leftarrow q^{X}_{i,j}$
13:         end if
14:     end for
15:     $\mathbf{Q}_{XY|\langle pa_{XY} \rangle} \leftarrow \mathbf{Q}^{XY}$
16:     $\mathbf{Q}_{XY|\mathbf{Pa}(XY)} \leftarrow \mathbf{Q}_{XY|\mathbf{Pa}(XY)} \cup \{\mathbf{Q}_{XY|\langle pa_{XY} \rangle}\}$
17: end for
18: return $\mathbf{Q}_{XY|\mathbf{Pa}(XY)}$
```

a multiplication operation over sets of conditional intensity matrices and is denoted $\mathbf{Q}_{XY|\mathbf{Pa}(XY)} = \mathbf{Q}_{X|\mathbf{Pa}(X)} \times \mathbf{Q}_{Y|\mathbf{Pa}(Y)}$.

Amalgamation takes two nodes and combines all of their CIMs, producing a set of CIMs that are conditioned on the combined parent states of X and Y. Thus, the set $\mathbf{Q}_{XY|\mathbf{Pa}(XY)}$ contains $\prod_{Z \in \mathbf{Pa}(XY)} |Z|$ conditional intensity matrices.

Algorithm 2.2 shows the pseudocode for amalgamating two nodes of a CTBN. Line 1 initializes the empty set of conditional intensity matrices for the amalgamated node. Lines 2-17 iterate over all combinations of parent state instantiations of X and Y, excluding the state of X and Y. Line 3 initializes the conditional intensity matrix to be populated. Lines 4-14 iterate over the state combinations of X and Y. Lines 5-6 assign the conditional intensity matrices to temporary variables for simpler notation. Lines 7-13 populate the parameters of the conditional intensity matrix initialized in line 3 per Equation 2.2. Lines 15-16 add the conditional intensity matrix to the set of conditional intensity matrices of the amalgamated node, which is returned in Line 18.

Example 2.3.6. *Assume that we have the CTBN from Example 2.3.4 and that we amalgamate X and Y. This turns the CTBN into $A \to XY \leftarrow B$.*

Definition 2.3.4 (Full Joint Intensity Matrix)**.** *The full joint intensity matrix of a CTBN is the matrix resulting from amalgamating all nodes of the CTBN,*

$$\mathbf{Q} = \prod_{X \in \mathcal{N}} \mathbf{Q}_{X|\mathbf{Pa}(X)}$$

The size of $\mathbf{Q}$ is $n \times n$, where $n = \prod_{X \in \mathcal{N}} |X|$.

2.3.5 DBNs vs. CTBNs

Note that, unlike a regular Bayesian network, cycles are allowed in $\mathcal{G}$, because the nodes represent subsystems of a single Markov process, and therefore only one state transition is allowed at a time for the whole CTBN. A cycle in a CTBN model would be analogous to a dynamic Bayesian network with variables X and Y, where arcs such as $X_t \to Y_{t+1}$ and $Y_t \to X_{t+1}$ would be valid.

Similar to the Bayesian network, the conditional dependencies allow a more compact representation for the model. Like a Bayesian network, the local conditionally dependent probability tables can be combined to form the full joint probability distribution. In the case of the CTBN, this is the full joint intensity matrix, which describes the evolution of the entire process. However, just as in the Bayesian network, in which the number of entries in the full joint probability distribution grows exponentially in the number of variables, so too the number of states in the full joint intensity matrix grows exponentially in the number of variables for the CTBN.

Despite these few similarities, the CTBN model is fundamentally different from the DBN model. Although the network topologies for both models encode conditional dependence, the models are differentiated by what the nodes represent. Whereas the nodes in a DBN are simple random variables, the nodes in a CTBN are conditional Markov processes. As a result, CTBNs can be queried about the state probabilities for any real-valued time. A DBN, unrolled for a discrete number of timesteps, can only be queried for state probabilities at these timesteps but not in-between adjacent timesteps. While the time interval between timesteps can be set with finer granularity, doing so multiplies the number of nodes needed to span the same amount of time as the original unrolled DBN and still is not continuous with respect to time. DBNs becomes asymptotically equivalent to a CTBN only as the interval of time between timesteps approaches zero [45].

2.3.6 Inference and Learning

The only exact inference algorithm that exists so far for CTBNs simply expands and works with the full joint intensity matrix, which is exponential in the number of nodes [46]. However, this inference algorithm does not take advantage of the factored nature of the network. Thus, research has focused on approximate methods.

The CTBN importance sampling algorithm [46, 43] is a sample-based inference algorithm. The algorithm uses a combination of exponential and truncated exponential distributions to select node sojourn times that conform to upcoming evidence, after which transitions are sampled from multinomial distributions. The sampler generates a sequence of sojourn times and transitions for all nodes to generate a complete trajectory. Because the trajectory was sampled from a distribution that assumed the evidence, the trajectory is weighted by its likelihood. A set of samples is generated, and queries are answered by calculating the weighted proportion of samples that match the query.

Using importance sampling, each interval of evidence causes a decreased weight in a fraction of the samples, resulting in an increase in variance of the importance weights. Because the weight of a complete trajectory corresponds to the product of its interval weights, the decreased weight of each interval approaching non-matching evidence produces a distribution of trajectory weights with potentially high variance, which can indicate that the sampled distribution is far from the true distribution and can lead to biased results unless the number of samples is large enough. To address this, the rejection-sampler framework of [47] attempts to improve the sampled distribution by learning when to accept or reject samples from that distribution that forces the evidence.

The Gibbs sampling algorithm for CTBNs is another sample-based inference algorithm [48]. Gibbs sampling takes a Markov Chain Monte Carlo (MCMC) approach. The algorithm starts by generating an arbitrary trajectory that is consistent with the evidence. The trajectory for a single node's dynamics depends on trajectories of its Markov blanket, as well as on the node's past and present evidence. Therefore, the algorithm alternates between randomly picking a node and sampling a trajectory from the distribution of the node, conditioned on the trajectories of the other components and the evidence. The algorithm samples a complete, continuous trajectory for that node in each iteration. After repeating this process over all the nodes, the idea is that the sampled trajectories will converge to the true distribution. The computational complexity of this algorithm is determined by the complexity of the current trajectories and the sampled one, rather than a pre-defined time granularity parameter (analogous to timesteps in a DBN). Thus, the computation time of the algorithm adapts to the complexity of the network and the complexity of the evidence.

The auxiliary variable Gibbs sampler for CTBNs [49] is based on the idea of uniformization, in which a set of "virtual" sojourn times are sampled from a Poisson process. The algorithm then sets up a Markov chain over paths of a node by alternately sampling the set of virtual sojourn times given the current path and then sampling a new path given the set of extant and virtual sojourn times using a standard hidden Markov model forward-filtering, backward-sampling algorithm. The auxiliary variable Gibbs sampler is computationally efficient, as compared to the original Gibbs sampling algorithm for CTBNs [48].

More recently, the Metropolis-Hastings (MCMC) algorithm [50] was developed for detecting hidden variables in a CTBN. Like the auxiliary variable Gibbs sampler, the Metropolis-Hastings algorithm exploits uniformization. The algorithm was found to be more efficient than the importance sampling algorithm, as well as the auxiliary Gibbs sampling algorithm when performing inference over larger state-spaces.

Methods for expectation propagation (EP) [51] have also been developed, which employ a message passing scheme in cluster graphs. In these methods, for each interval of evidence, the nodes exchange messages with their neighbors in the cluster. The idea is to pass approximate "marginals" which are unnormalized, unconditional intensity matrices, valid for that interval, until all of the nodes have a consistent distribution over that interval. Each cluster in the cluster graph encodes a distribution over trajectories of the nodes in the cluster throughout the duration of evidence. Therefore, expectation propagation, unlike discrete-time models, can adapt the time granularity at which it reasons for different nodes, for different segments, and under different conditions. In other words, computational resources can be allocated to different clusters based on how quickly the cluster is evolving and the desired accuracy in each cluster. Originally, this allocation would have to be specified manually based on the network and the evidence being applied. The extension to EP developed in

[52] provides an information-theoretic criterion to automatically and dynamically re-partition the clusters during the inference process, allowing EP to adapt the level of approximation to each cluster based on its current rate of evolution. Their algorithm avoids the need to manually select the appropriate inference granularity, allowing the granularity to adapt throughout the inference process.

The mean-field variational approximation method for CTBNs [45, 53] is another message passing algorithm. In this case, it uses products of inhomogeneous Markov processes to approximate a distribution over trajectories as systems of ordinary differential equations (ODEs). Using asynchronous updates, the variational approach is guaranteed to converge, and the approximation represents a consistent joint distribution over the nodes of the network which can then be efficiently queried. The approach is able to exploit the development of optimized ODE solvers, which automatically tune the trade-off between the time granularity and the approximation quality. As expected with a variational approach, the method is susceptible to local maxima, and it cannot capture certain complex interactions in the posterior distribution. By using a time-inhomogeneous representation, the approximation is able to capture many complex patterns in the evolution of the marginal distribution of each node.

Belief propagation (BP) [54] is similar to the mean-field variational approach, except that it only needs to converge to locally consistent distributions over neighborhoods of nodes rather than globally consistent distributions over all of the nodes. In this sense, BP is similar to the EP algorithm for CTBNs. The main difference is in the structure of the approximation. EP maintains the homogeneity assumption of each node by representing the dynamics of a node as piece-wise conditional intensity matrices. BP, on the other hand, represents the inhomogeneity through systems of ODEs.

Approaches specific to continuous-time particle filtering in CTBNs [55] have also been developed. These methods propagate probabilities forward in time, without having to be conditioned on upcoming evidence. They look at two different methods for computing the matrix exponential: ODE integration and uniformization of the Taylor expansion. To counteract the exponential size of the full joint intensity matrix, they consider approximations in which only a factored belief state is maintained. For factored uniformization, they demonstrate that the KL-divergence of the filtering is bounded.

The particle filtering algorithm of [56] applies to hybrid systems containing both discrete-state and continuous-state nodes. The dynamics of the discrete-state nodes are determined by a Markov process. Whenever the discrete-state process is observed, the filtering algorithm samples a trajectory of the underlying Markov process. This trajectory is then used to estimate the continuous-state nodes using the system dynamics determined by the discrete state in the trajectory.

As a data-driven model, algorithms have been developed for model learning, both with learning the network structure and model parameters. First, the learning algorithm of [57] addresses the problem of learning parameters and structure of a CTBN from fully observed data. They define a conjugate prior for CTBNs, showing how it can be used for both Bayesian parameter estimation and as the basis of a Bayesian score for comparing different network structures for use in structure learning algorithms.

Later, the work of [29] extended this to the problem of learning the parameters and structure of a CTBN from partially observed data. They showed how to apply expectation maximization (EM) and structural expectation maximization (SEM) to CTBNs. The introduced of the EM algorithm allowed CTBNs to learn more complex sojourn distributions than just a single exponential distribution. This extension addressed one of the limitations of CTBNs, namely the restriction of states to exponentially distributed sojourn times.

CTBN model maturation has also been explored. The algorithm of [58] updates parameters of an existing CTBN model with a new set of data samples. They present a framework for online parameter estimation and batch parameter updating for CTBNs. They derive parameter update rules from their framework, which attempts to balance the existing model and new data samples when calculating the updated parameters.

2.3.7 Applications and Extensions

CTBNs have found use in several applications. For example, CTBNs have been used for inferring users' presence, activity, and availability over time [59]; robot monitoring [56]; modeling server farm failures [60]; modeling social network dynamics [61, 62]; modeling sensor networks [63]; building intrusion detection systems [64, 65, 66]; predicting the trajectory of moving objects [67]; and diagnosing cardiogenic heart failure and anticipating its likely evolution [68, 69]. The CTBN has also been extended to support decision-making, resulting in structured continuous-time Markov decision processes [70].

The CTBN model has also undergone several specializations and generalizations. The Generalized CTBN (GCTBN) of [71] combines the conditional probability tables of BNs and the conditional intensity matrices CTBNs, allowing nodes to be either what they call "delayed" nodes or "immediate" nodes. They show how inference can be performed when combining the conditional probabilities of the immediate nodes with the intensity matrices of the delayed nodes by converting to a generalized stochastic Petri net (GSPN) which defines immediate and delayed transition types between states. The delayed nodes are exponentially distributed transitions in the GSPN, while the immediate nodes are immediate transitions in the GSPN. The GSPN model provides well-defined semantics for GCTBNs in terms of the underlying stochastic process, and it provides a means to perform inference (both prediction and smoothing) on GCTBNs through evaluating the associated GSPN.

The CTBN classifier (CTBNC) of [72, 73, 74] is similar to the GCTBN in that it combines conditional Markov processes with a static probability distribution. In this case, the CTBNC features a parent-less class node with a marginal probability distribution over the class label for classifying a static object given continuous-time evidence about that object. In the GCTBN, only the single class node uses a static probability distribution, while all the other nodes are CTBN nodes. The conditional intensity matrices are conditioned on the class label, which does not change in time. Given a continuous-time trajectory of the object as evidence, the model classifies the object with the class having the highest posterior probability.

The work of [75] changes the representation of the CTBN to be partition-based, using what they call conditional intensity trees and conditional intensity forests. This is analogous to representing the conditional probabilities in BNs as decision trees [76]. Because the number of parameters of a node is exponential in the number of parents, the idea is to more compactly represent the parameters encoding the parameters in trees in which duplicates can be removed. For example, if a node had only two distinct conditional intensity matrices, these unique matrices would take up only two leaves of the tree. The branches of the tree would dictate which matrix to return based on the current states of the parents.

The Erlang-Coxian CTBN (EC-CTBN) of [77] replaces the exponential distribution of the sojourn times with Erlang-Coxian distributions. The Erlang-Coxian distributions allow the model to approximate non-exponential sojourn time distributions. In [18], they had shown how combinations of nodes in the CTBN could represent

Erlang-Coxian distributions [29], but the EC-CTBN replaces this with a single node by introducing a generalized conditional intensity matrix. In addition, they describe an expectation-maximization algorithm for learning these non-exponential sojourn distributions.

The asynchronous dynamic Bayesian network (ADBN) of [78] uses a CTBN for the representation of the system but a DBN for inference over the system. At query time, the conditional intensity matrices of the CTBN are converted to conditional probability tables, creating a temporary DBN. The idea is that the parameters of the DBN will be populated on-the-fly from the conditional intensity matrices and the continuous-time evidence of the CTBN. This attempts to avoid the assumption of a uniform interval of time between the timesteps of the DBN (hence, an asynchronous DBN). After converting the conditional intensity matrices to conditional probability tables, inference can be performed over the DBN instead of the CTBN. The ADBN attempts to leverage the existing work on inference for BNs.

CHAPTER 3

COMPLEXITY OF INFERENCE

Before we introduce extensions to inference in CTBNs, we consider the difficulty of inference in CTBNs in general. Because the CTBN is relatively new, much of the complexity theory surrounding CTBNs has yet to be fully explored. The CTBN reduces the number of model parameters by using a factored representation. Even still, the number of conditional intensity matrices of a node is exponential in the number of the node's parents. If we constrained the maximum number of parents, does the factorization of the CTBN guarantee that inference is tractable? Also consider the quality of the answer returned by a CTBN inference algorithm. If computing the exact answer to a query is difficult in the general case, can we expect to more easily compute an approximate solution?

In this chapter, we address these questions by proving three new theorems specific to inference in CTBNs, that both exact and approximate (absolute and relative) inference in CTBNs is NP-hard. We start by reviewing the current progress in CTBN complexity theory and providing background on the complexity theory of BNs. We then prove the three theorems and provide empirical validation of the performance predicted by the result. We conclude with a discussion of the implications of these theorems and pose questions for future work on CTBN complexity theory.

3.1 Background Work

In this section, we review the complexity results proved for CTBNs so far and review some of the complexity theory of BNs relevant to our proofs.

3.1.1 CTBN Complexity Theory

Most of the complexity theory surrounding CTBNs is derived from the use of a BN for the initial distribution. One complexity result specific to CTBNs arises from the difference between BN and CTBN structure learning. In structure learning, it is common to assign a scoring function to arcs in the network that quantifies how well the network topology matches the training data. The learning algorithm of [18] gives a polynomial-time algorithm for finding the highest-scoring set of k parents for a CTBN node when given complete data (no missing values). The corresponding problem in a BN has been shown to be NP-hard, even for $k = 2$, due to the acyclic constraint of BNs [79]. The learning algorithm can maximize the score of each node independently, because it does not need to try all combinations of network structures that enforce the acyclic constraint while maximizing the score of a single node. We note that, while lifting the acyclic constraint makes structure learning easier, cycles can introduce difficulties for inference, because the complete behavior of a node depends on all of its ancestors, which becomes the entire cycle at the very least. Aside from this complexity result for structure learning, we found no other results specific to CTBN complexity theory in either learning or inference. Thus, our proofs concerning inference are novel contributions to the understanding of complexity in CTBNs.

3.1.2 Exact Inference in BNs

Our results build on the complexity results of BNs, which we review in this section.

Theorem 3.1.1. *(Cooper) Exact inference in Bayesian networks is NP-complete [80].*

Proof. Cooper proved the NP-completeness of BN inference via a reduction from 3SAT. The 3SAT problem consists of a set of m clauses $C = \{c_1, c_2, \ldots, c_m\}$ made up of a finite set V of n Boolean variables. Each clause contains a disjunction of three literals over V, for example, $c_3 = (v_2 \wedge \neg v_3 \wedge v_4)$. The 3SAT problem is determining whether there exists a truth assignment for V such that all the clauses in C are satisfied.

The 3SAT problem can be reduced to a BN decision problem of whether, for a $True(T)/False(F)$ node X in the network, $P(X = T) > 0$ or $P(X = T) = 0$. We can represent any 3SAT instance by a BN as follows. For each Boolean variable $v_i \in V$, we add a corresponding $True(T)/False(F)$ node V_i to the network such that $P(V_i = T) = \frac{1}{2}$ and $P(V_i = F) = \frac{1}{2}$. For each clause C_j, we add a corresponding $True(T)/False(F)$ node C_j to the network as a child of the three nodes corresponding to its three Boolean variables. Let w_j be the actual clause corresponding to the state of the three parents of C_j, and let $eval(w_j)$ be the truth function for this clause. The conditional probabilities of the node are

$$P(C_j = T|w_j) = \begin{cases} 1 & \text{if } eval(w_j) = T \\ 0 & \text{if } eval(w_j) = F \end{cases}$$

Finally, for each clause C_j, we add a $True(T)/False(F)$ node D_j. Each D_j is conditionally dependent on C_j and on D_{j-1} (except for D_1). The conditional probabilities for D_1 are

$$P(D_1 = T|C_1) = \begin{cases} 1 & \text{if } C_1 = T \\ 0 & \text{otherwise} \end{cases}.$$

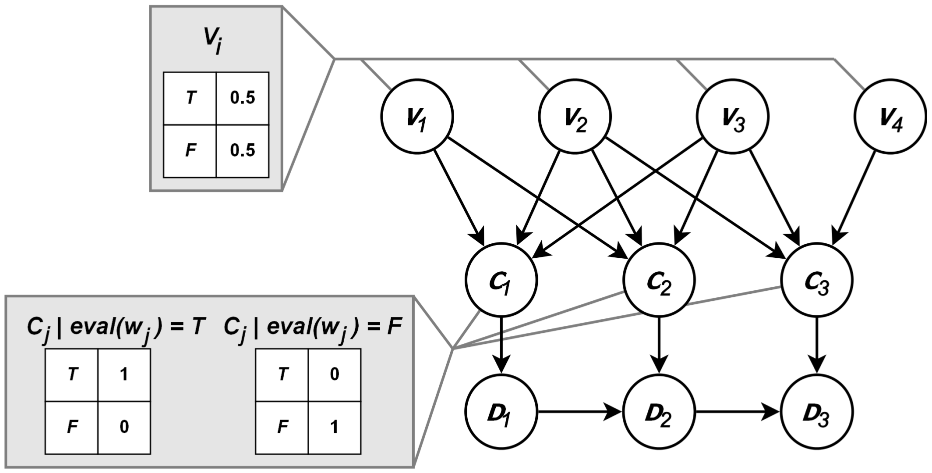


Figure 3.1: Example reduction from 3SAT to BN exact inference.

Similarly, the conditional probabilities for D_j $(j > 1)$ are

$$P(D_j = T|C_j, D_{j-1}) = \begin{cases} 1 & \text{if } C_j = T \wedge D_{j-1} = T \\ 0 & \text{otherwise} \end{cases} .$$

Example 3.1.2. *Figure 3.1 shows the BN for determining the satisfiability of the clause* $(v_1 \vee v_2 \vee v_3) \wedge (\neg v_1 \vee \neg v_2 \vee v_3) \wedge (v_2 \vee \neg v_3 \vee v_4)$.

Importantly, the construction of this network is polynomial in the length of the Boolean expression. For a 3SAT instance of $|V|$ variables and $|C|$ clauses, the corresponding network has $|V| + 2|C|$ nodes. Furthermore, each node of the network has no more than three parents, constraining the largest conditional probability table to have no more than 16 entries, for a maximum of $2|V| + 16|C| + 8(|C| - 1) + 4$ entries for the entire network.

The probabilities of the V nodes allow for every combination of truth assignments to the Boolean variables. From there, the C and D nodes enforce the logical relations of the clauses using the network's conditional probability tables. As such, the 3SAT instance is satisfiable if and only if $P(D_m = T) > 0$, that is, if and only if there is a non-zero probability that some instantiation of the V nodes to T and F will cause all of the clauses to be satisfied. Thus, if an algorithm exists that is able to efficiently compute the exact probabilities in arbitrary BNs, the algorithm can efficiently decide whether $P(D_m = T) > 0$ for the specially constructed networks that can represent arbitrary instances of 3SAT. □

3.1.3 Approximate Inference in BNs

Furthermore, it is known that even absolute and relative approximations in BNs are NP-hard [81]. These approximations are defined formally as follows. Suppose we have a real value $\epsilon \in [0, 1]$, a BN with binary-valued nodes V, and two nodes $X, Y \in V$ instantiated to x and y, respectively.

Definition 3.1.1. *A relative approximation is an estimate* $0 \leq Z \leq 1$ *such that*

$$\frac{P(X = x|Y = y)}{(1 + \epsilon)} \leq Z \leq P(X = x|Y = y)(1 + \epsilon).$$

Definition 3.1.2. *An absolute approximation is an estimate* $0 \leq Z \leq 1$ *such that*

$$P(X = x|Y = y) - \epsilon \leq Z \leq P(X = x|Y = y) + \epsilon.$$

The proof of NP-hardness for relative approximation follows from the proof for exact inference. Satisfiability of the clause is determined whether $Z = 0$ or $Z > 0$, which is not influenced by the choice of ϵ. Thus, there is no constant-factor relative approximation for inference in BNs.

Theorem 3.1.3. *(Dagum & Luby) Absolute approximate inference in Bayesian networks is NP-hard [81].*

The proof of NP-hardness for absolute approximation starts with the reduction for exact inference as above, representing the variables and clauses with the same network and parameters. This time, one by one a truth assignment is set for each Boolean variable v_i, and the corresponding node V_i is removed from the network. The truth assignment for v_i is determined by the higher probability of $P(V_i = T | D_m = T)$ and $P(V_i = F | D_m = T)$. However, if there exists an efficient approximate BN inference algorithm that can guarantee to be within $\epsilon = \frac{1}{2}$ of the exact probability on arbitrary BNs, this algorithm can be used to determine efficiently satisfying truth assignments to all Boolean variables of an arbitrary instance of 3SAT. Furthermore, any approximation with $\epsilon \geq \frac{1}{2}$ for a two-state node (the simplest case) is no better than random guessing.

These results are for BNs, which apply to the initial distribution of a CTBN. While the CTBN and DBN are formulated differently, a DBN becomes asymptotically equivalent to a CTBN as the interval of time between timesteps approaches zero [45]. One might be tempted to argue that the BN complexity proofs therefore apply to the CTBN. However, it is not always clear that moving from a discrete space to a continuous space preserves the complexity results. For example, take the difference between linear programming and integer linear programming, the former being solvable in polynomial time with the latter being NP-hard. Thus, we prove the complexity results for CTBNs explicitly.

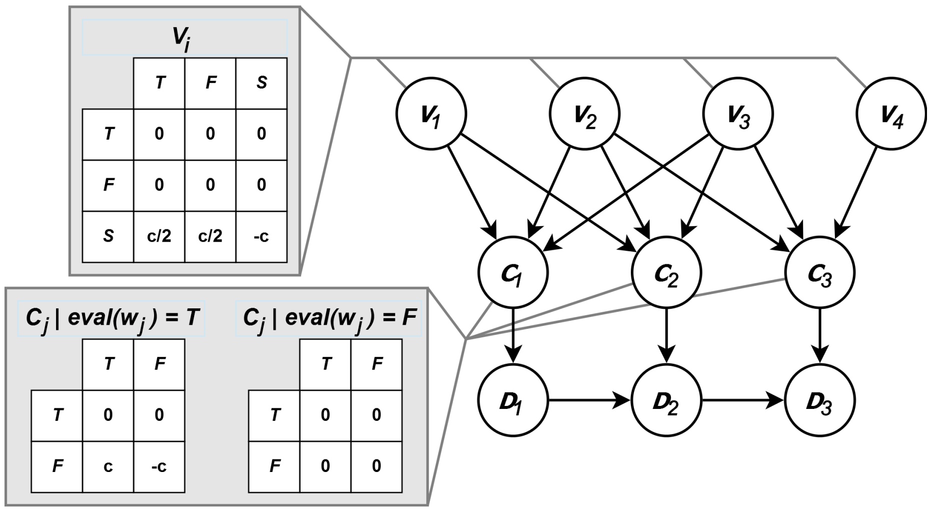


Figure 3.2: Example reduction from 3SAT to CTBN exact inference.

3.2 Complexity of Inference in CTBNs

3.2.1 Exact Inference

We show that exact inference in CTBNs is NP-hard, even when given the exact initial states, following a similar reduction as the proof for BNs but using the conditional intensity matrices of the CTBN instead of the conditional probability tables.

Example 3.2.1. *Figure 3.2 shows the CTBN for determining the satisfiability of the clause* $(v_1 \vee v_2 \vee v_3) \wedge (\neg v_1 \vee \neg v_2 \vee v_3) \wedge (v_2 \vee \neg v_3 \vee v_4)$.

Theorem 3.2.2. *Exact inference in continuous time Bayesian networks is NP-hard.*

Proof. The CTBN topology matches that of the BN for representing variables and clauses, but the nodes are specified differently. For each Boolean variable $v_i \in V$, we add a corresponding three-state node V_i to the network. The three states in order are $True(T)$, $False(F)$, and $Start(S)$, the last being the initial state for node V_i. We set the unconditional intensity matrix of V_i to be

$$\mathbf{Q}_{V_i} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ c/2 & c/2 & -c \end{pmatrix}$$

for any constant $c > 0$. The zero entries for T and F make them absorbing states. At some point in time, the system will transition to either T or F with equal probability, but not both.

For each clause C_j, we add a corresponding $True(T)/False(F)$ node C_j to the network as a child of the three nodes corresponding to its three Boolean variables. As before, let w_j be the clause corresponding to the state of the three parents of C_j, and let $eval(w_j)$ be the truth function for this clause. The function $eval$ is extended to return $False$ whenever the clause w_j contains a node in state S. The conditional intensity matrices of C_j are

$$\mathbf{Q}_{C_j|eval(w_j)=T} = \begin{pmatrix} 0 & 0 \\ c & -c \end{pmatrix}$$

and

$$\mathbf{Q}_{C_j|eval(w_j)=F} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

We set the initial state of each C_j to be the F state (the second row of the matrices). These conditional intensity matrices mean that there is a non-zero probability of C_j transitioning from F to T if and only if the states of the parents of C_j correspond to a satisfying truth assignment to the jth clause.

Finally, for each clause C_j, we add a $True(T)/False(F)$ node D_j. Each D_j is conditionally dependent on C_j and on D_{j-1} (except for D_1). The conditional intensity matrices for D_j are

$$\mathbf{Q}_{D_j|eval(C_j \wedge D_{j-1})=T} = \begin{pmatrix} 0 & 0 \\ c & -c \end{pmatrix}$$

and

$$\mathbf{Q}_{D_j|eval(C_j \wedge D_{j-1})=F} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}.$$

As with the C_j nodes, we set the initial state of each D_j to be the F state. As with the clause nodes, these conditional intensity matrices mean that there is a non-zero probability of D_j transitioning from F to T if and only if C_j and D_{j-1} are T.

The conditional intensity matrices of the CTBN enforce the logical constraints of the Boolean expression, replacing the conditional probability tables of the network. As before, a 3SAT instance of $|V|$ variables and $|C|$ clauses generate $|V| + 2|C|$ nodes in the corresponding CTBN. Each node still has no more than three parents but now each intensity matrix has 9 or 4 entries, meaning that there is a maximum of $9|V| + 108|C| + 16(|C| - 1) + 8$ conditional intensity matrix entries for the entire network. Thus the construction of the CTBN is polynomial in the length of the Boolean expression. Notice also that we have constrained the maximum number of parents in the CTBN to less than or equal to three.

Let $D_m(t)$ be the state of D_m at time t. The 3SAT instance is satisfiable if and only if $P(D_m(t) = T) > 0 \; \forall \; t > 0$. Assume that the Boolean expression is satisfiable by some combination of T/F state assignments to the variables in V. The V_i nodes start in the S state at time $t = 0$. The time that each variables remains in S is exponentially distributed, after which the variables transition to either T or F with equal probability and remain in that state. Therefore, there is a non-zero probability for each combination of T/F states in the V_i's at $t > 0$. Whenever three of these states satisfy a clause C_j, there is a non-zero probability for C_j to transition from F to T when $t > 0$. Likewise, once the parents of D_j are in T there is a non-zero probability for D_j to transition from F to T when $t > 0$. Thus, if the Boolean expression is satisfiable, there is a non-zero probability that each each clause is satisfied at $t > 0$, and therefore $P(D_m(t) = T) > 0$. On the other hand, assume that the Boolean expression is not satisfiable. Then there exists some clause C_j that remains in F for all $t > 0$. Therefore, D_j will remain in F for all $k \geq j$, which means that $P(D_m(t) = T) = 0$ for all $t \geq 0$. □

3.2.2 Approximate Inference

We prove similar results for approximate inference with CTBNs as well.

Theorem 3.2.3. *Relative approximate inference in continuous time Bayesian networks is NP-hard.*

Proof. Because the determination is whether $P(D_m(t) = T) = 0$ or $P(D_m(t) = T) > 0$, a relative approximation for $P(D_m(t) = T)$ with any error bound also gives a solution to the satisfiability of the Boolean expression. □

We now turn to the absolute approximation. Because even approximate inference in BNs is NP-hard, it seems reasonable to suspect that a similar conclusion also holds for approximate inference in CTBNs. We now show how an absolute error approximation algorithm for CTBNs can be used to find a satisfying assignment to the Boolean expression or to determine that it is not satisfiable.

Theorem 3.2.4. *Absolute approximate inference in continuous time Bayesian networks is NP-hard.*

Proof. We start by assuming that the expression has at least one satisfying assignment. A satisfying truth assignment can be found one variable at a time by choosing $t > 0$ and conditioning on $D_m(t) = T$. Whereas the proof for exact inference needed only to compute $P(D_m(t) = T)$ to solve the 3SAT instance, the proof for absolute approximate inference needs to condition on $D_m(t) = T$ to solve the 3SAT instance. Let $t' \geq t$.

By construction, $P(V_i(t') = S|D_m(t) = T) = 0$. This is important, because it ensures that

$$P(V_i(t') = T|D_m(t) = T) + P(V_i(t') = F|D_m(t) = T) = 1.$$

Let $a \in \{T, F\}$, and let $\widehat{P}_a^i$ denote the absolute error approximation with ϵ for the probability $P(V_i(t') = a|D_m(t) = T)$. Without loss of generality, assume that V_i can be satisfied only when $a = T$. Then $P(V_i(t') = T|D_m(t) = T) = 1$ and $P(V_i(t') = F|D_m(t) = T) = 0$. Therefore, it must be that $\widehat{P}_T^i > \widehat{P}_F^i$ whenever $\epsilon < \frac{1}{2}$. We compute both $\widehat{P}_T^i$ and $\widehat{P}_F^i$ and change the initial state of V_i to T if $\widehat{P}_T^i > \widehat{P}_F^i$ and to F otherwise. This process continues for $i = 1, \ldots, |V|$ to determine truth assignments for all variables in the Boolean expression. Therefore, if there exists a polynomial-time approximation algorithm for CTBN inference with $\epsilon < \frac{1}{2}$ that can condition on evidence, it can be used to solve arbitrary instances of 3SAT in polynomial time as well. $\square$

3.3 Empirical Validation

We can empirically validate these theoretical results by taking Boolean expressions and performing inference in the corresponding CTBN. Specifically, we demonstrate three Boolean expressions, listed as follows.

$$\begin{aligned} BE1 &= (v_1 \vee v_2 \vee v_3) \wedge (\neg v_1 \vee \neg v_2 \vee v_3) \wedge (v_2 \vee \neg v_3 \vee v_4) \\ BE2 &= (v_1 \vee v_1 \vee v_1) \wedge (\neg v_2 \vee \neg v_2 \vee \neg v_2) \wedge (v_3 \vee v_3 \vee v_3) \\ BE3 &= (v_1 \vee v_1 \vee v_2) \wedge (v_1 \vee v_1 \vee \neg v_2) \wedge (\neg v_1 \vee \neg v_1 \vee v_2) \wedge (\neg v_1 \vee \neg v_1 \vee \neg v_2) \end{aligned}$$

Note that $BE1$ is the Boolean expression given as an example earlier and with the CTBN shown in Figure 3.2. A total of 10 out of its 16 possible truth assignments satisfy the expression. Note that $BE2$ has a single satisfying assignment and that $BE3$ is unsatisfiable.

To determine the satisfiability of each of these expressions using the corresponding CTBN, we performed forward sampling (Algorithm 2.1) with 100,000 samples and $c = 100$ over the interval time $[0, 0.2)$. We queried the proportion of samples with which $D_m(t) = T$ for $t = 0$ to $t = 0.2$ in increments of 0.01. The results are shown in Figure 3.3. For the two satisfiable expressions, $BE1$ and $BE2$, $P(D_m(t) = T) > 0$ for $t \geq 0.01$, while for the unsatisfiable query $BE3$, $P(D_m(t) = T) = 0$ for all $t \in [0, 0.2)$.

Also note the values to which the probabilities are converging. For $BE1$, the probability ended at an estimated 0.622, whereas the proportion of satisfying assignments is $10/16 = 0.625$. For $BE2$, the probability ended at an estimated 0.127, whereas the proportion of satisfying assignments is $1/8 = 0.125$. As the number of samples increased, the probabilities converged to the proportion of satisfying assignments.

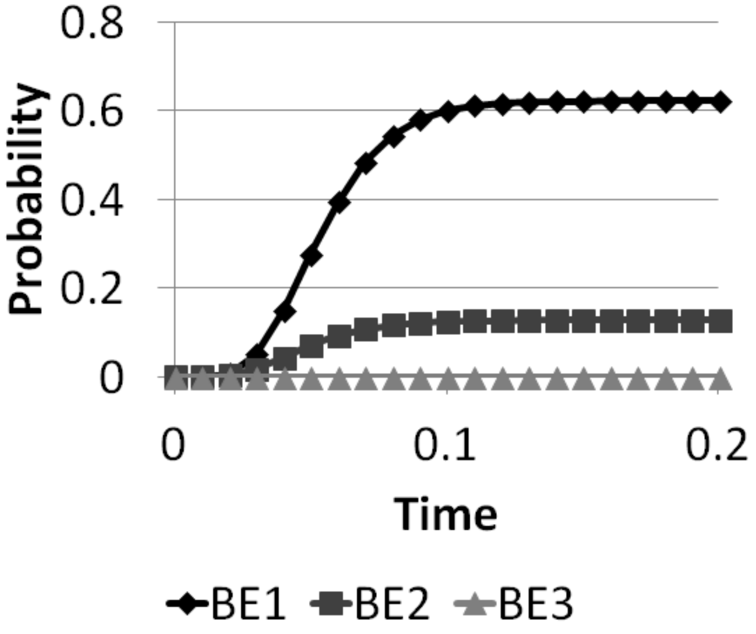


Figure 3.3: Empirical satisfiability results for the three Boolean expressions.

Table 3.1: Empirical results for approximating $P(V_i(0.2) = T|D_m(0.2) = T)$.

	V_1	V_2	V_3	V_4
$BE1$	≈ 0.50	≈ 0.60	≈ 0.60	≈ 0.60
$BE2$	1	0	1	-
$BE3$	NaN	NaN	-	-

Next, we validated the approach by which an approximation of $P(V_i(t) = T|D_m(t) = T)$ is able to determine a satisfying assignment to each V_i. We used importance sampling [46] because, unlike forward sampling, it is able to condition on evidence by weighting the samples with the likelihood of the evidence. To prevent division by zero when calculating the weights, we smooth the zero entries in the unconditional intensity matrices with $\pm 10^{-6}$. We ignored samples with infinitesimal weights, as an infinitesimal weight implied that the corresponding sample contained a transition that violates the Boolean expression. The results with 100,000 samples are shown in Table 3.1.

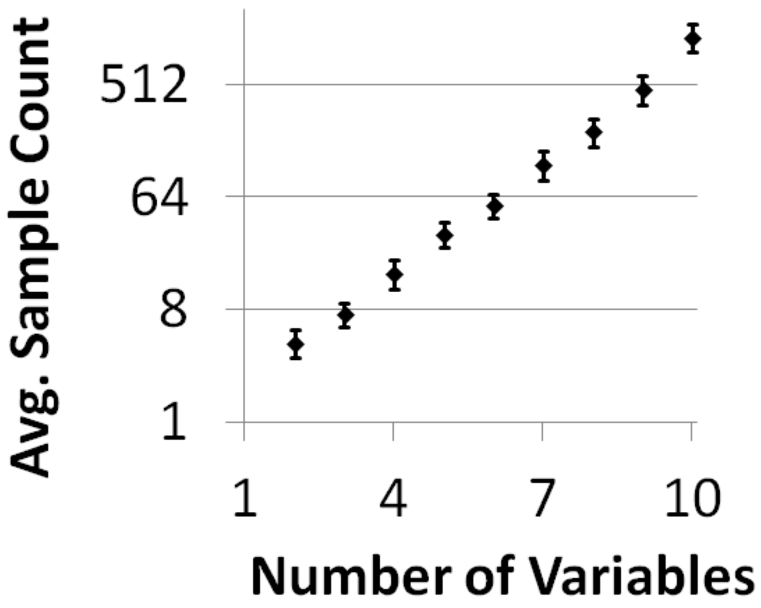


Figure 3.4: Sample complexity for CTBN inference.

The table shows that the importance sampling algorithm was correctly able to determine a satisfying truth assignments to each variable or determine that no truth assignments was possible. For $BE1$, by setting v_2, v_3, and v_4 to T, the Boolean expression is satisfied regardless of the value of v_1, which is why the estimate was approximately 0.5, that is, either T or F is equally probably for satisfying the expression. For $BE2$, the importance sampling algorithm determined the single satisfying truth assignment. For $BE3$, no feasible samples could be generated because it is conditioned on an impossible event $D_m(0.2) = T$, indicating that the expression is unsatisfiable.

While we showed that we are able to solve these instances of 3SAT by CTBN sampling methods, the complexity is still exponential in the length of the Boolean expression. To demonstrate this, we show the average sample count necessary to determine the satisfiability of the Boolean expression

$$\bigwedge_{i=1,\ldots,n} (v_i \vee v_i \vee v_i)$$

for $n = 2, \ldots 9$. Each expression has exactly one truth assignment that satisfies it (all variables set to $True$). We count the number of samples generated until we have the first sample for which $D_m(0.2) = T$, making $P(D_m(0.2) = T) > 0$ and thus showing that the 3SAT instance is satisfiable. For each number of variables, we averaged the number of samples generated over 100 runs. The average sample counts along with confidence intervals for $\alpha = 0.01$ are plotted in Figure 3.4. The $\log_2$ scale on the y-axis shows that the algorithm is exponential in the length of the expression.

3.4 Conclusion

We have shown that exact and approximate inference in CTBNs is NP-hard, even when given the initial distribution. Thus, the difficulty of CTBN inference is found not only in BN inference for calculating this initial distribution, but also in accurately propagating the probabilities forward in time. Given the similar results with BNs, these results are not surprising. However, as with BNs, further research may reveal special cases of the CTBN, whether in their structures or their parameters, which admit polynomial-time algorithms for approximate or even exact inference. We have shown that constraining the maximum number of parents to less than or equal to three is insufficient to guarantee tractable inference.

Proving the complexity of exact and approximate inference in CTBNs under different structural conditions may provide further insights into the complexity of working with these models and possibly suggest ways to modelers that the complexity of specific models can be better managed. It may serve as a step toward special-case CTBNs that can be proved to be tractable. Our results show that merely placing constraints on the maximum number of parents is insufficient to guarantee the feasibility of exact or approximate inference.

Even though exact and approximate inference in BNs is known to be NP-hard in the general case, practical algorithms still exist that are able to perform inference efficiently for many networks. Similarly, inference in CTBNs is an active and fruitful area of research. In the subsequent chapters, we focus on extending the CTBN model itself, as well as extending both the types of inference and the capabilities of inference.

CHAPTER 4

PERFORMANCE FUNCTIONS

In this chapter, we extend the CTBN model to include performance functions and show how an existing CTBN inference algorithm can be used to reason over these performance functions. The original CTBN definition allows users to specify the dynamics of how the system evolves, but users might also want to place value expressions over the dynamics of the model. We extend the CTBN to allow for this by formalizing performance functions for the CTBN. We show how the performance functions can be factored in the same way as the network, allowing what we argue is an intuitive and explicit representation. For cases in which a performance function must involve multiple nodes, we show how to augment the structure of the CTBN to account for the performance interaction while maintaining the factorization of a single performance function for each node.

4.1 Background Work

Instead of reasoning over the behavior of the network, performance functions allow the user to reason over user-defined performance measures over the network. This can be accomplished by importance sampling for CTBNs, which we briefly review. We then review the concept of using conditional intensity matrices as logical expressions, which we use to enable factored performance functions even when the performance is defined in terms of complex state interactions between multiple nodes.

4.1.1 Importance Sampling

We briefly review the CTBN importance sampling algorithm, which serves as our underlying inference method once we introduce our factored performance functions. While we discuss an extended version of importance sampling in detail in Chapter 7, in this chapter, we simply use the existing importance sampling algorithm to demonstrate inference over our CTBN performance functions.

Importance sampling is an example of a particle-based method. These methods use the model to generate a set of particles. Statistics are taken over the set particles to approximate the desired distribution. Particle-based methods can also condition on evidence by constraining and/or weighting the particles in various ways.

Importance sampling is able to estimate $E(f|\mathbf{e})$ for arbitrary functions f defined over complete trajectories of the system. It takes a partial trajectory $\mathbf{e}$ as evidence and samples a proposal distribution P' that conforms to the evidence to fill in the unobserved intervals to generate a complete trajectory σ. Because the sampler draws from a different distribution P' to force the sampling to conform to the evidence, each sample is weighted by the likelihood of the evidence to account for sampling from P' instead of P. This is calculated as

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

with the cumulative weight as

$$W = \sum_{\sigma \in \mathcal{S}} w(\sigma).$$

From a set of i.i.d. samples $\mathcal{S}$, we can approximate the conditional expectation of a function f given the evidence $\mathbf{e}$ as:

$$\hat{E}(f|\mathbf{e}) = \frac{1}{W}\sum_{\sigma\in\mathcal{S}} w(\sigma)f(\sigma)$$

4.1.2 Conditional Intensity Matrices as Logical Expressions

In Chapter 3, we used the CIMs of the C and D nodes to enforce the logical constraints of the Boolean expression. However, by the construction of the network, the clause nodes could only transition in one direction, from $False$ to $True$, if and only if the corresponding clause was satisfied. The use of CIMs as logical expressions were also used in [33], but in this case the nodes are used as temporal logic AND and OR gates. That is, the node transitions between $True$ and $False$ at various times, based on the states of the parents.

Suppose the $True(T)/False(F)$ node X is an AND node. Then the CIMs of X are given as follows.

$$\boldsymbol{Q}_{X|(\bigwedge_{x\in\langle pa_X\rangle} x)=T} = \begin{pmatrix} 0 & 0 \\ \infty & -\infty \end{pmatrix} \quad \boldsymbol{Q}_{X|(\bigwedge_{x\in\langle pa_X\rangle} x)=F} = \begin{pmatrix} -\infty & \infty \\ 0 & 0 \end{pmatrix}$$

These matrices perform conjunction over the states of the node's parents. Similarly, suppose the $True(T)/False(F)$ node X is an OR node. Then the CIMs of X are given as follows.

$$\boldsymbol{Q}_{X|(\bigvee_{x\in\langle pa_X\rangle} x)=T} = \begin{pmatrix} 0 & 0 \\ \infty & -\infty \end{pmatrix} \quad \boldsymbol{Q}_{X|(\bigvee_{x\in\langle pa_X\rangle} x)=F} = \begin{pmatrix} -\infty & \infty \\ 0 & 0 \end{pmatrix}$$

These matrices perform disjunction over the states of the node's parents. In practice, ∞ is simulated by a sufficiently large value that allows near-instantaneous transitions, as compared to the other transitions times.

Instead of just AND and OR nodes, we note that the CIMs can be defined uniquely for each combination of the parent states. Thus, augmenting the CTBN structure with these nodes can be used for complex performance interactions between subsystems.

4.2 Inference over Performance Functions

Sometimes the user may not be interested in querying $P(\mathbf{X}(t)|\mathbf{e})$ specifically, i.e., the expected behavior of the network given a partial trajectory as evidence $\mathbf{e}$. Instead, the user may place different values on particular behaviors of the network and want to calculate the expected value of a given instantiation of the system. In other words, the user has a function $f : \sigma \to \mathbb{R}$ defined over the behavior the network and wants to compute the expected value of f given the evidence, $E(f|\mathbf{e})$. This query is different from (although related to) the calculation of $P(\mathbf{X}(t)|\mathbf{e})$. While $\mathcal{G}$ may show X and Y to be independent in their *behavior*, there may be a dependence between X and Y in f because of how the user *values* their mutual behavior. Whereas the CTBN allows us to factor a complex system $\mathbf{X}$ into interdependent subsystems to tractably estimate $P(\mathbf{X}(t)|\mathbf{e})$, we would like to factor f into a set of functions such that we can also tractably estimate $E(f|\mathbf{e})$.

4.2.1 Importance Sampling with Performance Functions

Note that in the importance sampling algorithm, the entire sample σ is passed to $f(\sigma)$. While this allows f to be fully general and discern arbitrary behaviors of the system, this also means that the evaluation of f must be specially implemented for each function and for each network. Because the state-space of σ is exponential in the number of variables, simply enumerating f over all the states of the network is infeasible. A representation such as a lookup table over even just the relevant states of the network would also be difficult for a user to define by hand and subsequently difficult for others to interpret. We would like to find a way to factor f to make it more manageable and understandable while retaining as much of its expressive power as possible. To do this, we move into the novel contributions of this chapter, introducing performance functions local to each node and showing how to incorporate dependence in the performance functions by augmenting the structure of the CTBN with what we call *synergy nodes.*

We factor f according to the nodes in the network, assigning each node X its own performance function f_X. Originally, f is also defined over all the transitions of a single node. While this may be useful in some cases, the value we place on certain states of a node are not be dependent on the state of the node several states ago. Therefore, we also factor each performance function with respect to time. Instead of defining $f_X(\sigma)$ over a full sample σ, we define $f_X(t_s, t_e, x)$ to be over the observations $\langle t_s, t_e, X(t_s)\rangle \in \sigma[X]$, which are the transitions of only node X in the trajectory σ. The performance of the entire network can now be factored as

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

The factored performance function f_X is able to represent such things as fixed and variable costs over the states of X. For example, consider a performance function for some node X with states x_0 and x_1. Let $\Delta t = t_e - t_s$. Then suppose

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

in which c_1 and c_2 are two constants representing dollars and dollars per hour, respectively, and the time is in hours. This performance function means that the system incurs a fixed cost of c_1 every time it enters state x_0 and accrues a variable cost of c_2 for every hour it remains in state x_0.

Each performance function is now responsible for calculating the performance of a single node in the network. The factorization of f also allows for greater flexibility in generating the set of samples $\mathcal{D}$. If the performance function f is defined (non-zero) for only a subset of nodes $\mathbf{X}' \subset \mathbf{X}$, the sampling algorithm only needs to record

$$\bigcup_{X \in \mathbf{X}'} \sigma[X]$$

for each sample σ, instead of every transition of every node in $\mathbf{X}$.

We can further generalize performance functions by noting that a network is not restricted to a single performance function f. We could define an entire family of performance functions $\mathcal{F} = \{f^1, f^2, \ldots, f^m\}$ for a single CTBN. Each performance function gives one "view" of the network. For example, we could define $\mathcal{F}$ to represent competing metrics, such as quantity vs. quality, and measure the trade-offs incurred by the current instance of the system. Moreover, we can evaluate $\mathcal{F}$ with a single set of samples $\mathcal{S}$, regardless of the size of $\mathcal{F}$.

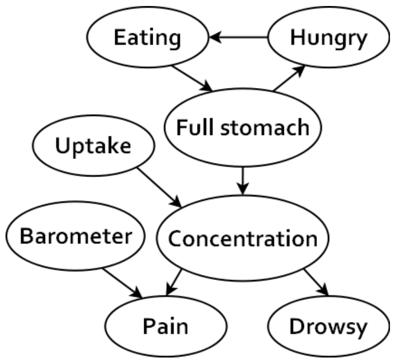


Figure 4.1: Drug effect network.

4.2.2 Synergy Nodes

The factorization of f into a single f_X for each node of the CTBN could be too restrictive for encoding the desired performance function. We now show how to augment the structure of the CTBN to make the performance functions more expressive while still preserving the factorization of f onto single nodes. First, we show how the performance function could be too restricted. Suppose that the performance functions for Pain (P) and Drowsy (D) from Figure 4.1 are defined as:

$$f_P(t_s, t_e, P(t_s)) = \begin{cases} 2\Delta t & \text{if } P(t_s) = \textit{pain-free} \\ 0 & \text{if } P(t_s) = \textit{in-pain} \end{cases},$$

$$f_D(t_s, t_e, D(t_s)) = \begin{cases} \Delta t & \text{if } D(t_s) = \textit{non-drowsy} \\ 0 & \text{if } D(t_s) = \textit{drowsy} \end{cases}.$$

In other words, we have $f = 3\Delta t$ when the Pain and Drowsy nodes are in states *pain-free* and *non-drowsy* simultaneously. But suppose a user values being non-drowsy and pain-free at the same time twice as much as the sum of the values of being non-drowsy and pain-free separately, and the user wants the following performance function defined over P and D together to be:

$$f_{\{P,D\}}(t_s, t_e, \{P(t_s), D(t_s)\}) = \begin{cases} 6\Delta t & \text{if } P(t_s) = \textit{pain-free} \wedge D(t_s) = \textit{non-drowsy} \\ 2\Delta t & \text{if } P(t_s) = \textit{pain-free} \wedge D(t_s) = \textit{drowsy} \\ \Delta t & \text{if } P(t_s) = \textit{in-pain} \wedge D(t_s) = \textit{non-drowsy} \\ 0 & \text{if } P(t_s) = \textit{in-pain} \wedge D(t_s) = \textit{drowsy} \end{cases}$$

In this case, $f = 6\Delta t$ instead of $3\Delta t$ when *pain-free* and *non-drowsy.* The performance function $f_{P \cup D}$ does not factor into f_P and f_D as before. This introduces the concept of synergy between nodes. Formally, we define synergy as follows.

Definition 4.2.1 (Synergy). *Nodes $X_1, \ldots, X_k$ exhibit synergy if their joint performance function $f_{\{X_1,\ldots,X_k\}}$ cannot be factored into $f_{X_1}, \ldots, f_{X_k}$ such that, for all $x_1 \in X_1, \ldots, x_k \in X_k$,*

$$f_{\{X_1,\ldots,X_k\}}(\{x_1, \ldots, x_k\}) = f_{X_1}(x_1) + \cdots + f_{X_k}(x_k).$$

Suppose, on the other hand, that $f_{\{X_1,\ldots,X_k\}}$ is able to be factored partially into $f_{X_1}, \ldots, f_{X_k}$ such that the above equality holds for at least one state in each node $x_1 \in X_1, \ldots, x_k \in X_k$. Then all other states $x'_1 \in X_1, \ldots, x'_k \in X_k$ for which the equality does not hold exhibit either *positive synergy* or *negative synergy.*

Definition 4.2.2 (Positive Synergy). *States $x_1 \in X_1, \ldots, x_k \in X_k$ exhibit positive synergy if and only if*

$$f_{\{X_1,\ldots,X_k\}}(\{x_1, \ldots, x_k\}) > f_{X_1}(x_1) + \cdots + f_{X_k}(x_k).$$

Definition 4.2.3 (Negative Synergy)**.** *States $x_1 \in X_1, \ldots, x_k \in X_k$ exhibit negative synergy if and only if*

$$f_{\{X_1,\ldots,X_k\}}(\{x_1,\ldots,x_k\}) < f_{X_1}(x_1) + \cdots + f_{X_k}(x_k).$$

Synergy implies that the performance of multiple nodes is dependent. Synergy occurs at the state level, and nodes can exhibit both positive and negative synergy at the different times. To account for synergy in the performance functions while maintaining the factorization of f, we add what we call synergy nodes into the network. A synergy node is set as the child of all nodes contributing to the synergy. Suppose that synergy node X_S is added as for synergy between states $x_1, \ldots, x_k$. Let the states of X_S be ordered as $\{\textit{inactive}, \textit{active}\}$. Then the conditional intensity matrices of X_S are specified as follows.

$$\mathbf{Q}_{X_S}|\langle pa_{X_S}\rangle = \begin{cases} \begin{pmatrix} -\infty & \infty \\ 0 & 0 \end{pmatrix} & \text{if } \langle pa_{X_S}\rangle = \{x_1,\ldots,x_k\} \\ \begin{pmatrix} 0 & 0 \\ \infty & -\infty \end{pmatrix} & \text{otherwise} \end{cases}$$

The performance function f_{X_S} is then specified as follows.

$$f_{X_S}(x_S) = \begin{cases} 0 & \text{if } x_S = \textit{inactive} \\ f_{\{X_1,\ldots,X_k\}}(\{x_1,\ldots,x_k\}) & \text{if } x_S = \textit{active} \end{cases}$$

Thus, the synergy node captures any additional performance between nodes that each node's factored performance function cannot represent by itself.

Table 4.1: Conditional intensity matrices of PD^+.

$A_{PD^+ \mid P(t_s)=\textit{pain-free}, D(t_s)=\textit{non-drowsy}}$		
	inactive	*active*
inactive	$-\infty$	∞
active	0	0

$A_{PD^+ \mid P(t_s)=\textit{pain-free}, D(t_s)=\textit{drowsy}}$ $A_{PD^+ \mid P(t_s)=\textit{in-pain}, D(t_s)=\textit{non-drowsy}}$ $A_{PD^+ \mid P(t_s)=\textit{in-pain}, D(t_s)=\textit{drowsy}}$		
	inactive	*active*
inactive	0	0
active	∞	$-\infty$

For the P and D synergy example, we add a synergy node PD^+ (denoting positive synergy) as a child of P and D. The augmented section of the network is shown in Figure 4.2, and the conditional intensity matrices for PD^+ are given in Table 4.1. The combination of ∞ and 0 forces the synergy node to transition immediately to the active (inactive) state and remain there for as long as the logical expression is satisfied (unsatisfied). Finally, we set the performance function of PD^+ as

$$f_{PD^+}(t_s, t_e, PD^+(t_s)) = \begin{cases} 3\Delta t & \text{if } PD^+(t_s) = \textit{active} \\ 0 & \text{if } PD^+(t_s) = \textit{inactive} \end{cases}$$

The CIMs of PD^+ act as a logical expression over the states of the parents P and D. Whenever P is *pain-free* and D is *non-drowsy*, the synergy node PD^+ immediately switches to *active* and yields an additional $3\Delta t$ in performance. This yields the desired performance function with the factorization as $f = f_P + f_D + f_{PD^+}$. Thus, f is still factored onto individual nodes. Furthermore, the synergy node PD^+ provides a graphical representation of the performance function. We can see at a glance in Figure 4.2 that the performance functions of P and D are dependent. Furthermore,

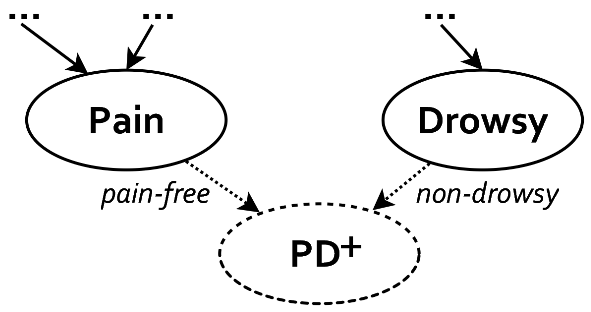


Figure 4.2: Positive synergy node for *pain-free* and *non-drowsy.*

we can define the synergy between *pain-free* and *non-drowsy* as its own value, instead of having to define it in a combined function $f_{\{P,D\}}$.

4.3 Experiments

We demonstrate the use of the synergy node concept on a reliability model adapted from [33] that describes the uptime of a vehicle system. The model, shown in Figure 4.3, consists of three 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). The initial distributions and intensity matrices for all the nodes can be found in Appendix C.

For our experiments, we have a fleet of vehicles. Each vehicle model can incorporate its own evidence, e.g., repair and usage history. We want to calculate a synergistic measure of performance across the entire fleet, represented in the node V^+. We compare the use of the synergy node with the brute force approach, i.e., of evaluating a performance function defined over all of the *Vehicle* nodes at once.

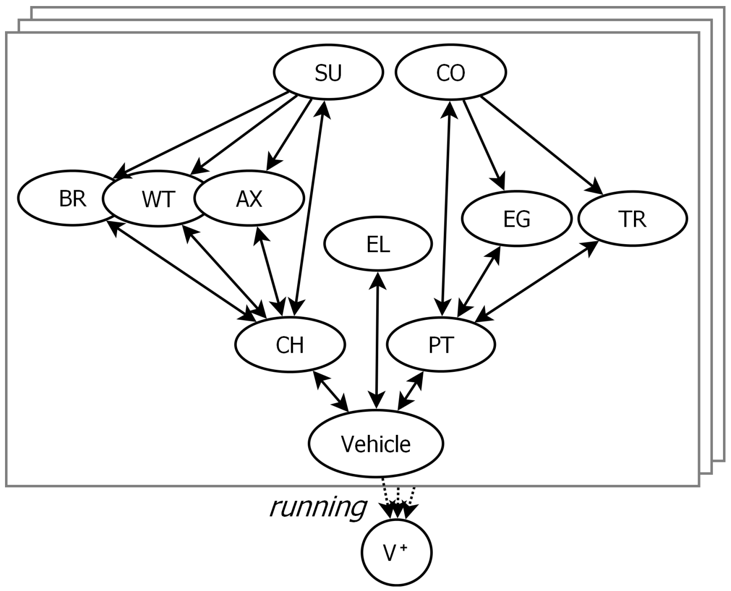


Figure 4.3: CTBN for fleet of vehicles with synergy node.

Therefore, the network for the brute force approach does not include the V^+ node. For the synergy node approach, the V^+ node becomes *active* when all vehicles are running, otherwise it remains *inactive*. Suppose that the performance function for V^+ is defined as

$$f_{V^+}(t_s, t_e, V^+(t_s)) = \begin{cases} \Delta t & \text{if } V^+(t_s) = \textit{active} \wedge \Delta t \geq 40 \\ 0 & \text{if otherwise} \end{cases} .$$

In other words, additional performance is gained when all of the vehicles are running simultaneously for at least 40 hours. The performance gained is proportional to the amount of time that all of the vehicles are running until the next repair.

We varied the fleet size from 2 to 16 vehicles and queried the expected performance over 2000 hours of operation starting with all vehicles in running condition. We simulated ∞ in the CIMs of V^+ as 10^{10}. We used importance sampling to generate 10,000 samples for each fleet size and for the brute force and synergy node approaches. We

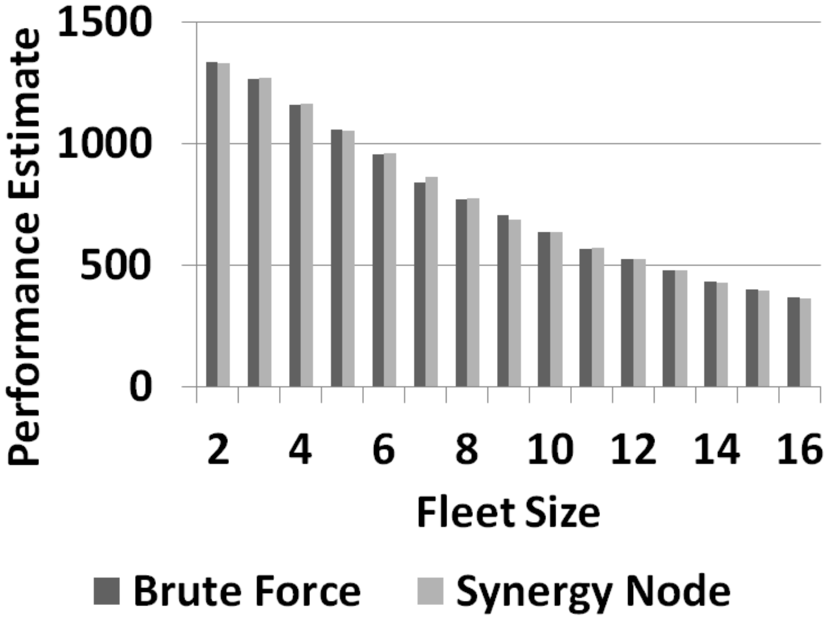


Figure 4.4: Comparison of performance estimates of the brute force and synergy node approaches.

compared accuracy of the performance estimate, average number of transitions, and average number of times the sampler must draw from an exponential or multinomial distribution. Because we only needed to save the trajectories for the *Vehicle* nodes, the average number of transitions per sample dictates how many times the performance function must be evaluated. Because each sample is a series of sojourn times and transitions, the number of times the sampler draws from an exponential or multinomial distribution is the driving factor in the complexity of creating the samples.

The performance estimates of the brute force approach and the synergy node approach is shown in Figure 4.4. As the graph shows, the synergy node is able to return an estimate consistent with the brute force approach. The average relative error between the two approaches is less than 1%.

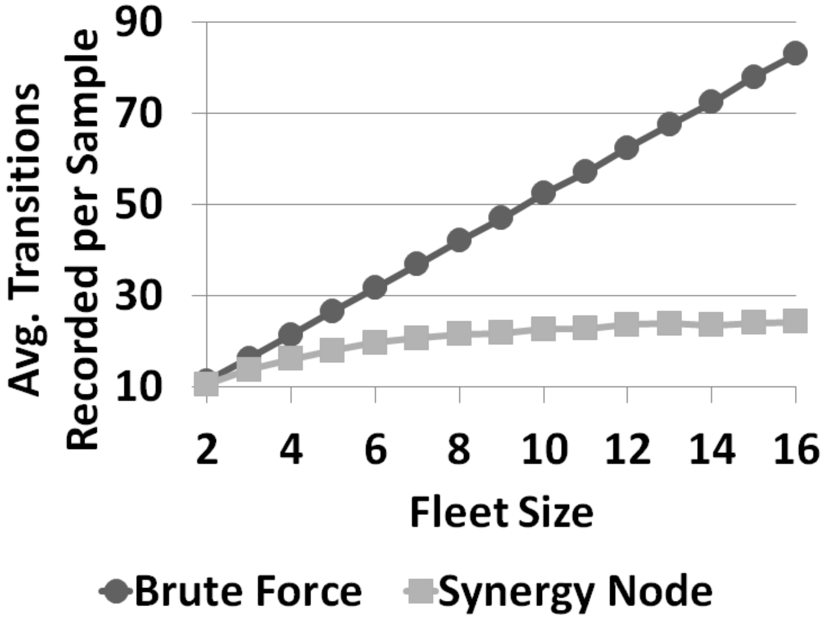


Figure 4.5: Average number of transitions per sample of the brute force and synergy node approaches.

The average number of transitions per sample of the brute force approach and the synergy node approach is shown in Figure 4.5. Note that we did not record transitions for all of the nodes, only the nodes contributing to the performance function. In other words, the brute force approach used $\sigma = \cup_i \sigma[\mathit{Vehicle}_i]$ for fleet size i, while the synergy node approach used $\sigma = \sigma[V^+]$. As the graph shows, the number of transitions increases linearly for the brute force approach, as expected. For the synergy node approach, on the other hand, the curve behaves logarithmically. This is because, as the number of vehicles increases, the proportion of time that all are running simultaneously decreases. Therefore, the number of transitions between the states of the synergy node decreases. This also means that the sample path for the synergy node takes fewer evaluations to estimate the performance.

Finally, the average number of times the sampler must draw from a distribution is shown in Figure 4.6. As the graph shows, the addition of the synergy node does

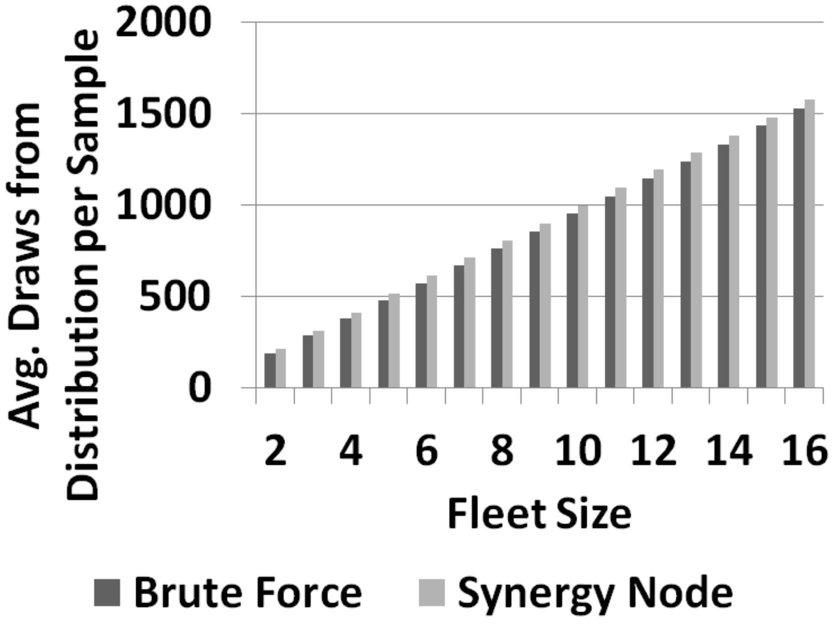


Figure 4.6: Average number of draws from a distribution per sample of the brute force and synergy node approaches.

increase the complexity of generating each sample; however, the complexity is not greatly increased, as the curve suggests only a small, constant-factor increase.

4.4 Conclusions

In this chapter, we formalized factored performance functions for the CTBN. Existing CTBN inference algorithms support estimating arbitrary functions over the behavior of the network, but by factoring these functions onto the nodes of the network, we can achieve a representation of performance that we argue is more easily understood and implemented. Furthermore, to support more complex interactions in which the performance function cannot be factored in a straightforward manner, we show how to maintain a factorization by augmenting the structure of the CTBN with synergy nodes. We argue that such complex performance information is more easily understood in terms of the synergistic relationship between nodes. We showed a real-world example in which the synergy node is able to capture the performance evaluation between multiple nodes without a significant increase in complexity and without degrading accuracy.

As future work, we would like to demonstrate other scenarios that use synergy nodes, as well as families of performance functions, on a wider variety of real-world networks. The concept of performance function families opens up opportunities for the CTBN to be useful for continuous-time multi-objective optimization problems.

CHAPTER 5

NODE ISOLATION FOR APPROXIMATE INFERENCE

In this chapter, we present three novel CTBN marginalization methods that approximate a set of conditional intensity matrices with a single conditional intensity matrix. This allows some forms of inference to be carried out in the CTBN without expanding to the full joint intensity matrix.

5.1 Background Work

The inference problems we consider in this chapter are filtering and prediction with point evidence. Recall that the state-space of $\mathbf{X}$ is exponential in the number of nodes. Therefore, we need a way to subdivide the network and calculate $P(\mathbf{S}(t))$ for smaller subnetworks $\mathbf{S} \subset \mathbf{X}$. The key idea is approximate node marginalization, which removes the incoming arcs and replaces the set of conditional intensity matrices with a single unconditional intensity matrix that approximates the node's former dynamics. We start by presenting the approximate inference algorithm developed in [17] that accomplishes this. The algorithm is based on the clique tree algorithm for Bayesian networks and can be used with their two methods for node marginalization.

5.1.1 Clique Tree Algorithm

In this adaptation of the clique tree algorithm, amalgamation of conditional intensity matrices is used in place of products, and marginalization is done by approximating a single unconditional intensity matrix from a set of conditional intensity matrices. Because cycles are allowed in a CTBN, the algorithm does not necessarily form a tree of connected cliques but an undirected graph of connected cliques in which cycles are possible.

5.1.1.1 Initialization. First, we construct a graph $\mathcal{G}$ consisting of cliques. To do this, we moralize the graph by connecting all the parents of a common child node with undirected edges. This creates a clique consisting of each child and all of its parents, because each child is connected to its parents, and each parent of the child is connected to each other. We then replace all directed edges with undirected edges. We associate each node with the clique that contains it and all of its parents.

Let $A_i \subseteq C_i$ be a set of nodes in clique C_i. Let S_{ij} be the set of nodes in $C_i \cap C_j$. Let $N_i = \{C_j | S_{ij} \neq \emptyset, i \neq j\}$ denote the set of neighboring cliques to C_i. That is, neighbors are cliques that share at least one node.

For each clique i, we compute the initial distribution $P^0_{C_i}$ using Bayesian network inference on $\mathcal{B}$.

We calculate the initial intensity potential f_i for each clique C_i as

$$f_i = \prod_{x \in C_i} \mathbf{Q}_{X|\mathbf{Pa}(X)}.$$

That is, we amalgamate the unconditional intensity matrices of C_i.

5.1.1.2 Message Passing. The message passing process is used to calibrate the network, removing all edges between cliques so that inference can be performed over each clique separately. Its goal is to compute an approximate probability distribution over the trajectories of the nodes in each C_i by representing each clique as a homogeneous Markov process.

To do this, clique C_i passes messages about the dynamics of the nodes in S_{ij} to its neighbors. Clique C_i sends a message, denoted $\mu_{i\to j}$ to clique C_j once it has received $\mu_{k\to i}$ for $k \in N_i$, except possibly j. Once all incoming messages have been received, clique C_i amalgamates the messages with f_i and computes outgoing messages to C_j by marginalizing out all variables in C_i not in S_{ij}. This marginalization, denoted $\text{marg}_Y^P(\mathbf{Q}_{S|C})$, takes a CIM $\mathbf{Q}_{S|C}$ and eliminates the nodes in Y. Formally,

$$\mu_{i\to j} = \text{marg}_{(C_i - S_{ij})}^{P_{C_i}} \left(f_i \times \left(\prod_{k\in N_i, k\neq j} \mu_{k\to i} \right)\right)$$

After each clique C_i has received all incoming messages, the clique's local intensity matrix is computed as

$$\mathbf{Q}_{C_i} = f_i \times \prod_{k\in N_i} \mu_{j\to i}.$$

In general, this single homogeneous process $\mathbf{Q}_{C_i}$ is not able to represent the dynamics of clique C_i exactly. As such, this clique tree algorithm for CTBNs is an approximation technique.

5.1.1.3 Queries. Once the clique tree algorithm terminates, we estimate the probabilities for any clique i as

$$P(C_i(t)) \approx P_{C_i}^0 \exp(\mathbf{Q}_{C_i} t).$$

Evidence can be incorporated at instantaneous points in time by re-conditioning the initial distribution on the evidence at the start of each interval. Let $\mathbf{e}^t$ denote point evidence at time t. If we have point evidence at the initial time $\mathbf{e}^0$, we condition the initial distribution on that evidence, $P(C_i(0)|\mathbf{e}^0)$. Then for $t > 0$,

$$P(C_i(t)|\mathbf{e}^0) \approx P(C_i(0)|\mathbf{e}^0)\exp(\mathbf{Q}_{C_i}t).$$

With point evidence at time t_1, we compute $P(C_i(t_1)|\mathbf{e}^0)$ and condition on $\mathbf{e}^{t_1}$, which yields

$$P(C_i(t_1)|\mathbf{e}^0, \mathbf{e}^{t_1}).$$

Now suppose there are k observations at times $t_1 < t_2 < \cdots < t_k$. Then for $t > t_k$,

$$P(C_i(t)|\mathbf{e}^0, \mathbf{e}^{t_1}, \ldots, \mathbf{e}^{t_k}) \approx P(C_i(t_k)|\mathbf{e}^0, \mathbf{e}^{t_1} \ldots, \mathbf{e}^{t_k})\exp(\mathbf{Q}_{C_i}(t_k - t_{k-1})).$$

5.1.2 Marginalization

The clique tree algorithm depends on the marginalization step—taking a joint intensity matrix amalgamated over a set of nodes and returning a smaller intensity matrix amalgamated over a subset of those nodes. The marginalization operator takes a CIM $\mathbf{Q}_{S|C}$, a set of nodes $Y \subset S$, and an initial distribution P over the nodes in S. It returns a reduced CIM of the form $\mathbf{Q}_{S'|C} = \text{marg}_Y^P(\mathbf{Q}_{S|C})$, where $S' = S - Y$.

There are multiple ways of approximating these smaller intensity matrices, and [17] presented two, the linearization method and the subsystem method.

5.1.2.1 Linearization Method. Let $s' \oplus y$ be the full state instantiation of S to s' and y to Y. Consider a transition from $s_1 = s'_1 \oplus y_1$ to $s_2 = s'_2 \oplus y_2$ during some interval Δt. Ideally, the marginalization process would ensure that

$$P(s'_2|s'_1, c) = \sum_{y_1, y_2} P(s'_2 \oplus y_1|s'_1 \oplus y_2, c)P(y_1|s'_1, c)$$

for all Δt, s_1, s_2, and c. As this is not generally possible, the linearization approach takes advantage of two approximations. First, we assume that y does not change over time such that we can use the values of y at the beginning of the interval, i.e.,

$$P(s_2'|s_1', c) \approx \sum_y P(s_2' \oplus y|s_1' \oplus y, c)P^0(y|s_1', c),$$

where P^0 is the distribution at the beginning of the interval. Second, we use the linear approximation to the matrix exponential,

$$\exp(\mathbf{Q}\Delta t) \approx \mathbf{I} + \mathbf{Q}\Delta t,$$

to give us an approximation of the intensity matrix as

$$\mathbf{Q}_{(S'|C)}(s_1' \rightarrow s_2'|c) \approx \sum_y \mathbf{Q}_{S|C}(s_1' \oplus y \rightarrow s_2' \oplus y|c)P^0(y|s_1', c).$$

<u>5.1.2.2 Subsystem Method</u>. Given an entrance distribution P_S^0 for subsystem S, we compute the holding time and the exit distribution P_S^E. Let $\mathbf{1}$ be a column vector of ones over the states of S. The distribution over the holding time within a subsystem, holding Y constant in state y_i, is

$$F(t) = 1 - P_S^0 \exp(\mathbf{Q}_S t)\mathbf{1}.$$

To approximate the expected holding times, we set the holding intensity value to be

$$q_{i,i} = -1/(P_S^0(-\mathbf{Q}_S)^{-1}\mathbf{1}).$$

The exit distribution is computed from the matrix $\mathbf{M}_S$ of the embedded Markov chain for the subsystem S. Let $q^S_{i,j}$ denote entry i, j of the intensity matrix $\mathbf{Q}_S$. The entries of $\mathbf{M}_S$ are

$$m_{i,j} = \begin{cases} 0 & \text{if } i = j \\ q^S_{i,j}/|q^S_{i,i}| & \text{otherwise} \end{cases}.$$

The exit distribution is then computed as

$$P^E_S = (\mathbf{I} + \mathbf{M}_S)^{-1}.$$

The exit distribution of subsystem S is used to weight the transition probabilities between subsystems, used for computing $q_{i,j}$ for $i \neq j$.

The subsystem method relaxes the assumption of the linearization method that the node being marginalized out does not change over time.

5.2 Node Isolation

The linearization and subsystem methods tend to degrade as Δt increases. To counteract this, the approximations can be time-sliced. The marginalization is done over a shorter interval, and the new initial distribution is computed for use in the next time-slice. As the time-slices become shorter, the accuracy of the two methods improves. However, this is at the cost of increased computational time.

Instead of trying to marginalize with intensity matrices that approximate the immediate dynamics of the cliques, we try to marginalize with intensity matrices that describe the long-term behavior. We call this approach "node isolation."

The following two sections show three methods for node isolation. First, we show a sample-based approach for node isolation that estimates unconditional intensity

Algorithm 5.3 Sample-based node isolation method for computing an unconditional intensity matrix.

```
IsolateNode($\sigma, X$)
 1: $\mathbf{Q}_X \leftarrow \mathbf{0}$
 2: $\boldsymbol{v} \leftarrow \mathbf{0}$
 3: for each $\langle t_s, t_e, X(t_s)\rangle \in \sigma[X]$
 4:     $x_i \leftarrow X(t_s)$
 5:     $x_j \leftarrow X(t_e)$
 6:     $q_{i,i} \leftarrow q_{i,i} + (t_e - t_s)$
 7:     $q_{i,j} \leftarrow q_{i,j} + 1$
 8:     $v_i \leftarrow v_i + 1$
 9: end for
10: for each $x_i \in X$
11:     $q_{i,i} \leftarrow q_{i,i} \;/\; v_i$
12:     $q_{i,i} \leftarrow -(1/q_{i,i})$
13:     $z \leftarrow 0$
14:     for each $x_i, x_j \in X \ni i \neq j$
15:         $z \leftarrow z + q_{i,j}$
16:     end for
17:     for each $x_i, x_j \in X \ni i \neq j$
18:         $q_{i,j} \leftarrow q_{i,j} \cdot (|q_{i,i}|/z)$
19:     end for
20: end for
21: return $\mathbf{Q}_X$
```

matrices from a single, sufficiently long trajectory of the network. Second, we show a closed-form solution with two variations that more efficiently computes the unconditional intensity matrices. Note that while these last two are closed-form solutions, they are still approximations because they are approximating the behavior of multiple conditional intensity matrices with a single unconditional intensity matrix.

5.2.1 Sample-Based Node Isolation

First we present the sample-based method for node isolation. This method approximates the unconditional intensity matrix from a trajectory by aggregating the average sojourn times and transition counts between subsystems.

Algorithm 5.3 shows the pseudocode for how node isolation can be accomplished for an arbitrary node in the network. The algorithm accepts a trajectory, as generated from Algorithm 2.1, and the node to be isolated and returns the unconditional intensity matrix of the isolated node.

First, lines 1-2 of the algorithm initialize a zero unconditional intensity matrix for X and a zero vector for the states of X. Although the states of ancestor nodes may be changing throughout the trajectory, lines 3-9 are concerned *only* with state changes of the node to isolate. Specifically, line 6 calculates the total amount of time spent in each state of the node, line 7 counts how many times the system has transitioned between each state of the node, and line 8 counts the total number of transitions that have occurred between states of the node. Next, lines 10-20 transform these statistics into an unconditional intensity matrix. Line 11 calculates the average sojourn time (amount of time spent per visit) for each state of the node, which is taken as the expected sojourn time. Therefore, line 12 takes the negative reciprocal for use in the exponentially decreasing probability function. The relative number of transitions from a state to the remaining states represents the transition probabilities, which lines 13-19 normalize in accordance with the intensity matrix row constraints.

5.2.2 Closed-Form Node Isolation

Now we describe the closed-form solution for CTBN node isolation. First, we amalgamate all of the ancestors of the node to isolate into a single supernode, creating a single parent of the node to isolate consisting of m states. Then we amalgamate the node to isolate and its parent. The states of the amalgamated supernode are represented as (p_k, c_l), meaning the parent p is in state k while the child c is in state

$$\boldsymbol{Q} = \begin{array}{c|c|c|c} & (p_1,c_1)\dots(p_m,c_1) & \cdots & (p_1,c_n)\dots(p_m,c_n) \\ \begin{matrix}(p_1,c_1)\\ \vdots \\ (p_m,c_1)\end{matrix} & Q_1 & \cdots & Q_{1,n} \\ \hline \vdots & \vdots & \ddots & \vdots \\ \hline \begin{matrix}(p_1,c_n)\\ \vdots\\ (p_m,c_n)\end{matrix} & Q_{n,1} & \cdots & Q_n \end{array}.$$

Figure 5.1: The amalgamated intensity matrix for a node and its parent as a block matrix.

l. We find the full joint intensity matrix of the parent with m states and the child of n states as shown in Figure 5.1.

The intensity submatrix $\boldsymbol{Q}_i$ along the diagonal of $\boldsymbol{Q}$ denotes the dynamics of subsystem i when the state of the child is held constant. The intensity submatrix $\boldsymbol{Q}_{i,j}$ is a diagonal matrix showing the transition probabilities from the states of subsystem i to the states of subsystem j. Each of these matrices is diagonal because two nodes of a CTBN cannot transition simultaneously, and an off-diagonal entry in the $\boldsymbol{Q}_{i,j}$ would represent a transition in both p and c. Thus, when the state of the child changes, the state of the parent must be held constant.

To find the unconditional intensity matrix of the child, we replace each submatrix $\boldsymbol{Q}_i$ with the mean sojourn time in subsystem i and replace each submatrix $\boldsymbol{Q}_{i,j}$ with the probability of transitioning from subsystem i to subsystem j.

First we find the stationary distribution of the embedded Markov chain of the entire system. The off-diagonal entries of $\boldsymbol{Q}$ are normalized by the diagonal to get the transition probabilities and the diagonals are zeroed out. The stationary distribution of this embedded Markov chain can then be found using a technique called first step analysis, which breaks down the problem by analyzing the possible transitions on the first step and then applying the Markov property [24]. The system of equations for the stationary distribution of the embedded Markov chain is

$$P_{i,j} = \sum_{\substack{1 \le k \le m \\ k \neq i}} \frac{(p_{i,k}, c_{j,j})}{|(p_{i,i}, c_{j,j})|} P_{k,j} + \sum_{\substack{1 \le l \le n \\ l \neq j}} \frac{(p_{i,i}, c_{j,l})}{|(p_{i,i}, c_{j,j})|} P_{i,l} \tag{5.1}$$

and, to ensure that the probabilities form a valid probability distribution,

$$\sum_{1 \le k \le m} P_{k,j} + \sum_{1 \le l \le n} P_{i,l} = 1, \tag{5.2}$$

for $1 \le i \le m$ and $1 \le j \le n$, where $P_{i,j}$ is the steady-state probability the parent being in state i and the child being in state j. In other words, first step analysis sets up the quantity to be calculated starting from a state i in terms of the probability of transitioning from i to state j, multiplied by the quantity to be calculated starting from j, $\forall j \neq i$. Setting up this relationship from each state results in a system of equations that can be solved to yield the quantity for each state.

Alternatively, we could use the initial distribution P^0 for the values of $P_{i,j}$ instead of the steady-state probabilities. Using the steady-state probabilities attempts to describe the long-term dynamics of the node, while using the initial distribution attempts to describe the immediate dynamics of the node. For the mean sojourn time in the subsystem, we start by examining the diagonal matrices $\boldsymbol{Q}_i$:

$$\boldsymbol{Q_i} = \begin{pmatrix} (p_{1,1}, c_{i,i}) & (p_{1,2}, c_{i,i}) & \cdots & (p_{1,m}, c_{i,i}) \\ (p_{2,1}, c_{i,i}) & (p_{2,2}, c_{i,i}) & \cdots & (p_{2,m}, c_{i,i}) \\ \vdots & \vdots & \ddots & \vdots \\ (p_{m,1}, c_{i,i}) & (p_{m,2}, c_{i,i}) & \cdots & (p_{m,m}, c_{i,i}) \end{pmatrix}$$

We can compute the expected time to leave the subsystem also using first step analysis. The system of equations for the expected time $T_{i,j}$ to leave subsystem i starting from each state j in the subsystem is:

$$T_{i,j} = \frac{1}{|(p_{j,j}, c_{i,i})|} + \sum_{\substack{1 \leq l \leq m \\ l \neq j}} \frac{(p_{j,l}, c_{i,i})}{|(p_{j,j}, c_{i,i})|} T_{i,l} \tag{5.3}$$

for $1 \leq j \leq m$. This gives the mean time to leave the subsystem starting from each state. Now we need the entry distribution into the subsystem, which we find by weighting the transition probability into each state of the subsystem by the originating state's stationary distribution. Each of the other subsystems can transition once into each state of the subsystem, as shown by the off-diagonal matrices $\boldsymbol{Q}_{i,j}$:

$$\boldsymbol{Q_{i,j}} = \begin{pmatrix} (p_{0,0}, c_{i,j}) & 0 & \cdots & 0 \\ 0 & (p_{1,1}, c_{i,j}) & \cdots & 0 \\ \vdots & & \ddots & \vdots \\ 0 & 0 & \cdots & (p_{m,m}, c_{i,j}) \end{pmatrix}$$

The transition probability between subsystems i and j is:

$$c_{i,j} = \sum_{k=1}^{m} \left(\frac{(p_{k,k}, c_{i,j})}{|(p_{k,k}, c_{i,i})|} P_{k,i} \right) \tag{5.4}$$

The expected sojourn time in subsystem i is:

$$c_i = \sum_{l=1}^{n} T_{i,l} \left(\frac{\sum_{\substack{k=1 \\ k \neq i}}^{m} \left(\frac{(p_{l,l}, c_{k,i})}{|(p_{l,l}, c_{k,k})|} P_{l,k} \right)}{\sum_{\substack{k=1 \\ k \neq i}}^{m} c_{k,i}} \right) \tag{5.5}$$

Having the expected sojourn time in each subsystem and the transition probabilities betweens subsystems, we can now populate the unconditional intensity matrix $\boldsymbol{Q}'$ for the child:

$$\boldsymbol{Q}' = \begin{pmatrix} -1/c_1 & c_{1,2}/z_1 & \cdots & c_{1,n}/z_1 \\ c_{2,1}/z_2 & -1/c_2 & \cdots & c_{2,n}/z_2 \\ \vdots & \vdots & \ddots & \vdots \\ c_{n,1}/z_n & c_{n,2}/z_n & \cdots & -1/c_n \end{pmatrix}, \tag{5.6}$$

where z_i is a normalizer to ensure each row sums to zero.

The pseudocode for the closed-form node isolation method is given in Algorithm 5.4. The algorithm accepts a child node and its parents. The algorithm returns an unconditional intensity matrix for the child node. Lines 1 and 2 amalgamate all of the parents of the child node by repeated calls to Algorithm 2.2 to amalgamate all nodes in $\mathbf{Pa}(X)$. This allows each combination of parent states to be represented as a single parent state. Line 3 then amalgamates the parent and child into one intensity matrix upon which to work. Line 4 calculates the steady-state distribution of the parent by solving the system of equations given in Equations 5.1 and 5.2. Line 5 calculates the expected sojourn time in each subsystem of the parent by solving the system of equations given in Equation 5.3. Lines 6-11 iterate over the states of the child, while line 9 calculates the transition probabilities between the child states using Equation 5.4. Lines 12-14 calculate the expected sojourn time in each state of the child using Equation 5.5. Line 15 uses the individually computed values of lines 9 and 13 to construct the matrix, normalizing the rows of the matrix to create a valid intensity matrix, as per Equation 5.6.

Compared to the sample-based IsolateNode in Algorithm 5.3, this closed-form solution performed substantially faster in all experiments. Furthermore, as the number of samples increased, the unconditional intensity matrix returned by the

Algorithm 5.4 Closed-form node isolation method for computing an unconditional intensity matrix.

```
$IsolateNode(X \cup \mathbf{Pa}(X))$
 1: $Y \leftarrow$ Amalgamate($\mathbf{Pa}(X)$) // Equation 6.7
 2: replace $\mathbf{Pa}(X)$ with $Y$
 3: $\mathbf{Q}_{Y \cup X} \leftarrow$ Amalgamate($Y \cup X$)
 4: $P \leftarrow$ GetSteadyStateDistribution($\mathbf{Q}_{Y \cup X}, Y$) // Equations 5.1 and 5.2
 5: $T \leftarrow$ GetExpectedTimeInSubsystems($\mathbf{Q}_{Y \cup X}, Y$) // Equation 5.3
 6: $n \leftarrow |X|$
 7: for $i = 1, \ldots, n$
 8:     for $j = 1, \ldots, n$
 9:         $c_{i,j} \leftarrow$ GetTransitionProbability($\mathbf{Q}_{Y \cup X}, Y, P$) // Equation 5.4
10:     end for
11: end for
12: for $i = 1, \ldots, n$
13:     $c_i \leftarrow$ GetExpectedTimeInState($\mathbf{Q}_{Y \cup X}, Y, T, P, c_{1,i}, \ldots, c_{n,i}$) // Equation 5.5
14: end for
15: $\mathbf{Q}_X \leftarrow$ ConstructIntensityMatrix($c_1, \ldots, c_n, c_{1,1} \ldots, c_{n,n}$) // Equation 5.6
16: return $\mathbf{Q}_X$
```

IsolateNode(σ, X) algorithm converged to the unconditional intensity matrix calculated by this closed-form solution. The closed-form node isolation method requires solving an $m \times n$ matrix, while the subsystem method requires solving m $n \times n$ matrices. The linearization method is linear in the size of the $m \times n$ matrix.

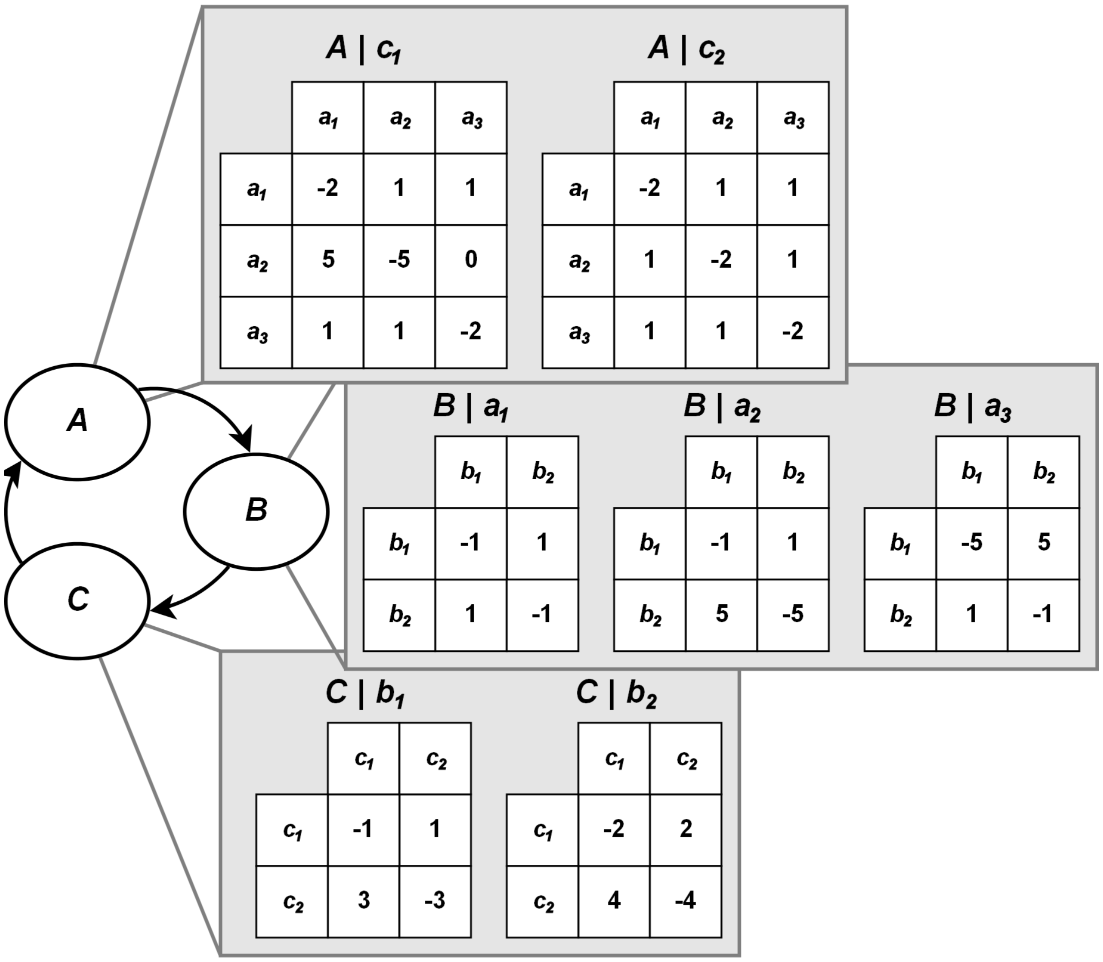
5.2.3 Node Isolation in Cycles

One difficulty of node marginalization is that the dynamics of a node depend on all of its ancestors. If the network is a directed acyclic graph (DAG), then we can marginalize each layer in succession, and the complexity of isolation depends on the number of immediate parents to each node. However, cycles are allowed in CTBNs. When a cycle is introduced, every node in the cycle must be included to marginalize any node in the cycle, because every node in the cycle is an ancestor of every other node in the cycle.

In this section, we show an iterative step to our node isolation algorithms that avoids dealing with the entire cycle all at once. The idea is that, given a cycle, we start at an arbitrary node in the cycle and temporarily remove the incoming arc. Previously, the node had a set of conditional intensity matrices, whereas now we need to replace it with one unconditional intensity matrix. While this unconditional intensity matrix depends on the dynamics of the parent that was just removed, we simply use an unconditional intensity matrix that is the average of its conditional intensity matrices.

Now, we isolate its child node, finding its unconditional intensity matrix. Depending on the actual parameters, this unconditional intensity matrix of the isolated child may be a poor approximation, because of how we constructed the unconditional intensity matrix of its parent. However, it is likely a better approximation of the unconditional intensity matrix than that of its parent. This process repeats, isolating nodes around the cycle, back to the originally chosen node. Now, this node is the child, and its conditional intensity matrices are used for its isolation. This process continues to loop around the cycle until convergence. This process is analogous to loopy belief propagation in cyclic graphs, such as in Markov random fields and in Bayesian networks in which the acyclic constraint has been relaxed [36].

We demonstrate this process on the simple, three-node network given in Figure 5.2. We used the algorithm for finding the closed-form solution from the previous section on the whole network to isolate node A. This was taken as the ground truth. We compared this unconditional intensity matrix $\mathbf{Q}_A$ with the unconditional intensity matrix $\mathbf{Q}'_A$ calculated from the iterative process described above. Error was calculated as



A | c_1

	a_1	a_2	a_3
a_1	-2	1	1
a_2	5	-5	0
a_3	1	1	-2

A | c_2

	a_1	a_2	a_3
a_1	-2	1	1
a_2	1	-2	1
a_3	1	1	-2

B | a_1

	b_1	b_2
b_1	-1	1
b_2	1	-1

B | a_2

	b_1	b_2
b_1	-1	1
b_2	5	-5

B | a_3

	b_1	b_2
b_1	-5	5
b_2	1	-1

C | b_1

	c_1	c_2
c_1	-1	1
c_2	3	-3

C | b_2

	c_1	c_2
c_1	-2	2
c_2	4	-4

Figure 5.2: Example CTBN to demonstrate iterative isolation in cycle.

$$\sum_{i=1}^{3} \sum_{j=1}^{3} \frac{|q_{i,j} - q'_{i,j}|}{|q_{i,j}|},$$

which is the sum of the relative errors between entries in the two matrices. After three iterations around the cycle, the process converged and the cumulative relative error for node A was 1.4%.

This experiment demonstrates the feasibility of iterative node isolation in cycles. More experiments are needed to analyze how the approach handles larger cycles with more states and more varied parameters and how these may affect accuracy. Further-

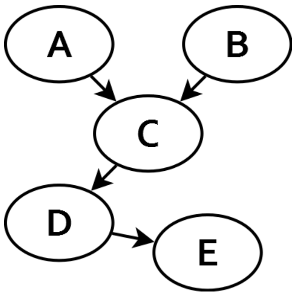


Figure 5.3: Example CTBN of a simple system.

more, we currently do not know if convergence is guaranteed for the unconditional intensity matrices in cycles. However, we can see the potential benefit of the approach even with this single, small cycle. The iterative approach works with at most half of the state-space at any one time. As cycles grow longer, the ability to work with only a single node and its immediate parents becomes even more beneficial.

5.3 Experiments

We compared the node isolation techniques with the linearization and subsystem methods on a simple, synthetic network shown in Figure 5.3 and the drug effect network from Figure 2.3 by plugging in the different node marginalization methods into the clique tree algorithm. We computed the exact probabilities for all the nodes through time using the forward-backward algorithm from Section 2.2.3. We measured the difference of the approximate distribution from the exact distribution using KL-divergence, defined formally as follows.

Definition 5.3.1. *(KL-Divergence) For discrete probability distributions P and Q, the KL-divergence of Q from P is defined as*

$$D_{KL}(P \| Q) = \sum_i P(i) \log_2 \left(\frac{P(i)}{Q(i)} \right).$$

KL-divergence is also called "relative entropy" and quantifies the cost of using a wrong distribution Q instead of P [36]. We can use it as a way to compare the accuracy between approximate distributions based on the true distribution.

In our experiments, we computed the average KL-divergence of all nodes in the network for each point in time. Let $P(X(t))$ denote the true distribution of X at time t, and let $P'(X(t))$ denote the approximate distribution for X at time t. For a given point in time t, we calculate the average KL-divergence as

$$\frac{1}{|\mathbf{X}|} \sum_{X \in \mathbf{X}} D_{KL}(P(X(t)) \| P'(X(t))).$$

5.3.1 Synthetic Network

First, we tested node marginalization on the simple, synthetic network. Each node has 2 or 3 states for a total state-space size of 72 over the entire network. The initial distributions and intensity matrices for all the nodes can be found in Appendix D.

We computed $P(\mathbf{X}(t))$ and $P'(\mathbf{X}(t))$ at each point in time on the interval $[0, 10)$ hours with t advancing in 18-minute increments. First, we compared the linearization method (Linear), the subsystem method (Subsystem), and the node isolation method using the steady-state distributions (Isolation-SS). The results are shown in Figure 5.4. The results show that the node isolation is better able to estimate

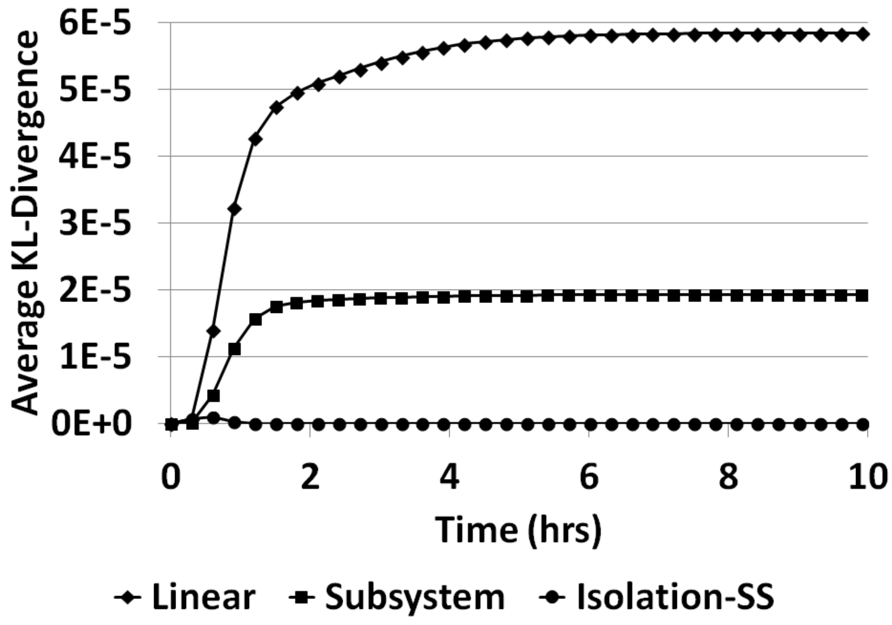


Figure 5.4: Average KL-divergence for the linearization, subsystem, and isolation methods on the simple, synthetic network.

the probabilities for this network, meaning that the unconditional intensity matrices computed by the node isolation method provide a better approximation.

Next, we compare four variations of the node isolation method. The first and second are the sample-based node isolation method as computed from Algorithm 5.3 using trajectories sampled to 1000 transitions (Isolation-1K) and 10,000 transitions (Isolation-10K). The third is the closed-form node isolation method using the initial distribution instead of the steady-state distribution (Isolation-P0). The fourth is the closed-form node isolation method using the steady-state distribution (Isolation-SS). The results are shown in Figure 5.5. We see that the Isolation-SS variant maintains the lowest KL-divergence and that the longer sample improves the accuracy of the sample-based method.

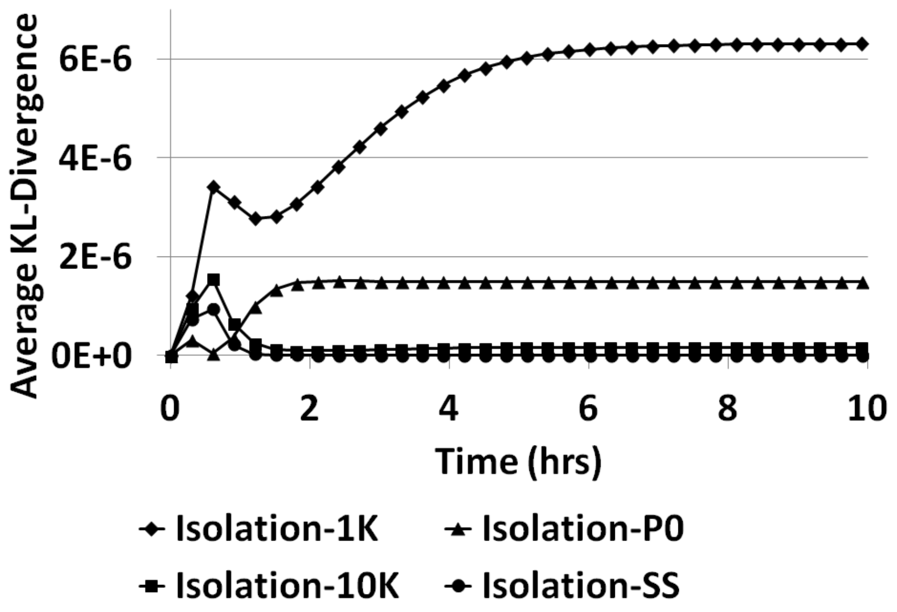


Figure 5.5: Average KL-divergence for sample-based and closed-form node isolation methods.

5.3.2 Drug Effect Network

For the drug effect network, we averaged the KL-divergence of all nodes at each point in time on the interval $[0, 20)$ hours in 30-minute increments. This time we incorporated the point evidence of Eating(10.0) $= e_0$. We compared the linearization method, the subsystem method, the node isolation with the steady-state probabilities (Isolation-SS), the sample-based node isolation method with 1000 samples (Isolation-1K), and the node isolation with the initial distribution (Isolation-P0). The results are shown in Figure 5.6. We notice that the KL-divergence of the linear and subsystem methods increase as they try to infer further in time. The node isolation methods, on the other hand, level out in average KL-divergence at the start and again after the evidence is applied. In this case, node isolation using the initial distribution did better than the other four on the interval $[0, 10)$. Both node isolation methods become better

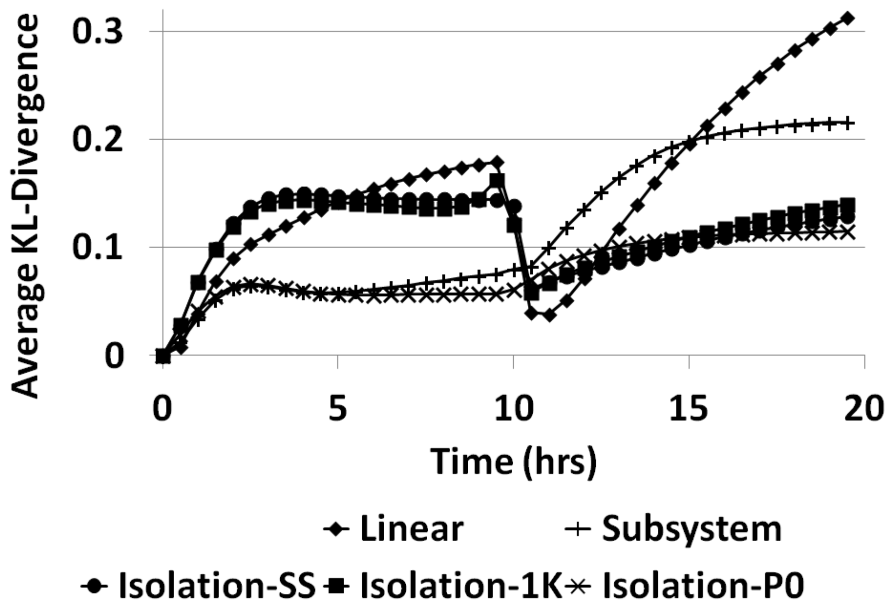


Figure 5.6: Average KL-divergence for different node marginalization methods on the drug effect network.

than the other two on the interval $[10, 20)$ after the evidence to Eating was applied. Again the node isolation methods were better able to describe the long-term behavior of the network, converging to a constant KL-divergence while the linearization and subsystem methods continue to increase past hour 20. The node isolation methods did not perform as well as with the synthetic, five-node network of the preceding section. Recall that the drug effect network includes a three-node cycle. The KL-divergence of these nodes are higher than that of the other nodes, indicating that cycles present a challenge for inference for all methods.

5.3.3 Ring Network

The cycle of the drug effect network appears to be especially challenging for these approximation methods. This experiment explores the difficulty by testing the node marginalization methods on cyclic networks of increasing length. For a ring network of size n, we construct the network by adding n two-state (T/F) nodes and connecting them as follows.

$$X_1 \to X_2 \to \cdots \to X_n \to X_1$$

Each node starts in the T state. The conditional intensity matrices for nodes X_1 and X_k for $k \geq 2$ are given as follows.

$$\mathbf{Q}_{X_1|X_n=T} = \mathbf{Q}_{X_k|X_{k-1}=T} = \begin{pmatrix} -k & k \\ k & -k \end{pmatrix}$$

$$\mathbf{Q}_{X_1|X_n=F} = \mathbf{Q}_{X_k|X_{k-1}=F} = \begin{pmatrix} -k & k \\ 2k & -2k \end{pmatrix}$$

We query the probability distribution of each node in the cycle on the interval $[0, 5)$ hours in 6-minute increments. We compute the average KL-divergence of the approximate distributions from the exact distributions over all the nodes in the cycle. We vary the size of the cycle from $n = 3$ to $n = 6$.

First, we tested the closed-form node isolation by itself as the length of the cycle increased. The results are shown in Figure 5.7. For the networks with these parameters at least, increasing the length of the cycle actually improved the approximation. We note that the closed-form method converges to non-zero average KL-divergence for the 3-node cycle, similar to what we saw with the drug effect network. Therefore, while the node isolation methods attempt to describe the long-term behavior, the unconditional intensity matrices calculated from the method are not guaranteed to converge to the true long-term behavior, as observed in these cases with cycles. Note that the temporary increase in KL-divergence around hour 1 is a result of the

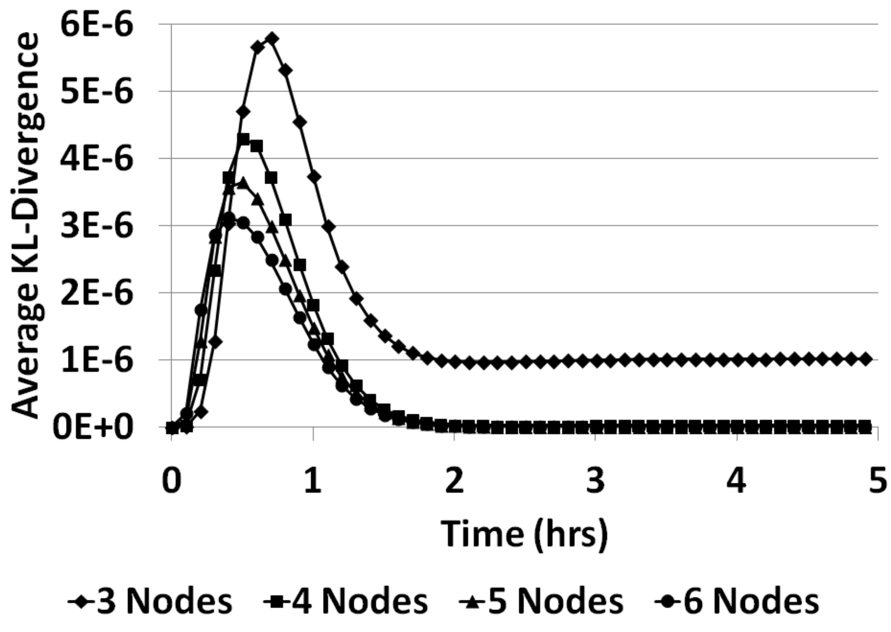


Figure 5.7: Average KL-divergence over all nodes for closed-form node isolation with increasing cycle length.

node isolation method's emphasis on approximating long-term behavior rather than immediate behavior, which results in less accurate approximations early on.

However, the node isolation methods still offer advantages over the linearization and subsystem methods. We show the five methods, the linearization method, the subsystem method, the closed-form node isolation method with the steady-state probabilities, the closed-form node isolation with the initial distribution, and the sample-based node isolation method with 1000 samples. For each method, we averaged the KL-divergence over all nodes and over all of the 6-minute increments. The results are shown in Figure 5.8.

The length of the cycle did not have much of an impact on the average KL-divergence for any of the methods. However, the experiments again show that the node isolation methods maintain lower KL-divergence. In this case, the node isolation

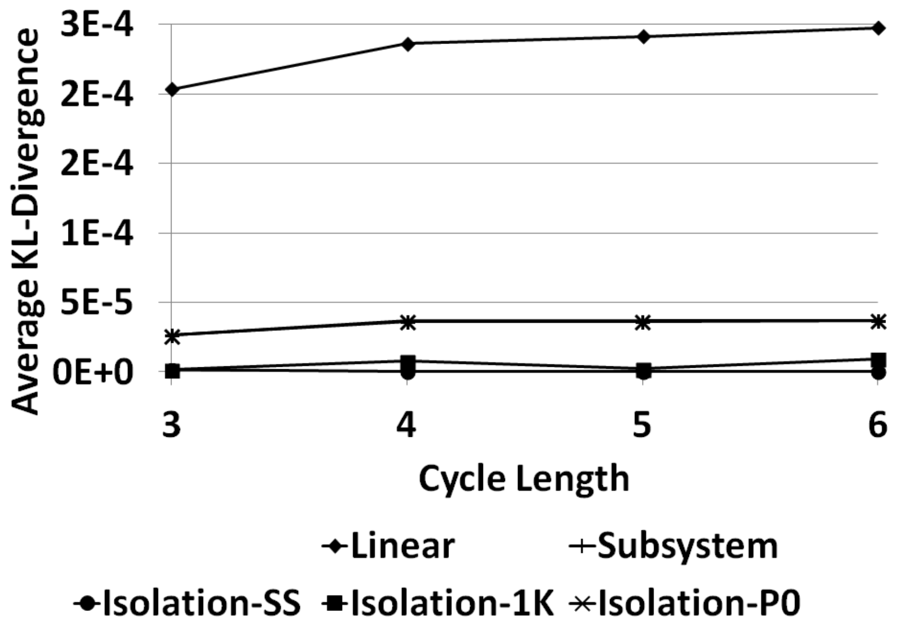


Figure 5.8: Average KL-divergence over all nodes and all timesteps for marginalization methods with increasing cycle length.

method with the initial distribution is nearly identical to the subsystem method, making them difficult to distinguish in Figure 5.8 because the two curves are overlapping. The magnitude of the KL-divergence for these cycles are small, as compared to the drug effect network. This seems to imply that the difficulty is not simply the presence of cycles but the parameters of the conditional intensity matrices of the nodes of those cycles. Further experiments on cycles with "hard" parameters is future work.

5.3.4 Cardiac Assist System

Finally, we compared the inference methods on a larger, real-world network. We used the model for a cardiac assist system (CAS), presented in [33], which is broadly used in the literature and based on a real-world system [82, 83]. In [33], they show how the CTBN is able to encode Dynamic Fault Trees (DFTs), which are reliability

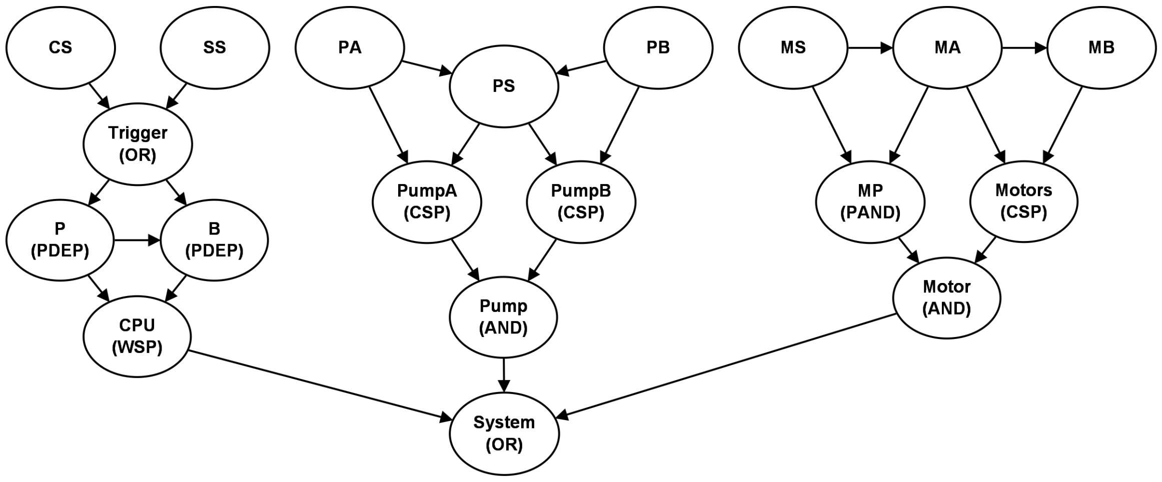


Figure 5.9: Cardiac assist system model.

models that use Boolean logic to combine series of lower-level failure events while preserving failure sequence information [84]. The intensity matrices of the CTBN are used to represent the gates available in the DFT, including AND, OR, warm spare (WSP), sequence enforcing (SEQ), probabilistic dependency (PDEP), and priority AND (PAND). Our model for this experiment is the DFT for the CAS system represented as a CTBN. The network is shown in Figure 5.10. The initial distributions and intensity matrices for all the nodes can be found in Appendix E. Of the various repair policies evaluated in [33], we use the repair rate of $\mu = 0.1$ for all components. The names of the nodes are given in Table 5.1.

We query the probability distribution of the System node's failed and non-failed states from 0 to 50 hours in 2 hour increments. For this model, exact inference is intractable, because the size of the state-space is over 6.6 million. Instead, we ran importance sampling for 100,000 samples and used this as our true distribution for calculating KL-divergence. The results are shown in Figure 5.10.

The results are consistent with the results of the previous experiments. While the KL-divergence increases early on for the both the sample-based node isolation

Table 5.1: CAS component names.

Abbreviation	Name	Subsystem
P	primary CPU	CPU
B	warm spare CPU	CPU
CS	cross switch	CPU
SS	system supervision	CPU
MA	primary motor	Motor
MB	cold spare motor	Motor
MS	switching component	Motor
PA	pump A	Pump
PB	pump B	Pump
PS	cold shared pump	Pump

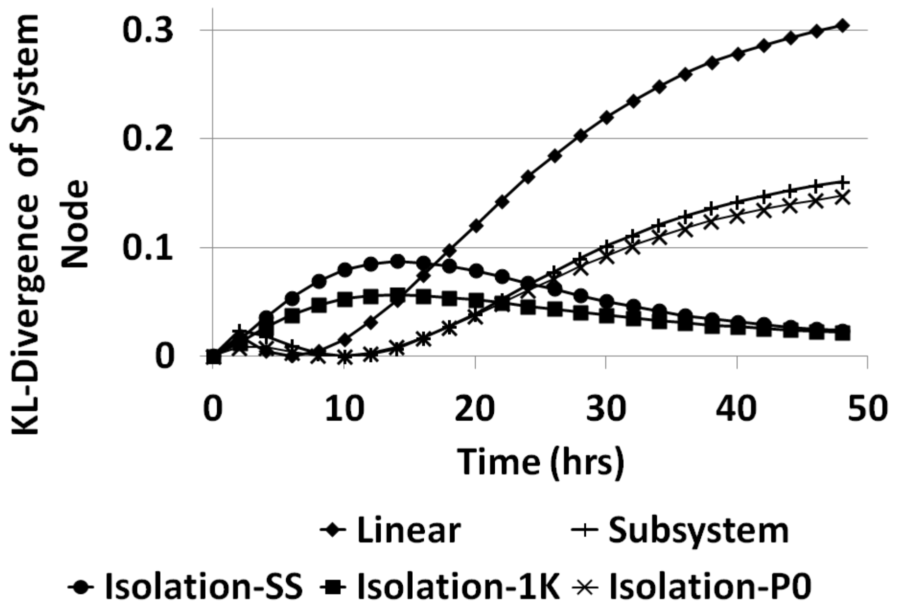


Figure 5.10: KL-divergence of System node for all marginalization methods.

method and the steady-state, closed-form node isolation method, it decreases below the other methods as the query progresses further out in time.

5.4 Conclusion

We have shown novel methods for approximate CTBN node marginalization that can better approximate a node's long-term behavior than previous methods. The node isolation methods take into account the probabilities of transitioning into and out of specific states of the subsystems to be marginalized out, rather than just computing the exit distribution and the mean time in the subsystem and then multiplying these by the entrance distribution.

Currently, the node isolation methods can only incorporate point evidence. Extending these methods to be able to support other types of evidence, in addition to being able to condition on later evidence, are left as future work.

Although node isolation is not a general CTBN inference algorithm, its ability to accurately estimate the long-term behavior of subsystems becomes especially useful in the next chapter, in which we apply the closed-form node isolation method to sensitivity analysis of CTBNs.

CHAPTER 6

SENSITIVITY ANALYSIS

In this chapter, we show how to perform sensitivity analysis on the performance functions introduced in Chapter 4. Sensitivity analysis looks at how variations in the input to a model affect the model's output and is useful in several contexts. In modeling, for example, sensitivity analysis can aid in model design, validation, and calibration. It can also be used to measure the robustness of model inference (the extent to which noise and uncertainty in the input affects model output) or to run hypothetical scenarios.

6.1 Background Work

We provide a formal definition of sensitivity analysis for probabilistic graphical models as follows.

Definition 6.1.1 (Sensitivity Analysis). *Let $\mathcal{M}$ denote a probabilistic graphical model specified with parameters $\mathcal{P}$. Let $\mathcal{R}$ be the result of some inference task over $\mathcal{M}$. Sensitivity analysis is the study of the relationship between $\Delta\mathcal{P}$ and $\Delta\mathcal{R}$ given $\mathcal{M}$.*

Research has been done on sensitivity analysis for many probabilistic networks, such as Bayesian networks [85, 86], Markov chains [87, 88, 89], Markov processes [90, 91, 92, 93, 94], and queuing networks [95, 96, 97]. To our knowledge, we present the first methods for sensitivity analysis specific to the CTBN model.

In our application of sensitivity analysis to CTBNs, the input is in perturbations to the parameters of the CTBN. Sensitivity analysis tests how changes to the network parameters affect the expected performance of the modeled system. Our application uses the CTBN for reliability modeling. We use the performance functions to represent reliability measures, such as mean time between failures (MTBF) and availability. Sensitivity analysis can then be performed with respect to these measures.

Reliability describes the ability of a system to function under stated conditions for a specified period of time [98]. To determine the reliability of a system, a model of the system is created and inference is performed over the model. Markov chains and Markov processes have often been used as reliability models. As a factored Markov process, the CTBN is a natural next step for reliability modeling, as it is able to represent more complex systems. As a relatively new model, the CTBN is only recently starting to be explored in the context of reliability analysis.

The work presented in this chapter shows how to perform reliability analysis more efficiently, taking advantage of the factored nature of the network and calculating quantities called potentials, which can be re-used for different queries. Instead of running inference over the whole network all at once and for each variation of parameters (e.g., uncertainty in the failure rates), the method presented here is able to isolate different subnetworks and only update the potentials when necessary.

6.1.1 Perturbation Realization

Although there are several methods for sensitivity analysis of Markov processes, this chapter builds on perturbation realization. Perturbation analysis using a single trajectory σ of an ergodic Markov process is discussed in [90], where the authors study the sensitivity of the steady-state performance of a Markov process with respect to its intensity matrix. They use trajectories to avoid costly computations on the intensity matrix itself. Because the full joint intensity matrix of the CTBN is exponential in the number of nodes, their approach is a natural candidate for extending sensitivity analysis to CTBNs. They are interested in the steady-state performance, regardless of the Markov process's initial state. Thus, they only need to use a single trajectory, provided the trajectory is long enough to converge to within some desired precision. The ergodic assumption assures that each state is reachable throughout the process and that the process will never reach an absorbing state, which would come to dominate the estimate the longer the process is run. Hence, the node isolation of Chapter 5, which effectively estimates long-term unconditional intensity matrices using the steady-state probabilities, can be used to subdivide the network and perform sensitivity analysis independently for different subnetworks.

6.1.1.1 Steady-State Probability Vector. Using the definition of the Markov process given in Section 2.2.1, let $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_n)$ denote the row vector representing the steady-state probabilities of the n states of the Markov process X. Letting $\mathbf{1} = (1, 1, \ldots)^T$ be a transposed n-sized vector, we have

$$\boldsymbol{\pi}\mathbf{1} = 1 \text{ and } \boldsymbol{\pi}\mathbf{Q} = \mathbf{0}.$$

6.1.1.2 Performance Measure. Let $f : X \to \mathbb{R}$ (mapping the state space of the Markov process to the real numbers) be a performance function of the process. This function is used to calculate the performance measure, defined as the function's expected value,

$$\eta = \sum_{i=1}^{n} \pi_i f(x_i) = \boldsymbol{\pi}\mathbf{f}, \tag{6.1}$$

where $\mathbf{f} = (f(x_1), f(x_2), \ldots, f(x_n))^T$ is a column vector and each entry is a state's performance function value. By themselves, the transition intensities of a Markov process do not encode state values, only transition probabilities. The performance function f, on the other hand, allows us to attach cost/reward values to the states. The performance measure represents the expected cost/reward per unit time of running the process. Furthermore, the performance measure gives direction for performing sensitivity analysis, because now we can measure how changes to the intensity matrix affect performance.

6.1.1.3 Partial Derivatives. Suppose that the intensity matrix $\mathbf{Q}_X$ changed to $\mathbf{Q}'_X = \mathbf{Q}_X + \epsilon\Delta\mathbf{Q}_X$ with ϵ being an arbitrarily small positive number and with $\Delta\mathbf{Q}_X$ being an $n \times n$ matrix (referred to as the perturbation matrix) such that each row in $\Delta\mathbf{Q}_X$ is also constrained to sum to zero, $\sum_j q_{i,j} = 0 \; \forall i$, such that $\mathbf{Q}'_X$ maintains the same constraints on the intensity matrix as given in Section 2.3.1. Therefore, $\epsilon\Delta\mathbf{Q}_X$ perturbs the values of $\mathbf{Q}_X$ to $\mathbf{Q}'_X$. Let X' be the Markov process with transition intensity matrix $\mathbf{Q}'_X$. The performance measure of X' can be decomposed as $\eta_\epsilon = \eta + \Delta\eta$. The derivative of η with respect to $\Delta\mathbf{Q}_X$ is then defined as

$$\frac{\partial\eta}{\partial\Delta\mathbf{Q}_X} = \lim_{\epsilon\to 0}\frac{\eta_\epsilon - \eta}{\epsilon}. \tag{6.2}$$

Analogously,

$$\frac{\partial\mathbf{Q}_X}{\partial\Delta\mathbf{Q}_X} = \lim_{\epsilon\to 0}\frac{\mathbf{Q}'_X - \mathbf{Q}_X}{\epsilon} = \mathbf{Q}_X.$$

As a derivative, Equation (6.2) can be thought of as the sensitivity of η with respect to the changes in $\mathbf{Q}_X$, i.e., in the direction of $\Delta\mathbf{Q}_X$.

6.1.1.4 Perturbation Realization. The concept of perturbation realization is the idea that any change in a parameter's value can be decomposed into a sum of the effects of many individual changes on a trajectory [90]. Therefore, the average effect of each individual change in the $\Delta\mathbf{Q}_X$ matrix can be measured and recorded in an $n \times n$ matrix called the realization matrix, denoted $\mathbf{D}_X$, where each entry $d_{i,j}$ is called a realization factor.

Let $X(t')^{\{j\}}$ denote the state of the Markov process X at time $t' > t$ having been in state x_j at time t. Let $S(t,i)^{\{j\}}$ be the first time a transition from state x_j to state x_i occurs since time t, defined as

$$S(t,i)^{\{j\}} = \inf\{t' : t' \geq t, X(t')^{\{j\}} = i\}.$$

Then, provided the performance measure is bounded, Cao and Chen [90] show that each entry $d_{i,j}$ in the realization matrix $\mathbf{D}_X$ is

$$d_{i,j} = E\left[\int_t^{S(t,i)^{\{j\}}} \left(f(X(t')^{\{j\}}) - \eta\right) dt\right], \text{ for } x_i, x_j \in X.$$

Each realization factor measures the difference of being in state x_j instead of x_i on the performance measure of the system. In the terminology of perturbation analysis, if the process at t is perturbed from being in state x_i to state x_j, then at $t' = t + S(t,i)^{\{j\}}$ the perturbation is *realized* by the process. That is, at $t' \geq t$, the perturbed trajectory makes it back to state x_i, and the process continues the same as before it was perturbed.

6.1.1.5 Performance Potential. Shown in [90], we have

$$d_{i,j} = d_{i,k} + d_{k,j}, \text{ for } x_i, x_j, x_k \in X,$$

which, they point out, is a property similar to that of potential energy in physics.

A quantity g_i, called the performance potential of state x_i, can be defined by choosing a base state $x_k \in X$ and a real number c such that

$$g_k = c \text{ and } g_i = g_k + d_{k,i}.$$

They prove that,

$$d_{i,j} = g_j - g_i,$$

or that each realization factor is the difference of two potentials. The vector of state performance potentials $\mathbf{g} = (g_1, g_2, \ldots, g_n)$ is called the *potential vector.* A potential can be thought of as the expected cost/reward over a set period of time having started from each state. While the performance function gives the cost/reward per unit time of being in each state, the potentials go further. They include the expected performance gains/losses that are reachable from the state as well. Thus, some states might be assigned zero value from the performance function, but they still have potential because the system has a specific probability of transitioning *to* a cost/reward state *from* that state.

The realization matrix and the potential vector give two different quantities by which we can calculate the derivative of the performance measure. Proven in [90],

$$\frac{\partial \eta}{\partial \Delta \mathbf{Q}_X} = \boldsymbol{\pi} \Delta \mathbf{Q}_X^T \mathbf{D}_X \boldsymbol{\pi}^T = \boldsymbol{\pi} \Delta \mathbf{Q}_X \mathbf{g}. \tag{6.3}$$

6.1.1.6 Algorithms for Single Trajectory. The perturbation matrix $\Delta\mathbf{Q}_X$ of Equation (6.3) is supplied by the user, but the other quantities must be calculated to determine $\frac{\partial\eta}{\partial\Delta\mathbf{Q}_X}$. Algorithms for computing $\boldsymbol{\pi}$ and $\mathbf{g}$ based on a single trajectory, summarized below, are provided in [91].

Let T_k be the kth transition epoch of X (the time of the kth transition), S_k be its kth sojourn time (the time it remains in the kth state), and X_k be its state after the kth transition. The indicator function $I(X_k)$ is 1 if $X_k = x_i$ for state x_i and 0 otherwise. Then the steady-state probability π_i and potential g_i of each state x_i can be calculated from a single trajectory of $N \to \infty$ transitions, with probability one, as:

$$\pi_i = \lim_{N\to\infty} \frac{1}{T_N} \left\{ \sum_{k=0}^{N-1} I(X_k) S_k \right\} \text{ and} \tag{6.4}$$

$$g_i = \lim_{N\to\infty} \frac{\sum_{k=0}^{N} \left\{ I(X_k) \int_{T_k}^{T_k+T} f(X(t))dt \right\}}{\sum_{k=0}^{N} I(X_k)}. \tag{6.5}$$

Equation (6.4) sums the amount of time spent in a state and divides by the total running time of the process, giving the steady-state probability of that state, which becomes the expected value within the limit. Equation (6.5) averages the realization factor of a state across the trajectory, giving the potential for that state within the limit. The parameter T is a tunable parameter controlling the amount of time used to estimate the realization factors. We found that our heuristic of setting T equal to the longest sojourn time in the trajectory gave consistently accurate results across the different networks and trajectories. The benefit of using Equation (6.3) is that once $\boldsymbol{\pi}$ and $\mathbf{g}$ are computed from a single trajectory, $\frac{\partial\eta}{\partial\Delta\mathbf{Q}_X}$ can be calculated for any number of user-defined perturbation matrices.

Note that closed-form solutions exist for calculating $\boldsymbol{\pi}$ without having to estimate it from the trajectory using Equation (6.4). Furthermore, we found that the accuracy of the $\frac{\partial \eta}{\partial \Delta \mathbf{Q}_X}$ estimates were greatly improved, especially for large perturbations, when using the steady-state probabilities of the *perturbed* intensity matrix rather than the original intensity matrix. The closed-form solution for the perturbed steady-state probabilities is found by calculating the normalized left eigenvector of $\mathbf{Q}'_X$ corresponding to the zero eigenvalue [24],

$$\boldsymbol{\pi}\mathbf{Q}'_X = \mathbf{0}. \tag{6.6}$$

6.1.2 Other Methods for Sensitivity Analysis

In addition to perturbation realization, there are two other approaches that have been developed for performing sensitivity analysis on Markov processes, namely, the likelihood ratio method and the reduced augmented chain method.

6.1.2.1 Likelihood Ratio Method. The likelihood ratio (LR) method [99, 100, 101] takes the ratio of the likelihood of a trajectory from the original process with the likelihood of one that incorporates small changes to the transition rates. After this ratio is simplified and differentiated, it can be estimated using the number of transitions between states and the state sojourn times, as taken from a trajectory. This estimate is then used with the performance function to estimate the expected performance measure derivative. The LR method faces an inherent trade-off between variance in the estimator and bias in the estimate of the steady-state probabilities depending on the length of the trajectory that the estimator is given. One benefit of using LR, however, is that the process yields confidence intervals on the performance measure derivatives without extra effort.

6.1.2.2 Reduced Augmented Chain Method. The reduced augmented chain (RAC) method [93, 89] also uses a single trajectory, but instead of simulating the nominal process by itself, it first creates a combined process of both the nominal and the perturbed process, representing a superposition of nominal and perturbed states. Depending on the number of states that have been perturbed, this could represent a substantial increase in the number of states that must be simulated. On the other hand, the method does not rely on knowing the intensity matrix values, working instead with direct observation of the system, and can be used for on-line estimation of the steady-state probability sensitivities. This method allows sensitivity analysis to be performed on the actual system itself without first constructing a model of the system on which to perform sensitivity analysis.

There is another important feature trade-off between the RAC method and perturbation realization. For perturbation realization, multiple perturbations can be tested for a given trajectory while the performance function remains fixed (otherwise the performance potentials would need to be re-estimated). With the RAC method, the performance function can be varied for a given reduced augmented chain. After varying the parameter perturbations, however, a new reduced augmented chain must be created and sampled.

Note that LR and RAC could also be incorporated into our approach to CTBN sensitivity analysis. Even so, we chose perturbation realization for use in our experiments. Our goal is efficient sensitivity analysis of CTBNs, and the ability of perturbation realization to calculate and retain potentials for different subnetworks is directly applicable to that end. LR does not compute potentials that can be reused between queries, while that is one of the main features of perturbation realization. RAC actually increases the size of the state-space, which is counter-productive when trying to deal with the already-exponential size of the state-space for CTBNs.

6.2 Perturbation Realization on CTBNs

As the CTBN is a generative model as shown in Section 2.3.3, the sample-based method of perturbation realization for Markov processes is a natural candidate for CTBN sensitivity analysis. A straightforward application of perturbation realization to CTBNs would result in simply amalgamating the entire network into one large Markov process, as described in Section 2.3.4. But this fails to take advantage of the factored nature of CTBNs, which attempts to reduce complex state-spaces into more compact representations that model conditional dependencies instead of the full joint distributions. The naïve approach to sensitivity analysis requires working with the full joint intensity matrix, ignoring the very reason for having the factored representation in the first place. Generating trajectories becomes exponentially expensive, and perturbation realization becomes infeasible. The factored nature of the networks enables sensitivity analysis to work on smaller subnetworks. These subnetworks can be sampled, the performance potentials calculated, and multiple perturbation matrices tested—all separately.

For a CTBN sensitivity analysis example, consider the network shown in Figure 5.3 as a reliability model. Suppose that a performance function is attached to the two states of the E node, which denote the failed and non-failed states of the system. Instead of amalgamating all of the nodes into one large Markov process, we would like to divide the system into smaller subnetworks. Smaller subnetworks limit the state-space size of any one subnetwork and allow for more tractable evaluation of Equation 6.3 for calculating the change in performance for given perturbations.

6.2.1 Sufficient Conditions for CTBN Ergodicity

Another issue to address when dividing the network into different subnetworks is ensuring ergodicity, required by the algorithms for perturbation realization. The CTBN, although in factored form, still represents a single process. Therefore, we would like to show that if each of the nodes (as a subprocess) is ergodic, then the process represented by larger sets of nodes in the CTBN (including the network as a whole) is also ergodic. If this is the case, then perturbation realization can be applied to trajectories generated from subsets of nodes as well.

Suppose we have two nodes with the most general case that both $A \rightarrow B$ and $A \leftarrow B$. If $|A|$ denotes the number of states of A, then the number of states in the amalgamated supernode AB is $|A||B|$. Assume that A and B are each ergodic. In other words, A and B are each irreducible and positive recurrent, defined formally as follows.

Definition 6.2.1 (Irreducible)**.** *Let x_i and x_j be any two states of X. X is irreducible if, for all t, there exist $t' > t$ such that*

$$P(X(t') = x_i|X(t) = x_j) > 0 \textit{ and } P(X(t') = x_j|X(t) = x_i) > 0.$$

That is, there is a non-zero probability of transitioning from x_i to x_j and from x_j to x_i.

Definition 6.2.2 (Positive Recurrent). *A node X is positive recurrent if, for all $x \in X$,*

$$\int_0^\infty P(X(t) = x | X(0) = x, X((0,t)) \neq x) \, dt = 1$$

That is, the probability of starting in state x, transiting out of x, and then returning to state x after some finite amount of time is 1.

Let $a_{ij}|b_k$ denote the entry at i, j of the conditional intensity matrix $A|b_k$. (Note that if A or B have additional parents, the following procedure applies individually to each conditional intensity matrix conditioned on the parents' states.) Each entry $(a_{ij}b_{kl})$ in the combined intensity matrix of A and B is calculated by the following:

$$(a_{ij}b_{kl}) = \begin{cases} a_{ij}|b_k & \text{if } i \neq j \text{ and } k = l \\ b_{kl}|a_i & \text{if } i = j \text{ and } k \neq l \\ a_{ij}|b_k + b_{kl}|a_i & \text{if } i = j \text{ and } k = l \\ 0 & \text{otherwise} \end{cases} \tag{6.7}$$

Lemma 6.2.1. *Let each conditional intensity matrix of A and B be irreducible. Then each conditional intensity matrix of the amalgamated supernode AB is irreducible.*

Proof. Because of the factorization of AB into A and B, the state of AB cannot change in both A and B simultaneously, represented by the zero value of the 4th case. We note that if both A and B are irreducible, then AB is also irreducible, as every state is reachable in at most 2 steps. The worst case would require a state change in both A and B individually (the state of A changes, then the state of B changes, or vice-versa), which is possible because of the non-zero transition values of the 1st and 2nd cases. In other words, for any state a_ibk, we have a non-zero probability of transitioning to (a_jb_l), because $(a_{ij}b_k)$ and (a_jb_{kl}) are non-zero and

$(a_i b_{kl})$ and $(a_{ij} b_l)$ are non-zero. In other words, there exists $t' > t$ such that

$$P(AB(t') = (a_j b_l)|AB(t) = (a_i b_k)) > 0 \text{ and } P(AB(t') = (a_i b_k)|AB(t) = (a_j b_l)) > 0.$$

□

Lemma 6.2.2. *Let each conditional intensity matrix of A and B be positive recurrent. Then each conditional intensity matrix of the amalgamated supernode AB is positive recurrent.*

Proof. Because A is ergodic, there is a non-zero probability of transitioning to any of its states during that time.

$$\int_0^\infty P(A(t) = a_i|A(0) = a_i, A((0,t)) \neq a_i)\, dt = 1.$$

Similarly, because B is ergodic, there is a non-zero probability of transitioning to any of its states during that time.

$$\int_0^\infty P(B(t) = b_k|B(0) = b_k, B((0,t)) \neq b_k)\, dt = 1.$$

Now consider the conditional intensity matrices of AB. The negative values along the diagonals, controlling state sojourn times, only add with other diagonals as shown in the 3rd case of Equation 6.7, meaning that these values will never go to zero, which would result in an absorbing state. There is a non-zero probability of transitioning from $(a_i b_k)$ to $(a_j b_k)$ and from $(a_i b_k)$ to $(a_i b_l)$ for all states in $a_i, a_j \in A$ and $b_k, b_l \in B$. This implies that

$$\int_0^\infty P(AB(t) = (a_i b_k)|AB(0) = (a_i b_k), AB((0,t)) \neq (a_i b_k))\, dt = 1$$

Therefore, each conditional intensity matrix of the amalgamated supernode AB is positive recurrent. □

Theorem 6.2.3. *If each conditional intensity matrix of a CTBN is ergodic, then the conditional intensity matrices from the amalgamation of any sets of nodes is also ergodic.*

Proof. From lemma 6.2.1 and lemma 6.2.2, amalgamation of two nodes with ergodic conditional intensity matrices will produce another node with ergodic conditional intensity matrices. Therefore, this process can be repeated to amalgamated any number of nodes in the CTBN and the resulting nodes will still have ergodic conditional intensity matrices. □

6.2.2 Estimating CTBN Performance Derivative

Using perturbation realization and the method for isolating subnetworks, we can describe a generalized method for performing sensitivity analysis on CTBNs. The primary obstacle to overcome is that the state-space size is exponential in the number and cardinality of the CTBN nodes. For complex CTBNs, even relatively long trajectories never reach all possible states. Because of these unvisited states, perturbation realization has no information on the state's performance potential and the value returned from Equation (6.3) becomes inaccurate. Thus, being able to create and analyze trajectories from smaller subnetworks, and then being able to combine the results, becomes essential.

Algorithm 6.5 shows the pseudocode for estimating the performance derivative. The algorithm accepts a CTBN and a set of perturbation matrices for the nodes of the CTBN that represent changes to the nodes' intensity matrices. The algorithm returns an estimate of the change in performance per unit time given the perturbations in

Algorithm 6.5 CTBN sensitivity analysis method for estimating performance measure derivative.

```
EstimateDerivative($\mathcal{N}$, $\Delta\mathbf{Q}_{\mathcal{N}}$)
1:  $\mathcal{G}' \leftarrow$ CollapseCycles($\mathcal{G}$)
2:  $\partial\eta/\partial\Delta\mathbf{Q}_{\mathcal{N}} \leftarrow 0$
3:  repeat until termination
4:      $\mathbf{X} \leftarrow \bigcup_{X\in\mathcal{G}'} X$ where $\mathbf{Pa}(X) = null$
5:      if $\mathbf{X} = null$ then terminate
6:      for $X \in \mathbf{X}$
7:          $\mathbf{Q}_X \leftarrow \mathbf{Q}_X + \Delta\mathbf{Q}_X$
8:      end for
9:      $\mathbf{Y} \leftarrow \bigcup_{X\in\mathbf{X}} \mathbf{Ch}(X)$ where $\mathbf{Pa}(\mathbf{Ch}(X)) \subseteq \mathbf{X}$
10:     for $Y \in \mathbf{Y}$
11:         $\sigma \leftarrow$ Sample($Y \cup \mathrm{Pa}(Y)$) // Algorithm 2.1
12:         $Y' \leftarrow$ IsolateNode($\sigma, Y$) // Algorithm 5.3
13:         $\mathbf{Q}_{Y|\mathbf{Pa}(Y)} \leftarrow \mathbf{Q}_{Y|\mathbf{Pa}(Y)} + \Delta\mathbf{Q}_{Y|\mathbf{Pa}(Y)}$
14:         $\sigma' \leftarrow$ Sample($Y \cup \mathrm{Pa}(Y)$)
15:         $Y'' \leftarrow$ IsolateNode($\sigma', Y$)
16:         $\Delta\mathbf{Q}_Y \leftarrow \Delta\mathbf{Q}_Y + (\mathbf{Q}_{Y''} - \mathbf{Q}_{Y'})$
17:         $\mathbf{Q}_Y \leftarrow$ Amalgamate($Y \cup \mathbf{Pa}(Y)$) // Equation 6.7
18:         $\boldsymbol{\pi} \leftarrow$ CalculateSteadyStateProbabilities($\mathbf{Q}_Y$) // Equation 6.6
19:         $\Delta\mathbf{Q}_{Y\cup\mathbf{Pa}(Y)} \leftarrow$ Amalgamate($\Delta\mathbf{Q}_Y \cup \Delta\mathbf{Q}_{\mathbf{Pa}(Y)}$)
20:         $\mathbf{g} \leftarrow$ CalculatePotentialVector($\sigma'$) // Equation 6.5
21:         $\partial\eta/\partial\Delta\mathbf{Q}_{\mathcal{N}} \leftarrow \partial\eta/\partial\Delta\mathbf{Q}_{\mathcal{N}} + \boldsymbol{\pi}\Delta\mathbf{Q}_{Y\cup\mathbf{Pa}(Y)}\mathbf{g}$
22:         for $X \in \mathbf{Pa}(Y)$
23:             remove edge $(X, Y)$ in $\mathcal{G}'$
24:         end for
25:         replace $Y$ with $Y'$ in $\mathcal{G}'$
26:     end for
27: end repeat
28: return $\partial\eta/\partial\Delta\mathbf{Q}_{\mathcal{N}}$
```

the intensity matrices. It calls four helper methods in addition to Algorithm 2.1 and the node isolation of Chapter 5. CollapseCycles($\mathcal{G}$) detects cycles in $\mathcal{G}$, amalgamates each cycle's intensity matrices, and replaces each cycle with a single node. Amalgamate($Y\cup\mathbf{Pa}(Y)$) expands the full joint intensity matrix of the CTBN subnetwork $Y\cup\mathbf{Pa}(Y)$ by repeatedly combining pairs of nodes as described in Section 2.3.4.

CalculateSteadyStateProbabilities($\mathbf{Q}_X$) calculates the steady-state probabilities $\boldsymbol{\pi}$ of the intensity matrix $\mathbf{Q}_X$ by calculating the normalized eigenvector corresponding to the zero eigenvalue as per Equation (6.6). CalculatePotentialVector(σ) parses the trajectory σ and calculates the potential vector $\mathbf{g}$ as per Equation (6.5).

First, line 1 collapses the cycles of the CTBN. This is necessary before the top-down isolation begins, because node isolation requires a trajectory with all of the node's ancestors, and every node in a cycle is an ancestor to every other node in the cycle. This turns the network into a directed acyclic graph. Line 2 initializes the performance measure derivative estimate, which will be updated incrementally during the top-down isolation process. Lines 3-27 repeat until the condition of line 5 is satisfied, that is, when every node has been isolated. Line 4 finds the roots of the network, those without any parents. Lines 6-8 apply any perturbations to those root nodes. Line 9 finds all the immediate children of root nodes, i.e., all the second-level nodes. These will be isolated and become the new root nodes. Lines 10-26 iterate over the children. Line 11 creates a trajectory for the child node, while line 12 uses that trajectory to isolate it. Line 13 applies any perturbations to the child's conditional intensity matrices, and lines 14-15 isolate the perturbed child node. Line 16 adds the difference between the unconditional intensity matrices to the child perturbations. Lines 17-18 calculate the steady-state probabilities of the child node and its parents. Line 19 calculates the perturbation matrix for the child node and its parents. Line 20 calculates the potential vector from the perturbed trajectory. Line 21 calculates the performance measure derivative for the subnetwork consisting of the child and its parents. Lines 22-25 finish the isolation of the child, removing the arcs from its parents and replacing the conditional intensity matrices with the single unconditional intensity matrix calculated in line 12. After reaching the leaves of the CTBN, the aggregated performance measure derivative is returned in line 28.

Algorithm 6.5 avoids handling the whole CTBN at once. Trajectories only have to be created for a node and its parents. Thus, the complexity of the algorithm is driven by the size of the cycles that have to be collapsed into single nodes and the number of parents of each node (reminiscent of tree width in clique tree inference on Bayesian networks). However, the algorithm is able to take advantage of the network factorization and only deal with smaller subnetworks at any one time.

6.3 CTBN Reliability Measures

While our approach for sensitivity analysis applies to general CTBNs and general performance functions [102], we now present how CTBNs can model reliability measures such that our algorithm can be used to perform sensitivity analysis with respect to those measures.

6.3.1 Mean Time Between Failures

The measure for MTBF does not actually rely on the performance function, but is a value derived from the network parameters themselves. The MTBF can be computed as a direct result of node isolation. Isolation of the performance nodes yields their unconditional intensity matrices. The diagonal parameters of these unconditional intensity matrices give the expected sojourn times in the failed and non-failed states, from which can be derived the MTBF. Take a two-state node in which the non-failed state is 0 and the failed state is 1. After finding the node's single unconditional intensity matrix, the MTBF can be estimated as:

$$\frac{1}{|q_{0,0}|}.$$

In fact, our algorithm for sensitivity analysis provides the MTBF as a direct result of applying our node isolation method of Section 5.2. Perturbations applied to ancestors in the network are carried down in the node isolation in parallel with the node isolation of the original network. Thus, the algorithm gives the MTBF for the original network and the new MTBF as a result of the perturbations. The change in MTBF is the difference between the MTBF results of these two unconditional intensity matrices.

6.3.2 Point Availability

Point availability is the proportion of time that a system is in a functioning condition. We can represent this reliability measure in the CTBN using our concept of performance functions from Section 4.2. In this case, a unit cost is associated with the network's failed state. As the performance function represents cost per unit time, a performance function value with unit cost gives the proportion of time that the system remains in the failed state, from which we can derive the point availability of the system. Take an n-state node in which the failed state is $n-1$. The performance function is:

$$f(i) = \begin{cases} 1 & i = n-1 \\ 0 & \text{otherwise} \end{cases}.$$

The performance measure η is the proportion of time that the system is in the failed state. The point availabity is therefore:

$$1 - \eta.$$

Sensitivity analysis allows us to measure how perturbations in the ancestors will be expected to impact availability. Suppose that the failure rates of different individual components in a complex system are perturbed. How does this affect the point availability and the MTBF of the overall system?

6.4 Experiments

We demonstrate the algorithms for CTBN sensitivity analysis on three networks. The first is a small, synthetic network in which the exact answers can be computed easily. The second is the DFT for the cardiac assist system encoded as a CTBN. The third is a model of a milling machine learned from run-to-failure data.

For each network in our experiments, the point availability and MTBF are calculated using our algorithms for sensitivity analysis and compared to the ground truth, taken as either the exact solutions for the synthetic network or estimated from brute-force sampling over the whole network for the two real-world networks. We measure relative error between our algorithm and the ground truth, as well as measure complexity by comparing state-space sizes of different subnetworks and total number of samples generated.

6.4.1 Simple Network

This is the example network shown in Figure 5.3. The initial distributions and intensity matrices for all the nodes can be found in Appendix D. This network is simple enough that the exact solutions can be computed easily using Equation 6.1.

Node E has two states, representing failed and non-failed states. Suppose that we want to observe how the parameters of C affect system reliability. The naïve approach would be to amalgamate the entire CTBN into a single Markov process and apply perturbation realization directly. For this simple network, this is possible, but ignores the advantages of having the factored representation. Using the approach presented here, we can perform top-down isolation of the levels in the network and avoid ever having to deal with the entire network all at once.

Let the conditional intensity matrix for D given c_0 and the corresponding perturbation matrix be as follows.

$$\mathbf{Q}_{D|c_0} = \begin{pmatrix} -1 & 1 \\ 2 & -2 \end{pmatrix} \qquad \Delta\mathbf{Q}_{D|c_0} = \begin{pmatrix} 0.9 & -0.9 \\ 10 & -10 \end{pmatrix}$$

We test this perturbation on the MTBF and the point availability. These are relatively extreme perturbations for this conditional intensity matrix, changing the expected sojourn time in d_0 to be 10 times longer and the expected sojourn time in d_1 to be 6 times shorter whenever C is in state c_0. Note that perturbations that represent uncertainty in the parameters will most likely not be this severe. However, the performance measure derivative becomes harder to estimate as the magnitude of the perturbations increases. Therefore, by choosing these large perturbations, we show that the method is still able to maintain accuracy even for extreme cases.

Before the perturbations are applied, the closed-form solution yields an availability of 94.30%, and a MTBF of 16.55 time units. After the perturbations are applied, the closed-form solution shows that the availability increases to 96.02% and the MTBF increases to 24.10 time units.

Now we apply our sensitivity analysis algorithm to the network, performing top-down isolation of each level and cascading the perturbations down to each unconditional intensity matrix. For each isolation, we generate trajectories of 100,000 tran-

Table 6.1: Perturbed reliability estimates of simple network.

	Availability	MTBF	Sample Count
Exact	96.02%	24.10 time units	N/A
Brute-Force	96.05%	24.26 time units	12.3 million
Node Isolation	96.04%	24.31 time units	2.6 million

sitions in the node to isolate. Applying the perturbations, our sensitivity analysis algorithm estimates the point availability at 96.04% and the MTBF at 24.31 time units. In both cases, this is less than 1% relative error. If required, the trajectories can be lengthened to further improve the accuracy.

Although this network is small enough that we can work with the full joint intensity matrix directly, we also compare our sensitivity analysis approach with the brute-force approach over the whole network. When we sampled over the whole network all at once, it took almost 12.3 million transitions to be generated before we reached 100,000 transitions in node E. By isolating each level separately, we reached 100,000 transitions in all of the isolated nodes with around 2.6 million total samples generated across all levels. Our sensitivity analysis algorithm restricted the largest state-space size of any subnetwork to only 18 states (the A-B-C subnetwork). The brute-force approach was 4 times as large, sampling from the whole network with 72 states. The results for this network are summarized in Table 6.1.

For this simple network, the reliability measures could be computed exactly. However, we note that as the networks become more complex, some form of approximate solution, such as the brute-force or node isolation sampling methods, is required to keep inference tractable. For this network, we showed that node isolation can be more efficient than the brute-force approach without a loss in accuracy. With node isolation, sampling can be targeted to specific subnetworks. The brute-force approach is more susceptible to over-sampling and under-sampling different parts of the network.

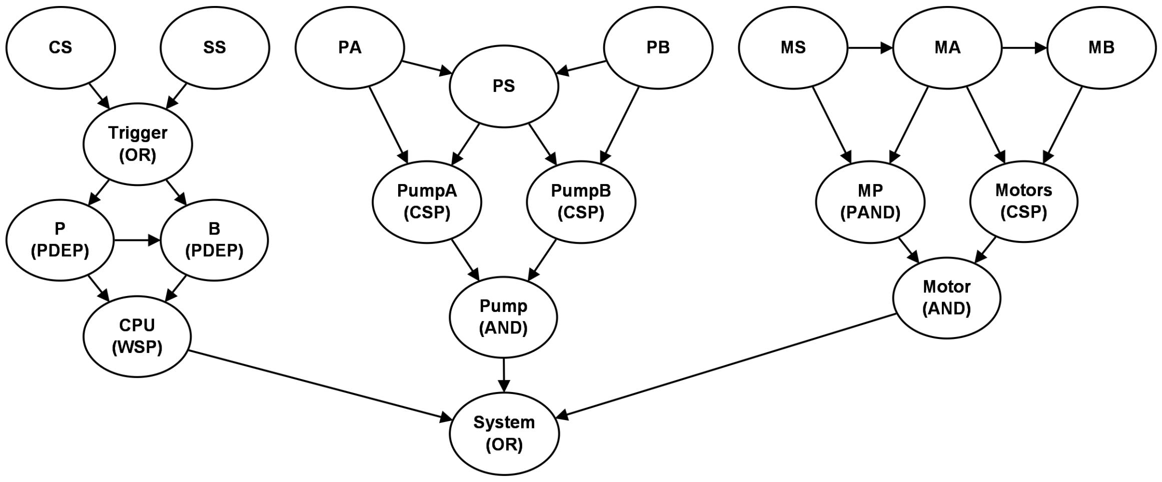


Figure 6.1: Cardiac assist system model.

<u>6.4.2 Cardiac Assist System</u>

As mentioned in Section 5.3.4, specially constructed CTBNs have been used to represent DFTs specifically for reliability modeling, in which each of the gates of a DFT can be mapped onto CTBN nodes. In this experiment, we use the cardiac assist system (CAS) model as given in [33]. The network is shown in Figure 6.1. The initial distributions and intensity matrices for all the nodes can be found in Appendix E. Of the various repair policies evaluated in [33], we use the repair rate of $\mu = 0.1$ for all components.

The network is divided into three subsystems: the CPU, pump, and motor. Here, we can see the advantage of the CTBN's factored representation by how it can model the different subsystems as different subnetworks. In fact, as a starting point for reliability analysis, we can look at the reliability of each of the subsystems separately using our node isolation technique. We divide the network into the three subsystems and find the unconditional intensity matrices of CPU, Pump, and Motor to get the MTBF of each. The MTBF estimates are shown in Table 6.2.

Table 6.2: MTBF estimates of CAS subsystems.

CPU	154 hours
Pump	4.1×10^6 hours
Motor	4.2×10^8 hours

From the table, we see that the CPU subsystems contributes most to the unreliability of the system. This is not surprising, because the pump and motor subsystems have multiple redundancies, while the CPU subsystem does not, requiring both the cross-switch (CS) and system supervision (SS) to be in the non-failed state. Now that we have identified the CPU subsystem as the most unreliable subsystem, we can run scenarios on how variations (such as bounds on the parameter's uncertainty) in the subsystem parameters affect the system as a whole.

Before we apply our sensitivity analysis technique, we use the brute-force approach of generating samples from the entire network. We generate samples over the entire network until we have one million instances of failure of the entire system. From these samples, we estimate the MTBF to be 163 hours and the point availability to be 94.3%. Now suppose that the best-case failure rates of both CS and SS changed the parameters from $\lambda = 2 \times 10^{-4}$ to $\lambda = 1 \times 10^{-6}$. Applying these changes and generating samples over the entire network again until we have one million instances of failure of the entire system, we estimate the new MTBF to be 176 hours and the point availability to be 94.8%.

Using our sensitivity analysis technique, we can isolate the three nodes for the different subsystems: CPU, Pump, and Motor. We estimate the unconditional intensity matrix of CPU with and without the perturbations to CS and SS. The difference between the two becomes the perturbations to CPU. We then perform perturbation realization on just these four lowest nodes. These results also estimate the MTBF to be 176 hours and the point availability to be 94.8%, with less than 1% relative error in both cases.

The brute-force approach is sampling from a state-space size of over 6.6 million states over the whole network. When isolating individual subsystems, the state-space sizes are 144, 144, and 160 states for the CPU, pump, and motor subsystems, respectively. Once isolated, the three subsystems are combined into the lower four-node subnetwork of only 16 states.

We note that, in this case, the sensitivity analysis approach had to generate substantially more samples than the brute-force approach, specifically in isolating the pump and motor subsystems. This is explained by the large difference in reliability between those subsystems and the CPU subsystem. Because of the rarer occurrence of failure in these subsystems, it required more samples before one million transitions in each of Pump and Motor were collected. However, this also suggests that the brute-force approach was over-sampling the CPU subnetwork in order to get its estimate. Thus, node isolation offers a way to ensure that different subnetworks are sampled adequately.

Furthermore, saving the potentials and the unconditional intensity matrices of the three subsystem nodes, we could apply different perturbations to different nodes throughout the network, compute the perturbations on the unconditional intensity matrices of the three subsystem nodes and update the reliability estimates without ever having to generate samples from the entire network all at once.

6.4.3 Milling Machine

For this experiment, the CTBN was learned from run-to-failure data. The data were collected from 16 cases of a milling machine under various operating conditions

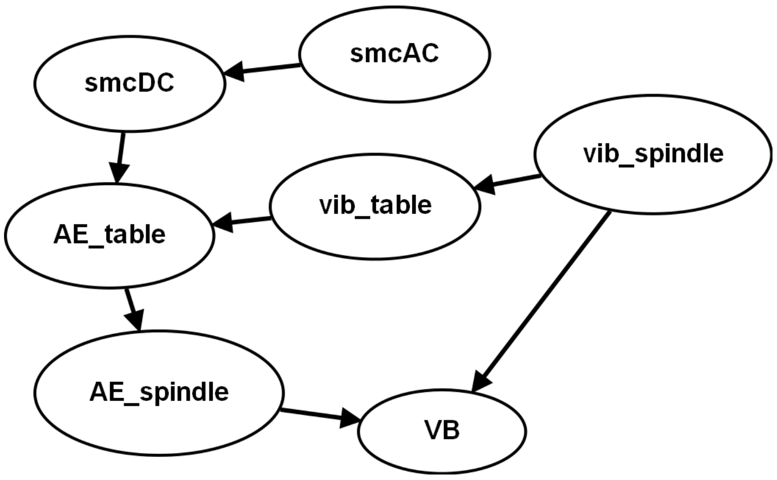


Figure 6.2: Milling machine model.

measuring the flank wear of the cutting tool [103]. The data consisted of over 1.5 million timestamped records. Intervals for equal-frequency discretization were computed from 100,000 samples drawn uniformly at random from these records. All variables were discretized into 3 bins, except for the flank wear, which was discretized into 2 bins, representing failed and non-failed states. All records were then discretized, and the CTBN structure was learned using the Continuous Time Bayesian Network Reasoning and Learning Engine (CTBN-RLE) [104]. The edges were pruned and oriented to balance network complexity and place flank wear (VB) as a descendant of the other variables. The network is shown in Figure 6.2, and the node names are given in Table 6.3. The initial distributions and intensity matrices for all the nodes can be found in Appendix F. Parameter estimation was then performed, also using the CTBN-RLE. Each of the 16 cases described one run-to-failure of the cutting tool but did not include tool replacement. To make the model ergodic, a repair rate of $\mu = 0.1$ was added to the conditional intensity matrices of VB.

First, we perform reliability analysis without any perturbations. From the brute-force approach generating 100,000 transitions in VB using the entire network, we

Table 6.3: Milling machine node names.

Abbreviation	Name
smcAC	AC spindle motor current
smcDC	DC spindle motor current
vib_spindle	spindle vibration
vib_table	table vibration
AE_table	acoustic emission at table
AE_spindle	acoustic emission at spindle
VB	flank wear

have an estimated MTBF of 44,047 time units and an estimated point availability of 99.977%. Now suppose we apply the perturbations to the intensity matrix of smcAC, given as follows.

$$\mathbf{Q}_{\text{smcAC}} = \begin{pmatrix} -0.134 & 0.132 & 0.002 \\ 0.134 & -0.267 & 0.133 \\ 0.002 & 0.131 & -0.133 \end{pmatrix} \qquad \Delta\mathbf{Q}_{\text{smcAC}} = \begin{pmatrix} -1.0 & 0.5 & 0.5 \\ 0.5 & -1.0 & 0.5 \\ 0.5 & 0.5 & -1.0 \end{pmatrix}$$

These perturbations speed up the transitions between the states of smcAC by almost a factor of 10. Using the brute-force approach, the new estimated MTBF changes to 40,873 time units while the estimated point availability hardly changes, at 99.976%.

Using our sensitivity analysis technique, we can divide the network in half by first isolating AE_table with and without the perturbations applied to smcAC. Running perturbation realization on the lower half of the network, we have an estimated MTBF of 40,744 time units and an estimated point availability of 99.976%, which is less than 1% relative error in both cases.

Again, perturbations in the upper half of the network can be included in the perturbations of the unconditional intensity matrix of AE_table, and the potentials can be re-used. By dividing the network into upper and lower halves, the size of the

Table 6.4: Summary of sensitivity analysis results.

Network	Node count	State count	State count of largest subnetwork	Magnitude of perturbation	Sample count	Rel. error
Synthetic	5	72	18	$\times 10$	100K	$< 1\%$
CAS	19	6.6M	160	$\times 200$	1M	$< 1\%$
Milling	7	1458	81	$\times 10$	100K	$< 1\%$

state-space for sampling is 81 and 54 states, respectively, instead of 1458 states when sampling over the entire network.

The brute-force approach had to generate close to two billion samples to get 100,000 transitions in VB. The sensitivity analysis approach generated less than 360,000 samples to isolate AE_table and then also around two billion samples to get 100,000 transitions in VB. While the total number of samples generated by the two approaches is similar, the advantage of using perturbation realization is to calculate potentials and be able to plug in any number of perturbation matrices without resampling. However, by first dividing the network into upper and lower halves, the sizes of the perturbation matrices and system of equations for the steady-state probabilities are reduced from 1458×1458 to 54×54. Any perturbations in the upper network can be included in the perturbation matrix for AE_table after generating a relatively small numbers of samples. Thus, our method for sensitivity analysis in CTBNs combines the advantages of both perturbation realization and the factored nature of the networks to manage complexity. The results for all three networks are summarized Table 6.4.

6.5 Conclusion

We have demonstrated how sensitivity analysis can take advantage of the factored nature of CTBN models to perform efficiently. Specifically, we have devised a method that is able to exploit the conditional independence of the CTBN to analyze subnetworks independently. For large, complex networks, the node isolation method can be used to counteract the exponential blow-up in the size of the state-space. Node isolation also provides a mechanism for more selectively sampling different parts of the network so that different subnetworks are neither over-sampled nor under-sampled. Furthermore, this is the first time a method for sensitivity analysis has been developed for CTBNs.

As noted earlier, the node isolation is an approximation method that approximates the behavior of multiple nodes with a single node. The performance potentials are also approximated by a trajectory of the system. The relationship between the length of the trajectories (computational complexity) and the accuracy of the estimates, both of the performance measure derivatives and the unconditional intensity matrices, are areas for future work.

CHAPTER 7

CONTINUOUS-TIME EVIDENCE

In this chapter, we define both uncertain and negative types of continuous-time evidence and show how to support these definitions in the context of CTBN inference.

Probabilistic models, such as the BN and CTBN, provide a mathematically rigorous framework for reasoning under uncertainty. Given observations about the system the network represents, the network can update the posterior probabilities of other states in light of that evidence. However, the representation of uncertainty in BNs has been extended further to allow for uncertainty in the evidence as well, such as with soft evidence or virtual evidence, in which there is uncertainty in the observations themselves. Uncertain evidence provides a generalization of evidence, in which the evidence also has degrees of uncertainty associated with it.

As a relatively new model, the CTBN currently only supports "crisp" evidence, in which all of the temporal state evidence is trusted with complete confidence. However, in many applications, such as fault prognosis in which we rely on instrument measurements, the evidence may contain noise or errors and can be trusted only to a certain degree. Uncertain evidence for CTBNs has not yet been defined, and algorithms have not been extended to allow for incorporating the uncertainty of temporal evidence. The inclusion of continuous time into the evidence gives rise to subtleties in negative evidence, as well.

7.1 Background Work

This section briefly describes existing techniques for handling uncertain and negative evidence in the context of BNs and describes why these do not carry over to the CTBN. In BNs, uncertain evidence is when the state of a node is only known with a given probability. Negative evidence is when a node is known to not be in a particular state.

One approach for uncertain evidence in BNs is called virtual evidence, as described in [105]. To represent this type of uncertain evidence, a node is added as a child to an observation node. This child node then becomes the node that is observed, and the conditional probabilities of this child are set to match the strength of the evidence. In effect, virtual evidence sets up a ratio of likelihoods to represent the confidence of an observation correctly observing a particular state. Another approach is soft evidence, as differentiated from virtual evidence in [106, 107]. Soft evidence changes the marginal distribution of the evidence itself, using a generalization of conditioning on observed variables to condition on an observation of a probability distribution, in which the observed distribution holds the uncertainty of the observations.

In this section we describe the types of evidence currently used in inference with BNs, which include certain, uncertain, and negative evidence. We then review the types of evidence currently defined for CTBNs.

7.1.1 Evidence in Bayesian Networks

We start by reviewing the types of evidence used in BNs. We distinguish between certain/uncertain types and positive/negative types.

7.1.1.1 Certain Positive Evidence. Traditional evidence in a BN falls under certain evidence. That is, the observations are trusted with complete confidence, and the observations are of specific states. Suppose that we had a set of observations $\mathbf{e}$. To perform inference with this evidence, we would compute the posterior probabilities $P(\mathbf{X}|\mathbf{e})$. Once we observe the state of a variable, the probability of that state for that variable becomes 1. From this, we can generalize to uncertain evidence, in which the probabilities of observed states can become less than 1.

7.1.1.2 Uncertain Positive Evidence. One approach for uncertain evidence in BNs is called virtual evidence [105]. To simulate this type of uncertain evidence, a node is added as a child to an observed node, as shown in Figure 7.1. This child node then becomes the node that is observed, and the conditional probabilities of this child are set to match the strength of the evidence. In effect, virtual evidence sets up a ratio of likelihoods to represent the the confidence of an observation observing a particular state.

Suppose the uncertain evidence for node X is given as η. Then for $x \in X$ we set

$$P(\eta|X = x) = \lambda_x, \tag{7.1}$$

where λ_x denotes the likelihood of being in state x.

Let α be an event in the BN but not in X. Because the added child node is only dependent on X, η and α are conditionally independent given X. Therefore, for $x \in X$,

$$P(\eta|X = x, \alpha) = P(\eta|X = x). \tag{7.2}$$

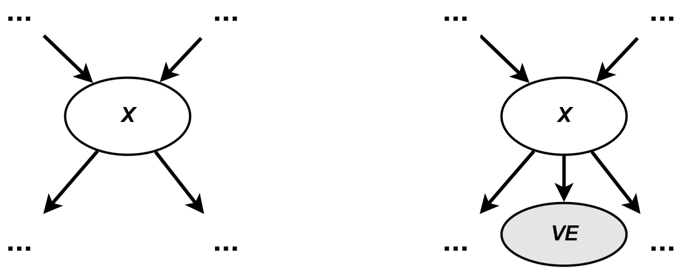


Figure 7.1: Example of virtual evidence for node X.

Then the probability of event α given the uncertain evidence η is

$$P(\alpha|\eta) = \frac{\sum\limits_{y \in X} \lambda_y P(\alpha, X = y)}{\sum\limits_{z \in X} \lambda_z P(X = z)}. \tag{7.3}$$

By so doing, virtual evidence weights the marginal probabilities $P(X)$ by the likelihood of the evidence λ_x for $x \in X$.

Another approach for representing uncertainty in an observation is through what is called soft evidence [106, 107]. Soft evidence uses Jeffrey's rule as a generalization of conditioning on observed variables to condition on an observation of a probability distribution, in which the observed distribution holds the uncertainty of the observations. Suppose that, for $x \in X$, the evidence is specified by a set of probabilities

$$P'(X = x) = q_x. \tag{7.4}$$

Then the new posterior distribution for event α is calculated as

$$P'(\alpha) = \sum_{x \in X} q_x \frac{P(\alpha, X = x)}{P(X = x)}. \tag{7.5}$$

As shown by Equation 7.4, soft evidence actually transforms the marginal distribution of X from $P(X)$ to $P'(X)$ such that it conforms exactly to the probability of the evidence.

Because each approach specifies uncertain evidence differently, virtual evidence and soft evidence yield different probabilities given the same values for q_x and λ_x, but conversions exist to derive the resulting probability of one given the other [105].

While the name "continuous time Bayesian network" suggests that the model as a special type of Bayesian network, we have seen that the underlying model is quite different from that of a BN. Thus, the methods for including uncertain evidence in BNs do not translate over to CTBNs. No literature for uncertain evidence has been found in the context of continuous-time Markov processes or CTBNs. In these cases, the evidence is temporal, containing state information over a real-valued period of time, which prohibits the uncertain evidence techniques of BNs (even DBNs) from being applied directly.

7.1.1.3 Certain Negative Evidence. In a BN, certain negative evidence is just a special case of uncertain positive evidence. If we observe the variable to *not* be in some states, this is the same as uncertain evidence in which there is zero probability of those states and the remaining probabilities for all other states are re-normalized.

Let $A \subset X$ be a subset of states of X that can be ruled out. Negative evidence states that $P(x \in A) = 0$. We can use virtual evidence to support negative evidence in BNs by setting

$$\lambda_x = \begin{cases} 0 & \text{for } x \in A \\ c & \text{for } x \notin A \text{ and any constant } c \end{cases}. \tag{7.6}$$

It follows that for every event α given the virtual event η that the states of A can be ruled out with likelihoods λ_x given above,

$$P(\alpha|\eta) = \frac{\sum\limits_{y\in X} \lambda_y P(\alpha, X = y)}{\sum\limits_{z\in X} \lambda_z P(X = z)} = \frac{c\sum\limits_{y\notin A} P(\alpha, X = y)}{c\sum\limits_{z\notin A} P(X = z)} = \frac{P(\alpha, X \notin A)}{P(X \notin A)} = P(\alpha|X \notin A). \tag{7.7}$$

Thus when static, uncertain evidence is sufficient to represent negative evidence. However, we can differentiate between positive and negative evidence when the evidence becomes temporal. Temporal evidence introduces subtleties between uncertain evidence and negative evidence, because now evidence must be defined in terms of state *and* time.

7.1.2 Evidence in CTBNs

While static BNs use evidence as observations of states, evidence in CTBNs must include temporal information. Three types of evidence in CTBNs have been defined so far: point, transition, and interval.

7.1.2.1 Certain Point Evidence.

The first inference algorithms defined for CTBNs supported only point evidence [17]. Certain point evidence in a CTBN is defined formally as follows.

Definition 7.1.1 (Certain point evidence)**.** *Let t be an instantaneous point in real-valued time. Let X be a node in the CTBN, and let x be a state of X. Node X was observed to be in state x at time t.*

Point evidence would result when the system cannot be monitored continuously, and sensors can only "poll" the state at various instantaneous points in time.

7.1.2.2 Certain Transition Evidence. Unlike a static model, the states of the CTBN could be changing throughout time. When monitoring a system, we might be able to detect changes in the state of the system. We would like to be able to incorporate this transition information into our inference procedures. Certain transition evidence in a CTBN is defined formally as follows.

Definition 7.1.2 (Certain transition evidence)**.** *Let t be an instantaneous point in real-valued time. Let X be a node in the CTBN, and let x_1 and x_2 be two distinct states of X. Node X was observed to change from state x_1 to state x_2 at time t.*

Transition evidence would result when sensors can detect certain changes in the system. In this case, the sensors can detect exactly when and how the change occurs. Note that if the sensors can detect every state change, they observe the complete trajectory of the system.

7.1.2.3 Certain Interval Evidence. For continuous-time systems, however, evidence at an instantaneous point in time is not powerful enough. We might be monitoring the system continuously and be able to make claims about the state of the system throughout an interval of time. When this was first introduced, it was called negative evidence, but the "negative" referred to transitions rather than states [77]. The idea was that a transition did *not* occur over an interval of time. Since then, it has been referred to as continuous evidence or interval evidence. Certain interval evidence in a CTBN is defined formally as follows.

Definition 7.1.3 (Certain interval evidence)**.** *Let t_1 and t_2 be two instantaneous points in real-valued time such that $t_1 < t_2$. Let X be a node in the CTBN, and let x be a state of X. Node X was observed to be in state x throughout the interval $[t_1, t_2)$.*

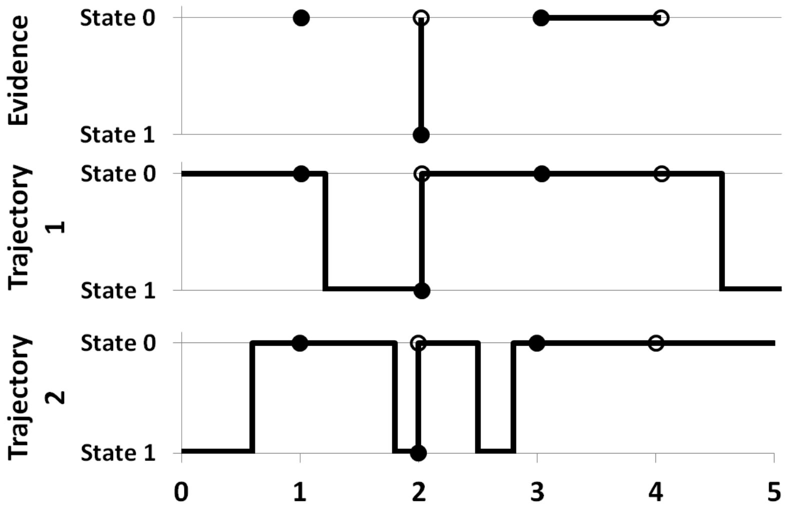


Figure 7.2: Example of evidence for CTBN.

Certain interval evidence is able to approximate both certain point and certain transition evidence [43]. For point evidence, we set the interval as ϵ for some infinitesimal value. For certain transition evidence, this becomes two successive instances of infinitesimally short interval evidence such that on a real-valued interval of time $[t - \epsilon, t)$, node X was observed to be in state x_1, while on a real-valued interval of time $[t, t + \epsilon)$, the state of node X was observed to be in state x_2.

An example of these three evidence types is shown in Figure 7.2 for a two-state node. The evidence can be thought of as a partial trajectory, a sequence of state and time pairs with gaps during which the state becomes unknown. In the example, point evidence at time $t = 1$ observes the node to be in state 0. Transition evidence at time $t = 2$ observes the state to transition from state 1 to state 0. Interval evidence from time $t = 3$ to $t = 4$ observes the state to remain in state 0. Figure 7.2 also shows two possible complete trajectories that conform to the evidence.

7.2 Extending Evidence in CTBNs

As seen with the advance from point evidence to transition and interval evidence, the temporal nature of the model gives rise to more varied types of continuous-time evidence. However, uncertain evidence, as used in BNs, has not yet been extended to CTBNs. Furthermore, the introduction of time adds another dimension. In this case, uncertain evidence and negative evidence in continuous-time become two distinct types of evidence. We now present the first definitions for these types of evidence in the CTBN.

7.2.1 Uncertain Positive Evidence

In this section, we define and describe uncertain positive evidence for CTBNs. First, we have uncertain point evidence, defined formally as follows.

Definition 7.2.1 (Uncertain point evidence)**.** *Let t be an instantaneous point in real-valued time. Let X be a node in the CTBN, and let λ_X be a set of likelihoods over the states of X. Node X was observed to be state x at time t with likelihood λ_x.*

Uncertain point evidence would result when the sensors can only be trusted to a certain degree, such as when they are known to have certain false positive and non-detect rates.

The likelihoods are analogous to λ_x of Equation 7.1. Because the observation is at a single instant in time, uncertain point evidence is sufficient to represent negative point evidence.

We can also have uncertainty in an observed transition. Uncertain transition evidence is used when the destination of the transition is known only with some probability. Uncertain transition evidence in a CTBN is defined formally as follows.

Definition 7.2.2 (Uncertain transition evidence)**.** *Let t be an instantaneous point in real-valued time. Let X be a node in the CTBN, and let λ_X^S and λ_X^E be two sets of likelihoods over the states of X. Node X was observed to change from state x_i with likelihood $\lambda_{x_i}^S$ and transition to state x_j with likelihood $\lambda_{x_j}^E$ at time t.*

Uncertain transition evidence would result when the sensor is able to determine only partially the source and/or destination state. The set of likelihoods can also be used to represent uncertainty that the state actually changed. This is done by setting λ_x^E to be non-zero whenever λ_x^S is non-zero. Because a state does not transition to itself in a continuous-time model, this implies that the state simply remained in state x at time t.

The definitions above represent uncertainty in the state. However, we could have uncertainty in the timing information as well. Therefore, we can formally define temporally uncertain transition evidence as follows.

Definition 7.2.3 (Temporally uncertain transition evidence)**.** *Let t_1 and t_2 be instantaneous points in real-valued time such that $t_1 < t_2$. Let X be a node in the CTBN, and let x_i and x_j be two distinct states of X. Node X was observed to transition from state x_i to x_j sometime during the interval $[t_1, t_2)$.*

Temporally uncertain transition evidence would result when the sensor is able to detect state changes in the system, but not instantaneously. The sensor might have a non-constant time-delay before the state change is detected.

Lastly, we have uncertain positive interval evidence. In this case, uncertain interval evidence cannot be used for negative interval evidence. Therefore, uncertain interval evidence must be identified as either positive or negative. Negative interval evidence is discussed in the next section, while uncertain positive interval evidence is defined formally as follows.

Definition 7.2.4 (Uncertain positive interval evidence)**.** *Let t_1 and t_2 be instantaneous points in real-valued time such that $t_1 < t_2$. Let X be a node in the CTBN, and let λ_X be a set of likelihoods over the states of X. Node X was observed to be in one state over the entire interval $[t_1, t_2)$, each state x with likelihood λ_x.*

Uncertain positive interval evidence would result when the sensor is able to detect states of the system, but is unable to determine the state with certainty. For uncertain interval evidence, the state of node X is known to be in exactly one state over the entire interval, but the identity of that state is known only with some likelihood.

7.2.2 Negative Evidence

In CTBNs, uncertain evidence is distinct from negative evidence. Uncertain interval evidence, for example, says that the system was in different states with different likelihoods, but whichever it was, the system stayed in that state over the whole interval. Negative evidence is saying something different. Negative interval evidence, for example, says that the system was never in a certain state over an interval; however, the system could have experienced multiple transitions between the other states over that interval. Formally, we define negative evidence as follows.

Definition 7.2.5 (Negative point evidence)**.** *Let t be an instantaneous point in real-valued time. Let X be a node in the CTBN, and let $X' \subset X$. Node X was observed to* ***not*** *be in any state $x \in X'$ at time t.*

Definition 7.2.6 (Negative transition evidence)**.** *Let t be an instantaneous point in real-valued time. Let X be a node in the CTBN, and let $X_S \subseteq X$ and $X_E \subset X$. Node X was observed to transition at time t from state $x_S \in X_S$ but* ***not*** *to any state $x_E \in X_E$.*

Definition 7.2.7 (Negative interval evidence)**.** *Let t_1 and t_2 be instantaneous points in real-valued time such that $t_1 < t_2$. Let X be a node in the CTBN, and let $X' \subset X$. Node X was observed to **not** be in states X' on a real-valued interval of time $[t_1, t_2)$.*

Note that uncertain and negative evidence are not mutually exclusive. We could have uncertainty in our negative evidence, in which we can rule out some states only with a certain probability. For instantaneous evidence, such as point evidence and transition evidence, uncertain evidence is sufficient to represent these. Uncertain positive interval evidence, on the other hand, cannot represent uncertain negative interval evidence, which is defined formally as follows.

Definition 7.2.8 (Uncertain negative interval evidence)**.** *Let t_1 and t_2 be instantaneous points in real-valued time such that $t_1 < t_2$. Let X be a node in the CTBN, let $X' \subseteq X$, and let $\lambda_{X'}$ be a set of likelihoods over the states of X'. Node X was observed to **not** be in state $x' \in X'$ on a real-valued interval of time $[t_1, t_2)$ with likelihood $\lambda_{x'}$.*

7.2.3 Relationships Between Types of Evidence

Given these definitions, note that, unlike with BNs, uncertain evidence is insufficient to represent all types of negative evidence. Uncertain evidence introduces uncertainty in the state, not the duration of the evidence. In other words, for an interval of uncertain evidence $\mathbf{e}$ on $[t_s, t_e)$,

$$P(X(t_1)|\mathbf{e}) = P(X(t_2)|\mathbf{e}),\ \forall\ t_1, t_2 \in [t_s, t_e). \tag{7.8}$$

On the other hand, for an interval of negative evidence $\mathbf{e}$ on $[t_s, t_e)$, we can say that

$$P(X(t) \in A|\mathbf{e}) = 0,\ \forall\ t \in [t_s, t_e), \tag{7.9}$$

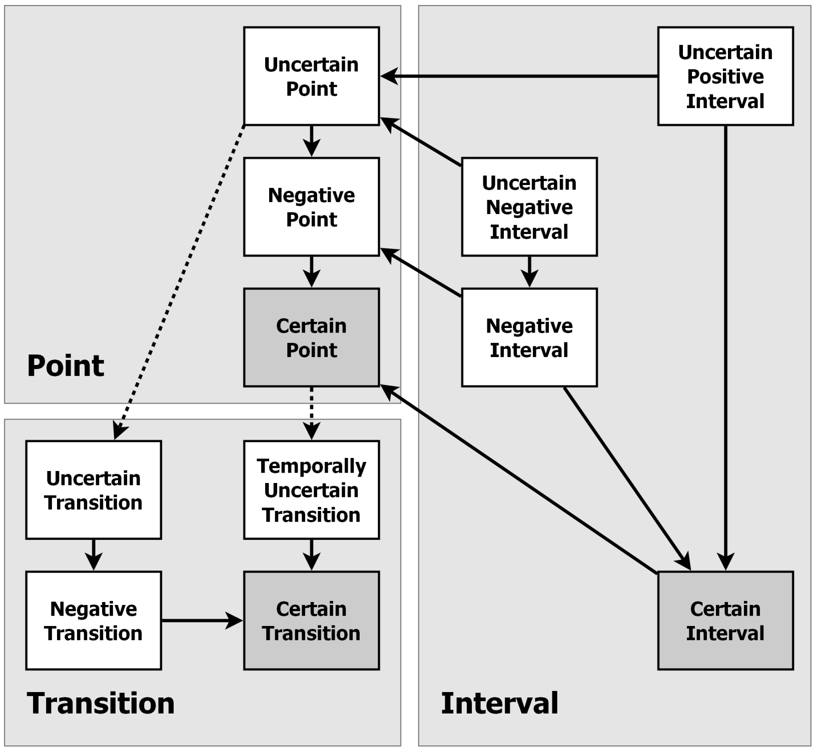


Figure 7.3: Relationships between types of continuous-time evidence.

but the probabilities $P(X(t) = x|\mathbf{e})$ for $x \notin A$ could be changing throughout $t \in [t_s, t_e)$.

As already seen, the various types of evidence are related, whether through generalizations or combinations. These relationships are summarized in Figure 7.3. The solid arrows show when the child type is a special case of the parent type, while the dashed arrows show when the child type is a combination of the parent type. We now provide the proofs for these relationships.

Proposition 7.2.1. *Uncertain point evidence is a special case of uncertain positive interval evidence.*

Proof. Set $t_2 = t_1 + \epsilon$ for infinitesimal value ϵ. □

Proposition 7.2.2. *Uncertain point evidence is a special case of uncertain negative interval evidence.*

Proof. Set $t_2 = t_1 + \epsilon$ for infinitesimal value ϵ. $\square$

Proposition 7.2.3. *Negative point evidence is a special case of uncertain point evidence.*

Proof. Set

$$\lambda_x = \begin{cases} 0 & \text{if } x \in X' \\ \frac{1}{|X|-|X'|} & \text{otherwise} \end{cases}.$$

$\square$

Proposition 7.2.4. *Negative point evidence is a special case of negative interval evidence.*

Proof. Set $t_2 = t_1 + \epsilon$ for infinitesimal value ϵ. $\square$

Proposition 7.2.5. *Certain point evidence is a special case of negative point evidence.*

Proof. Let y be the observed state. Set $X' = X/\{y\}$. $\square$

Proposition 7.2.6. *Certain point evidence is a special case of certain interval evidence.*

Proof. Set $t_2 = t_1 + \epsilon$ for infinitesimal value ϵ. $\square$

Proposition 7.2.7. *Uncertain transition evidence is a combination of two instances of uncertain point evidence.*

Proof. Let t_1 and t_2 be the times of the two instances of uncertain point evidence. Set $t_2 = t_1 + \epsilon$ for infinitesimal value ϵ. $\square$

Proposition 7.2.8. *Temporally uncertain transition evidence is a combination of two instances of certain point evidence.*

Proof. Set t_1 and t_2 as the times of the two instances of certain point evidence. $\square$

Proposition 7.2.9. *Negative transition evidence is a special case of uncertain transition evidence.*

Proof. Set

$$\lambda_x^S = \begin{cases} 0 & \text{if } x \in X_S \\ \frac{1}{|X|-|X_S|} & \text{otherwise} \end{cases}$$

and

$$\lambda_x^E = \begin{cases} 0 & \text{if } x \in X_E \\ \frac{1}{|X|-|X_E|} & \text{otherwise} \end{cases}.$$

$\square$

Proposition 7.2.10. *Certain transition evidence is a special case of negative transition evidence.*

Proof. Let state x_S be the observed source state and x_E be the observed destination state. Set $X_S = X/\{x_S\}$ and $X_E = X/\{x_E\}$. $\square$

Proposition 7.2.11. *Certain transition evidence is a special case of temporally uncertain transition evidence.*

Proof. Set $t_1 = t_2$ as the times of the two instances of certain point evidence. $\square$

Proposition 7.2.12. *Negative interval evidence is a special case of uncertain negative interval evidence.*

Proof. Set

$$\lambda_x = \begin{cases} 1 & \text{if } x \in X' \\ 0 & \text{otherwise} \end{cases}$$

$\square$

Proposition 7.2.13. *Certain interval evidence is a special case of uncertain positive interval evidence.*

Proof. Let state y be the observed state. Set

$$\lambda_x = \begin{cases} 1 & \text{if } x = y \\ 0 & \text{otherwise} \end{cases} .$$

□

Proposition 7.2.14. *Certain interval evidence is a special case of negative interval evidence.*

Proof. Let state y be the observed state. Set $X' = X/\{y\}$. □

7.3 CTBN Inference Algorithms with Extended Evidence Types

The types of continuous-time evidence defined up to this point are only useful if we can incorporate them into inference over the system. In this section, we show how exact inference and importance sampling can be extended to support the new types of evidence.

We note that we cannot represent uncertain and negative evidence simply by modifying the network itself. This is in contrast to the DBN with virtual evidence, for example. When unrolled, the DBN has distinct nodes in each timestep to which the virtual evidence nodes can be attached. In a CTBN, there is a single node for each variable that persists throughout the entire process. Therefore, the uncertain evidence must be applied to the node at the right time during the inference process itself.

We now show how exact inference and importance sampling can be extended to reason with uncertain and negative evidence. For the case of exact inference with uncertain negative evidence, we have not yet found a closed-form solution, so we use a rejection sampling step to validate that our extensions to importance sampling correctly incorporate uncertain negative evidence.

7.3.1 Exact Inference for Uncertain and Negative Evidence

As described in Section 2.2.3, exact inference for CTBNs can be achieved by amalgamating all of the nodes into the full joint intensity matrix (turning the CTBN into one large Markov process) and performing the forward-backward algorithm for Markov processes. Because the size of the full joint intensity matrix is exponential in the number of nodes, this algorithm quickly becomes intractable. We still show how to include uncertain and negative evidence with exact inference to provide a baseline for comparison to approximate methods.

7.3.1.1 Exact Inference with Uncertain Evidence.

For uncertain evidence over node X on segment i, the rows and columns representing transitions between the states of X are zeroed out, because the uncertain evidence knows that the state is constant during the interval but is uncertain as to the identity of that state. The matrix $\boldsymbol{Q}_{i-1,i}$ starts as the identity matrix. The diagonal elements corresponding to each state of X are multiplied by the uncertain evidence for that state. This weights the transition into each state according to the uncertainty in the evidence.

7.3.1.2 Exact Inference with Negative Evidence.

Negative evidence is a straightforward extension for exact inference. In this case, the rows and columns of $\boldsymbol{Q}_i$ are zeroed out in accordance with uncertain evidence. For certain evidence, all but one state of X are zeroed out. For negative evidence, there can be multiple states of X that are not zeroed out in $\boldsymbol{Q}_i$.

7.3.1.3 Exact Inference with Uncertain Negative Evidence. Suppose that segment i contains uncertain negative evidence. Exact inference over this segment will require more than just modifying the entries of the $\boldsymbol{Q}_i$ matrix. In fact, new matrices $\boldsymbol{Q_i}^{<t}$ and $\boldsymbol{Q_i}^{>t}$ need to be calculated first for each $t \in [t_i, t_{i+1})$ to compute

$$P(X(t) = k|\sigma_{[0,T)}) = \frac{1}{Z}\boldsymbol{\alpha}_{t_i} \exp(\boldsymbol{Q}_i^{<t}(t - t_i))\Delta_{k,k} \exp(\boldsymbol{Q}_i^{>t}(t_{i+1} - t))\boldsymbol{\beta}_{t_{i+1}}. \tag{7.10}$$

This equation is a modification of the forward-backward equation introduced in Section 2.2.3 that attempts to account for uncertain negative evidence. The matrix $\boldsymbol{Q_i}^{<t}$ is the intensity matrix for segment i prior to the query time t, the matrix $\boldsymbol{Q_i}^{>t}$ is the intensity matrix for segment i after the query time t, but note that $\boldsymbol{Q_i}^{<t} \neq \boldsymbol{Q_i}^{>t}$ for uncertain negative evidence. This is because the uncertainty in the negative evidence concerns the entire interval. Specifically, the magnitude of the uncertainty dictates how the probability decreases for the negative states being entered any time during the interval. On the other hand, $\boldsymbol{Q}_i$ specifies the instantaneous probability of entering those states. Depending on how the system is evolving, the probability of transitioning to the negative states during $[t_i, t)$ may be different than during $[t, t_{i+1})$. Thus, the entries of $\boldsymbol{Q}_i$ need to be re-weighted, but these weights are not constant over the interval. It is not clear whether there is a closed-form solution for calculating $\boldsymbol{Q}_i^{<t}$, $\boldsymbol{Q}_i^{>t}$, or even $\boldsymbol{Q}_i$ for a past or future segment of uncertain negative evidence. This is left as future work.

Algorithm 7.6 CTBN rejection sampling - subroutine for performing inference with uncertain negative evidence.

```
CTBNAcceptOrRejectSample($\sigma, \mathbf{e}'$)
1: for each $\langle \lambda_X, [t_1, t_2) \rangle \in \mathbf{e}'$
2:     for each $x \in X$
3:         $u \sim \text{Uniform}(0, 1)$
4:         if $\exists \langle X, t \rangle \in \sigma \ni (t \in [t_1, t_2) \wedge X = x)$
5:             if $u < \lambda_x$
6:                 return Reject
7:             end if
8:         end if
9:     end for
10: end for
11: return Accept
```

7.3.2 Rejection Sampling for Uncertain Negative Evidence

Instead of exact inference for validating the case of uncertain negative evidence, we can use rejection sampling. Here, rejection sampling refers to the process of approximating $P(X(t) = x|\mathbf{e})$ by generating samples from $P(X(t))$, removing all samples that violate $\mathbf{e}$, and computing the proportion with which $X(t) = x$ occurs in the remaining samples. Algorithm 7.6 shows the pseudocode for determining whether to accept or reject a sample with uncertain negative evidence. The algorithm is passed a trajectory σ that was generated using forward sampling (which can be generated using Algorithm 7.7 with no evidence). This sample is either accepted or rejected based on whether it conforms to the evidence $\mathbf{e}'$. In this case, $\mathbf{e}'$ consists of only uncertain negative interval evidence, consisting of likelihoods defined over time intervals $\langle \lambda_X, [t_1, t_2) \rangle$. The value $\lambda_x \in \lambda_X$ gives the likelihood with which state x of node X can be ruled out on the time interval $[t_1, t_2)$. Thus, to incorporate this uncertainty in the negative evidence, trajectories are rejected in accordance with the likelihood of the uncertain negative evidence. For example, suppose that the likelihood that state

x_0 of node X can be ruled out is 50% during the time interval $[t_1, t_2)$. The rejection sampling algorithm generates trajectories by forward sampling and randomly rejects 50% of all trajectories in which state x_0 occurred any time on $[t_1, t_2)$.

In general, rejection sampling is not feasible for inference in CTBNs. If the evidence includes any transition, rejection sampling cannot be used, as there is a zero probability of creating a trajectory that includes a transition at that exact time by chance. Even when evidence does not include transitions, the probability of generating a trajectory that conforms to that evidence by chance decreases as the evidence becomes more complex and its likelihood decreases. However, when there are only a few intervals of disjoint evidence, rejection sampling can provide a way to validate the extension to importance sampling for uncertain negative evidence.

7.3.3 Importance Sampling for Uncertain and Negative Evidence

The size of the matrices in the forward-backward algorithm for exact inference are exponential in the number of nodes in the CTBN. Furthermore, rejection sampling will break down for all but the most likely of evidence. For uncertain and negative evidence to be useful, we need to extend existing approximation algorithms to be able to handle these new types of evidence.

In this section, we extend the importance sampling algorithm [43]. The algorithm generates a set of weighted samples that conform to a partial trajectory $\mathbf{e}$ taken as evidence. The algorithm samples a proposal distribution P' that conforms to the evidence to fill in the unobserved intervals, generating a complete trajectory. Because the samples are drawn from P' to force each sample to be consistent with the evidence, each complete trajectory σ is weighted by the likelihood of the evidence, calculated as $w(\sigma) = \frac{P(\sigma,\mathbf{e})}{P'(\sigma)}$, with the cumulative weight as $W = \sum_{\sigma \in \mathcal{S}} w(\sigma)$. After generating a set of i.i.d. samples $\mathcal{S}$, the algorithm approximates the conditional expectation of any function f given the evidence $\mathbf{e}$ as:

$$\widehat{E}(f|\mathbf{e}) = \frac{1}{W} \sum_{\sigma \in \mathcal{S}} w(\sigma) f(\sigma) \tag{7.11}$$

While the extended importance sampling algorithm is similar in structure to the original algorithm of [43], the introduction of negative evidence requires several modifications that must be made throughout the entire algorithm. Because of these substantial differences, we present the extended importance sampling algorithm in full. The pseudocode for the main loop is given in Algorithm 7.7. This algorithm calls several helper methods, given in Algorithms 7.8 through 7.10. The algorithm uses the following notation:

- t is the current time of the sample.
- σ is the trajectory, consisting of a sequence of timestamp/state pairs.
- w is the likelihood (weight) of the sample.
- $\mathbf{e}'$ is a set of certain and/or uncertain observations, while $\mathbf{e}$ is a set containing only certain observations.
- $\mathbf{e}_X^{val}(t)$ is the value of X at time t according to the evidence or *null* if X has no evidence at time t. In the case of negative evidence, $\mathbf{e}_X^{val}(t)$ could be a set of values.
- $\mathbf{e}_X^{type}(t)$ is the type of evidence for X at time t, with values *pos* and *neg* for positive and negative evidence, respectively.
- $x(t)$ is the state of node X at time t.

Algorithm 7.7 CTBN importance sampling for performing inference with uncertain and negative evidence.

```
$\mathit{CTBNImportanceSample}(\mathcal{N}, \mathbf{e}', t_{end})$
 1: $t \leftarrow 0$
 2: $\mathit{Time} \leftarrow \mathit{null}$
 3: $\langle \mathbf{e}, w \rangle \leftarrow \mathit{SampleEvidence}(\mathbf{e}')$
 4: $\langle \mathbf{X}, \sigma, w \rangle \leftarrow \mathit{SampleInitialStates}(\mathcal{N}, \mathbf{e})$
 5: **loop** until termination
 6:     $\mathit{Time} \leftarrow \mathit{SampleTransitionTimes}(\mathit{Time}, \mathcal{N}, \mathbf{e})$
 7:     $X \leftarrow \arg\min_{X \in \mathbf{X}}[\mathit{Time}[X]]$
 8:     **if** $\mathit{Time}[X] \geq t_{end}$
 9:         $w \leftarrow \mathit{UpdateWeight}(X, w, t, t_{end}, \mathcal{N}, \mathbf{e})$
10:         break
11:     **else**
12:         $w \leftarrow \mathit{UpdateWeight}(X, w, t, \mathit{Time}[X], \mathcal{N}, \mathbf{e})$
13:     **end if**
14:     $t \leftarrow \mathit{Time}[X]$
15:     **if** $\mathbf{e}_X^{end}(t) = t \wedge (\mathbf{e}_X^{val}(t) = \mathit{null} \vee x(t) = \mathbf{e}_X^{val}(t) \vee \mathbf{e}_X^{type}(t) = \mathit{neg})$
16:         $\mathit{Time}[X] \leftarrow \mathit{null}$
17:     **else**
18:         **if** $\mathbf{e}_X^{type}(t) = \mathit{pos} \wedge \mathbf{e}_X^{val}(t) \neq \mathit{null} \wedge x(t) \neq \mathbf{e}_X^{val}(t) \wedge \mathbf{e}_X^{end}(t) - t < \epsilon$
19:             $w \leftarrow w \cdot \theta_{x(t)|\mathbf{u}_X(t)}[\mathbf{e}_X^{val}(t)]$
20:             $x(t) \leftarrow e_X^{val}(t)$
21:         **else**
22:             $\theta \leftarrow \theta_{x(t)|\mathbf{u}_X(t)}$
23:             **if** $(\mathbf{e}_X^{type}(t) = \mathit{neg})$
24:                 $w \leftarrow w \cdot (1 - \sum_{e \in \mathbf{e}_X^{val}(t)} \theta[e])$
25:                 $\mathit{Constrain}(\theta, \mathbf{e}_X^{val}(t))$
26:             **end if**
27:             $x(t) \sim \text{Multinomial}(\theta)$
28:         **end if**
29:         $X \leftarrow x(t)$
30:         $\mathit{Append}(\sigma, \langle X, t \rangle)$
31:         $\mathit{Time}(X) \leftarrow \mathit{null}$
32:         **for each** $Y \in \mathbf{Ch}(X)$
33:             $\mathit{Time}(Y) \leftarrow \mathit{null}$
34:         **end for**
35:     **end if**
36: **end loop**
37: return $\langle \sigma, w \rangle$
```

- $\theta^{\mathcal{B}}_{X|\mathbf{pa}_{\mathcal{B}}(X)}$ is the prior probability distribution of X given the parents of X in $\mathcal{B}$.
- $Time[X]$ is the proposed transition time of node X.
- $\mathbf{e}_X^{time}(t)$ is the first time after t when $\mathbf{e}_X^{val}(t)$ is defined.
- $\mathbf{e}_X^{end}(t)$ is the first time after or equal to t when $\mathbf{e}_X^{val}(t)$ changes value or becomes $null$.
- $q_{x(t)|\mathbf{u}_X(t)}$ is the exponential parameter of node X in state $x(t)$ given the parent states of X at time t.
- $\theta_{x(t)|\mathbf{u}_X(t)}$ is the transition probabilities out of state $x(t)$ given X's parents' states at time t.
- $\mathbf{e}_X^{conf}(x(t), t)$ is the soonest time at which the current state conflicts with upcoming evidence, either positive or negative, or $null$ if there is no future evidence or the current state matches the soonest positive evidence.

The method $Constrain(\theta, \mathbf{e})$ takes the probabilities for a multinomial distribution θ, zeroes out the states in θ listed in $\mathbf{e}$, and then re-normalizes θ. This is used for negative evidence, ensuring that a node can still transition but never to negated states.

The method *SampleEvidence*($\mathbf{e}$) handles uncertainty in the evidence, whether positive or negative. The states of any uncertain evidence are re-sampled before the generation of each sample. This applies to both uncertain positive and uncertain negative evidence. The method also checks to make sure the evidence sampled is feasible. For example, uncertain negative evidence must not rule out every state. Thus, for each sample generated by the sampling algorithm, all of the evidence can be treated as certain. However, the certain evidence could change between samples, according to the uncertainty of the evidence, and the weights of the final set of samples will reflect this in the given query.

The main method *CTBNImportanceSample*$(\mathcal{N}, \mathbf{e}', t_{end})$ generates a single, weighted trajectory σ that conforms to the evidence $\mathbf{e}$ and is weighted according to the likelihood of that evidence. Line 1 initializes the current time t to 0, while line 2 initializes the set of proposed transition times. Line 3 samples any uncertain evidence in $\mathbf{e}'$ and updates the weight w based on the likelihood of sampling that evidence. During the generation of this sample, the sampled evidence in $\mathbf{e}$ is treated as certain evidence. Line 4 generates the initial states for all nodes according to the prior distribution $\mathcal{B}$ while conforming to any evidence in $\mathbf{e}$ defined at $t = 0$. The weight of the sample is updated according to the likelihood of this evidence.

Lines 5-36 continue to generate transitions until the duration of the trajectory is at least t_{end}. Line 6 ensures all nodes have potential transition times. Line 7 gets the node with the soonest potential transition time. In lines 8-37, if that time is later than t_{end}, the trajectory is finished. The weight is updated through the last segment up until t_{end}, and the trajectory and its likelihood are returned. Otherwise, lines 11-13 update the weight over the current segment. Line 14 updates the current time to the end of the segment. A transition may not always occur at the end of the segment (hence the term "potential transition time"). A transition will not occur if the end of the segment falls on a change in evidence, such that the current time falls on the end or beginning of a segment of positive evidence for this node or the evidence is negative. In this case, lines 15-16 reset the potential transition time for this node, and the process returns to line 5. Otherwise, a transition will occur. Lines 18-20 handle the case when the evidence specifies the transition. This occurs when there is positive evidence at the current time (or within ϵ of the current time) and the current state of the node does not match the evidence. Thus, to conform to

Algorithm 7.8 CTBN importance sampling subroutine - sample initial states.

```
SampleInitialStates($\mathcal{N}, \mathbf{e}$)
1: $\sigma \leftarrow null$, $w \leftarrow 1$
2: for each $X \in \mathbf{X}$
3:     if $\mathbf{e}_X^{val}(0) \neq null$
4:         if $\mathbf{e}_X^{type}(0) = pos$
5:             $x(0) \leftarrow \mathbf{e}_X^{val}(0)$
6:             $w \leftarrow w \cdot \theta^{\mathcal{B}}_{x(0)|\mathbf{pa}_{\mathcal{B}}(0)}$
7:         else
8:             $\theta^{\mathcal{B}} \leftarrow \theta^{\mathcal{B}}_{x(0)|\mathbf{pa}_{\mathcal{B}}}$
9:             $Constrain(\theta^{\mathcal{B}}, \mathbf{e}_X^{val}(0))$
10:            $x(0) \sim \text{Multinomial}(\theta^{\mathcal{B}})$
11:            $w \leftarrow w \cdot (1 - \sum_{e \in \mathbf{e}_X^{val}(0)} \theta^{\beta}[e])$
12:        end if
13:    else
14:        $x(0) \sim \text{Multinomial}(\theta^{\mathcal{B}}_{X|\mathbf{Pa}_{\mathcal{B}}(X)})$
15:    end if
16:    $X \leftarrow x(0)$
17:    $Append(\sigma, \langle X, 0 \rangle)$
18: end for
19: return $\langle \mathbf{X}, \sigma, w \rangle$
```

the evidence, a transition is forced in line 20. Line 19 updates the weight with the likelihood of that transition. Otherwise, the node can transition to multiple states, and the destination state must be sampled. If there is negative evidence, lines 23-26 zero out the transition probabilities for these states and update the weight with the likelihood that the node did *not* transition to these states. In line 29, the state of the node is updated. Because a transition has occurred, line 30 adds the transition to the trajectory. Furthermore, lines 31-34 reset the potential transition times for the node and all of its children, as the change of state in the parent changes the current intensity matrix of each child. The process returns to line 5 to generate new potential transition times.

The helper method *SampleInitialStates*($\mathcal{N}, \mathbf{e}$) is given in Algorithm 7.8. The method is responsible for sampling the initial states of the trajectory while conform-

Algorithm 7.9 CTBN importance sampling subroutine - sample sojourn times.

```
$SampleTransitionTimes(Time, \mathcal{N}, \mathbf{e})$
1: for each $X \in \mathbf{X}$
2:     if $Time[X] = null$
3:         if $\mathbf{e}_X^{val}(t) \neq null$ and $\mathbf{e}_X^{type}(t) = pos$
4:             $\Delta t \leftarrow \mathbf{e}_X^{end}(t) - t$
5:         else
6:             $t_{conf} \leftarrow \mathbf{e}_X^{conf}(x(t), t)$
7:             if $t_{conf} \neq null$
8:                 $\Delta t \sim \text{Exponential}(q_{x(t)|\mathbf{u}_X(t)})$ given $\Delta t < (t_{conf} - t)$
9:             else
10:                $\Delta t \sim \text{Exponential}(q_{x(t)|\mathbf{u}_X(t)})$
11:                $t_e = \mathbf{e}_X^{time}(t)$
12:                if $x(t) = \mathbf{e}_X^{val}(t_e)$ and $t + \Delta t > t_e$
13:                    $\Delta t \leftarrow t_e - t$
14:                end if
15:            end if
16:        end if
17:        $Time[X] \leftarrow t + \Delta t$
18:    end if
19: end for
20: return $Time$
```

ing to the evidence. Line 1 creates an initially empty trajectory σ and initializes the weight w. Lines 2-18 loop over all nodes in $\mathcal{N}$. Lines 3-12 handle the case when the node has evidence set at $t = 0$. If the evidence is positive, lines 4-6 set the node to that state and update the weight with the likelihood of that evidence. If the evidence is negative, lines 7-12 zero out the transition probabilities for these states and update the weight with the likelihood that the node did *not* transition to these states. If no evidence is specified for this node at $t = 0$, line 14 samples from the prior distribution, and no weighting is necessary. Line 16 sets the initial state of the node, and line 17 adds the initial states to the trajectory. The current states, the current trajectory, and the current weight are returned in line 19.

The helper method $SampleTransitionTimes(Time, \mathcal{N}, \mathbf{e})$ is given in Algorithm 7.9. The method is responsible for generating proposed transition times that conform to the evidence. These are only proposed transition times, and transitions are not guaranteed to occur at these times. For example, whenever a parent node transitions, the children's proposed transition times will be re-sampled to account for their new conditional intensity matrix, and the proposed transitions times will change. If a node is currently within an interval of positive evidence or has upcoming positive evidence, the proposed transition times will be the start or end of the interval, respectively. However, a transition will not occur, because the state must be kept constant during the interval of positive evidence.

Lines 1-19 loop over all nodes in $\mathcal{N}$, while line 2 checks whether the current node needs a new proposed transition time. Line 3 checks whether the node is currently within positive evidence. If so, line 4 sets the node's proposed transition time as the end of the evidence. This does not mean that the node will transition immediately after the interval of positive evidence, but will have its proposed transition time sampled again once it becomes unobserved. If the node is not currently observed, then line 6 gets the soonest time (if it exists) at which the current state of the node conflicts with upcoming evidence. Line 7 checks whether the current state conflicts with upcoming evidence (if the time of the soonest conflict is set). This could be positive evidence (the current state will need to transition *to* the observed state at some point) or negative evidence (the current state will need to transition *away* from the set of states that are ruled out). In either case, the proposed transition time must be sampled from a truncated exponential distribution, shown in line 8, to condition on the upcoming evidence. Otherwise, line 10 simply samples from an exponential distribution. While the current state could be conforming to upcoming evidence, the sampled transition time could be past the end of the upcoming evidence. Thus,

Algorithm 7.10 CTBN importance sampling subroutine - update weight for interval evidence.

```
$UpdateWeight(Y, w, t_1, t_2, \mathcal{N}, \mathbf{e})$
 1: for each $X \in \mathbf{X}$
 2:     $t_e \leftarrow \mathbf{e}_X^{end}(t_1)$
 3:     if $\mathbf{e}_X^{val}(t_1) \neq null \wedge \mathbf{e}_X^{type}(t_1) = pos \wedge (\mathbf{e}_X^{val}(t_e) = null \vee \mathbf{e}_X^{type}(t_e) = neg)$
 4:         $w \leftarrow w \cdot \exp(-q_{x(t_1)|\mathbf{u}_X(t_1)}(t_2 - t_1))$
 5:     else
 6:         $t_{conf} \leftarrow \mathbf{e}_X^{conf}(x(t), t)$
 7:         if $t_{conf} \neq null$
 8:             if $X = Y$
 9:                 $w \leftarrow w \cdot (1 - \exp(-q_{x(t_1)|\mathbf{u}_X(t_1)}(t_e - t_1)))$
10:             else
11:                 $w \leftarrow w \cdot \frac{1-\exp(-q_{x(t_1)|\mathbf{u}_X(t_1)}(t_e - t_1))}{1-\exp(-q_{x(t_1)|\mathbf{u}_X(t_1)}(t_e - t_2))}$
12:         end if
13:     end if
14: end for
15: return $w$
```

line 11 gets the time of the next change in the evidence, and lines 12-14 make sure the sampled transition time does not exceed this time. Line 17 sets the proposed transition time for this node, and the set of proposed transition times for all nodes are returned in line 20.

The helper method *UpdateWeight*(Y, w, t_1, t_2) is given in Algorithm 7.10. The method is responsible for weighting the likelihood of the transition times, whether the state was observed over an interval or the transition time was sampled from a truncated exponential to conform to upcoming evidence.

Lines 1-14 loop over all nodes in $\mathcal{N}$. Line 2 gets the time of the next change in observation for this node. Line 3 checks whether the state of the node is currently known but will become unobserved before transitioning to another state. If this is the case, line 4 updates the weight by the likelihood that the node remained in the state for at least the interval observed. Otherwise, if the state was unknown, the transition time might have been sampled from a truncated exponential distribution. Line 6 gets the time at which the current state of this node conflicts with upcoming evidence, if it exists. Line 7 checks whether this time is set, i.e., whether the most recent proposed transition time for this node was sampled from a truncated exponential distribution. If the current node was the node with the soonest proposed transition time, the weight is updated with the likelihood of sampling the transition time from the truncated exponential, as per line 9. Otherwise, the proposed transition time must be later, and the weight is updated according to line 11. The weight for this segment over all of the variables is returned in line 15.

Note the extended importance sampling algorithm is able to estimate the conditional expected value $\widehat{E}(f|\mathbf{e}')$ for uncertain and negative evidence $\mathbf{e}'$, which means that our extension for uncertain and negative evidence is fully compatible with the performance functions of Chapter 4. Thus, we can still reason about the performance of a system using factored performance functions even with uncertain and negative evidence. Although the approximate inference algorithms of Chapter 5 are not designed for interval evidence, they are compatible with the uncertain and negative point and transition evidence presented in this chapter.

7.4 Experiments

We demonstrate the extended types of evidence on the drug effect network presented by [17] as shown in Figure 7.4. All of our observations involve state c_0 of Concentration. This node has three states, which allows us to show uncertain transitions and to show how negative evidence rules out one of those states and renormalizes the probability between the other two states. The node also has parents and children, which allows us to show how the new types of evidence affect the probabilities of nodes around the observed node. For each setting of evidence, we ran the modified impor-

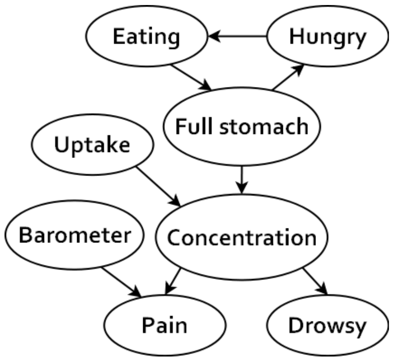


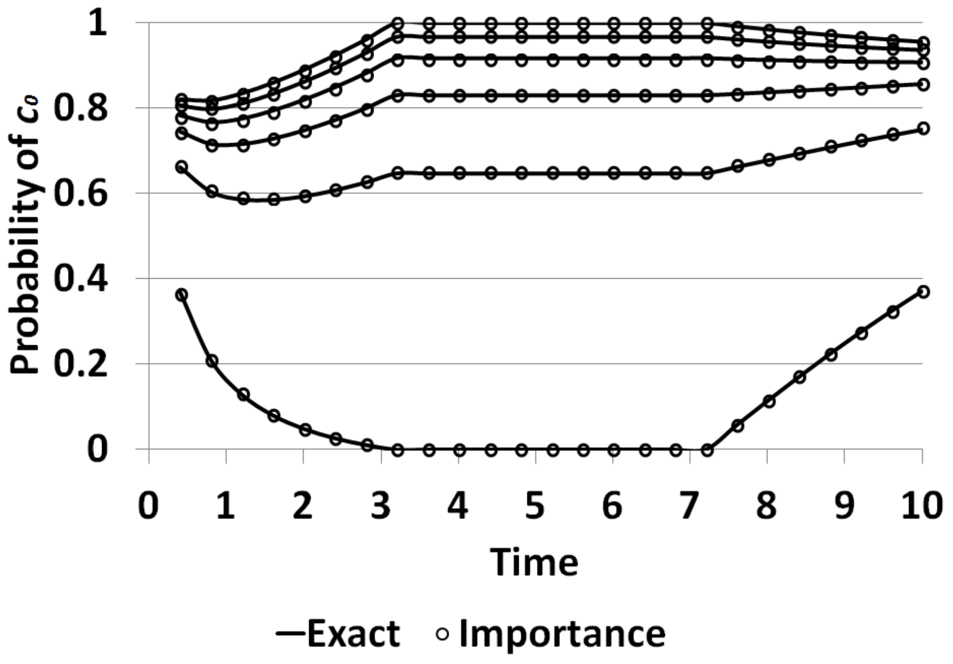
Figure 7.4: Drug effect network.

tance sampling algorithm with 100,000 samples over the time interval $[0.0, 10.0)$. We query the probabilities over time in 0.4 increments. For exact inference, we interpolate the points and plot the curves. We plot the probabilities returned by importance sampling and rejection sampling as points, shown as circles and crosses, respectively. We can evaluate the accuracy of the approximate sampling methods by how far the points deviate from the curves.

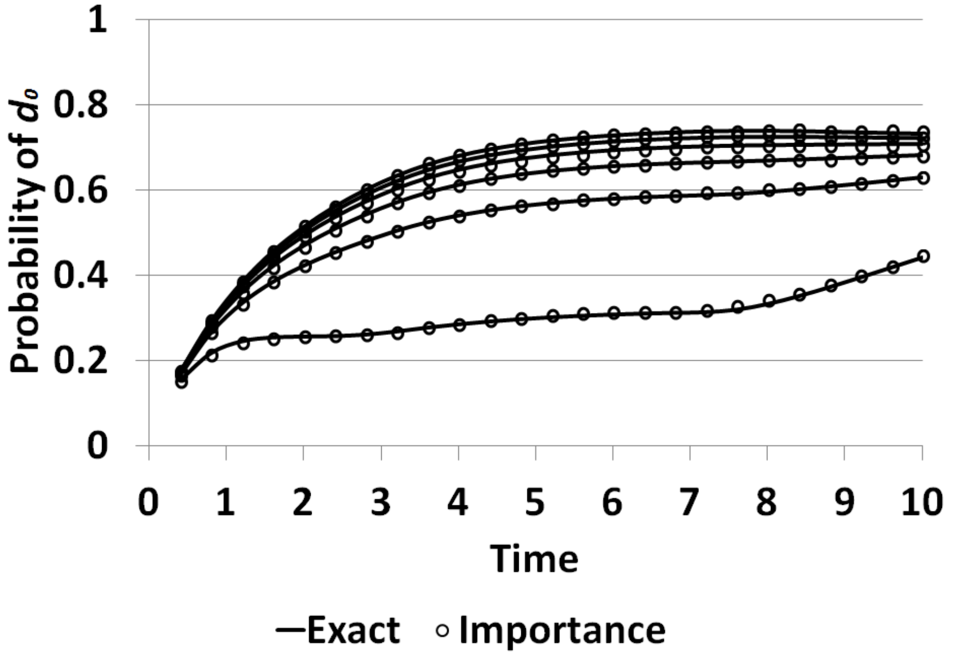
7.4.1 Uncertain Positive Interval Evidence

For uncertain positive evidence, we vary the likelihood of the system being in state c_0 over the interval $[3.0, 7.0)$ from 0.0 to 1.0 in increments of 0.2 and divide the remaining probability uniformly between c_1 and c_2. The top-most curve of Figure 7.5a shows when c_0 can be ruled out with complete certainty. The bottom-most curve of Figure 7.5a shows when c_0 is observed with complete certainty.

As expected, the uncertain positive observation of Concentration affects the probability of the states of other nodes as well. Figure 7.5b shows the evolving probabilities of state d_0 of Drowsy given the same varying uncertain positive observations to Concentration. The top-most curve of Figure 7.5b shows the effect on d_0 when c_0 can be ruled out with complete certainty, while the bottom-most curve shows the effect on



(a) Probability of c_0 through time.



(b) Probability of d_0 through time.

Figure 7.5: Uncertain positive interval evidence.

d_0 when c_0 is observed with complete certainty. As we can see by how the points fall close to the curves, the importance sampling algorithm is able to accurately estimate the probabilities with uncertain positive evidence.

7.4.2 Uncertain Negative Interval Evidence

Now we show uncertain negative interval evidence. For the uncertain negative evidence, we vary the probability of state c_0 being ruled out over the interval $[3.0, 7.0)$, from 0.0 to 1.0 in increments of 0.2. The top-most curve of Figure 7.6a shows when c_0 cannot be ruled out with any certainty, which is the same as no observation. The bottom-most curve of Figure 7.6a shows when c_0 can be ruled out with complete certainty. Hence, the probability of c_0 is exactly 0.0 for the entire interval $[3.0, 7.0)$.

As above, the uncertain negative observation of Concentration affects the other node. Figure 7.6b shows the evolving probabilities of state d_0 of Drowsy given the same varying uncertain negative observations to Concentration. The top-most curve of Figure 7.6b shows the effect on d_0 when c_0 cannot be ruled out with any certainty, while the bottom-most curve shows the effect on d_0 when c_0 can be ruled out with complete certainty.

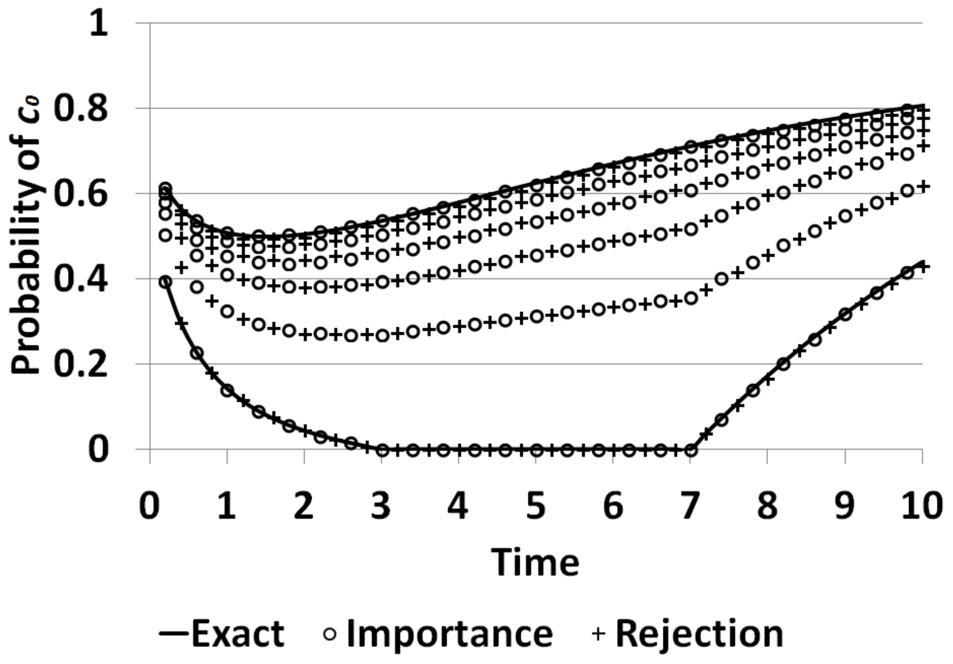
To more easily distinguish between the points arising from the two sampling methods, the probabilities calculated from rejection sampling are offset in time from importance sampling by 0.2. As we can see by how the points fall close to the curves, both importance sampling and rejection sampling are able to accurately estimate the probabilities with certain negative evidence (top and bottom curves), thereby validating both approaches. For uncertain negative evidence, we can see that the two give consistent results, indicating that they are correctly incorporating the uncertainty in the negative evidence.

As already discussed, uncertain and negative evidence become distinct types of evidence when the evidence is temporal. The difference between them becomes apparent when comparing Figures 7.5 and 7.6. Uncertain positive evidence observes the state to be constant over the interval, shown by the constant probability of Figure 7.5a on the interval $[3.0, 7.0)$. However, the uncertain negative evidence rules out states with some probability, but does not rule out state transitions. Therefore, as time progresses, the state probability might also vary, even if it has a non-zero probability of being ruled out. Because the system is tending toward c_0, the probability of c_0 increases even with negative evidence on c_0 when there is uncertainty in that evidence.

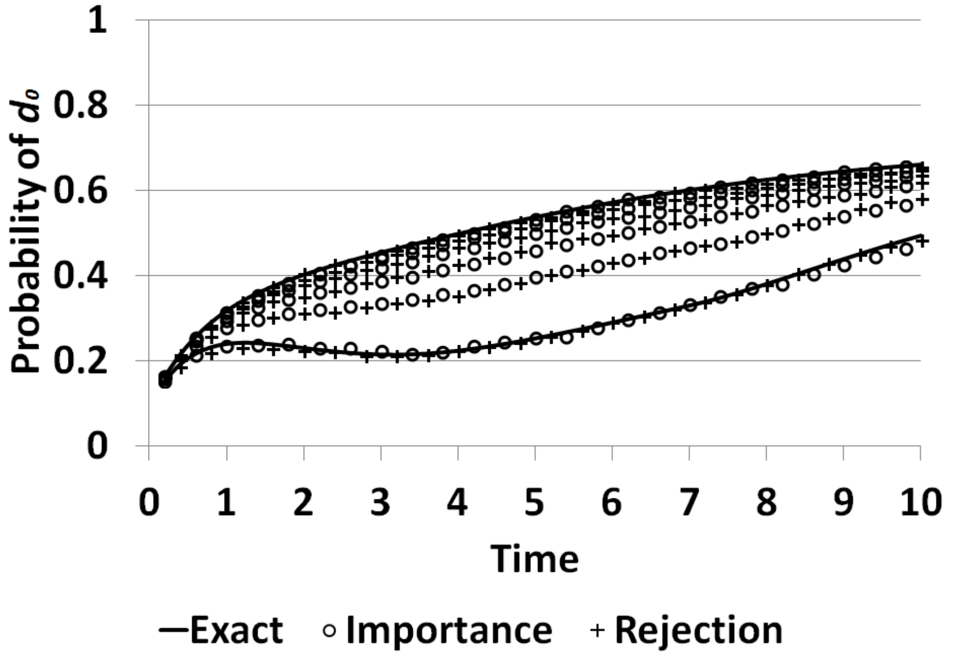
7.4.3 Duration of Uncertain Interval Evidence

Next we demonstrate how the duration of an observation can affect uncertain evidence. We set the likelihoods for the states of Concentration to be 0.2, 0.4, and 0.4 for states c_0, c_1, and c_2, respectively. We vary the duration of this observation from point evidence at time 3.0 to interval evidence over $[3.0, 7.0)$ in increments of 1.0. The bottom-most curves of Figure 7.7 show the effect of point evidence on the probabilities of c_0 and d_0, while the top-most curves show the effect of interval evidence over $[3.0, 7.0)$ on the probabilities of c_0 and d_0.

As we can see from these results, if an observation is uncertain, the duration of the observation affects the probabilities. In this network, Concentration tends to remain in c_0 longer than in c_1 or c_2. Therefore, if we know that the state of Concentration did not change over an interval, c_0 becomes more and more likely as the length of that interval increases, even when given the same likelihoods in the observations.

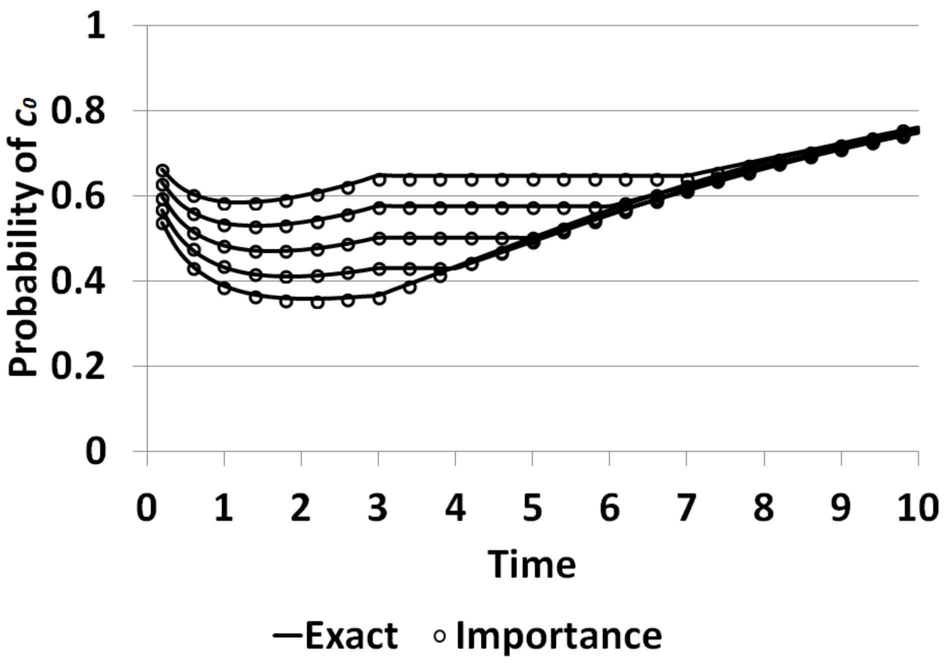


(a) Probability of c_0 through time.

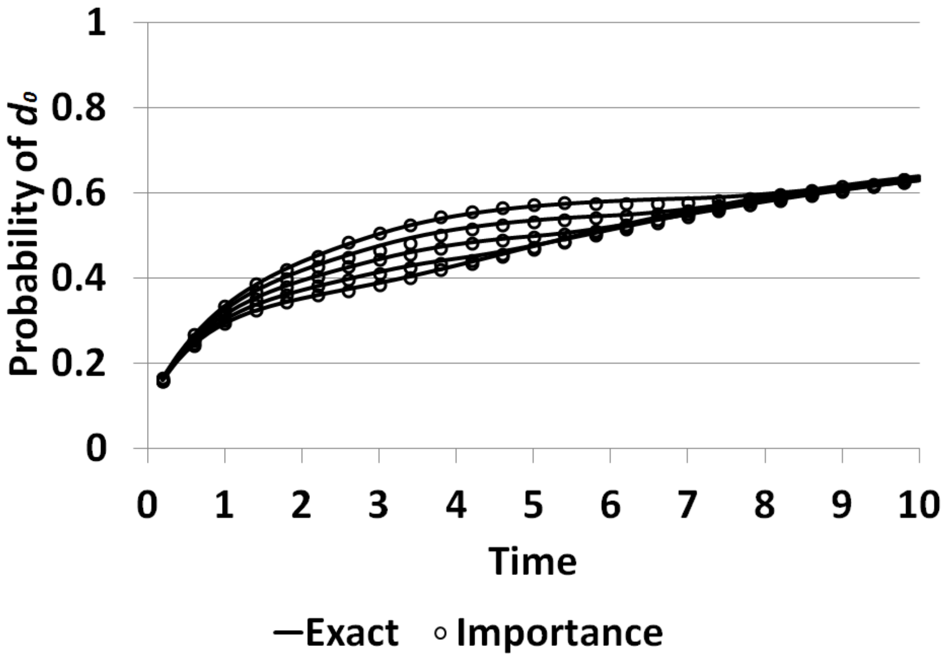


(b) Probability of d_0 through time.

Figure 7.6: Uncertain negative interval evidence.

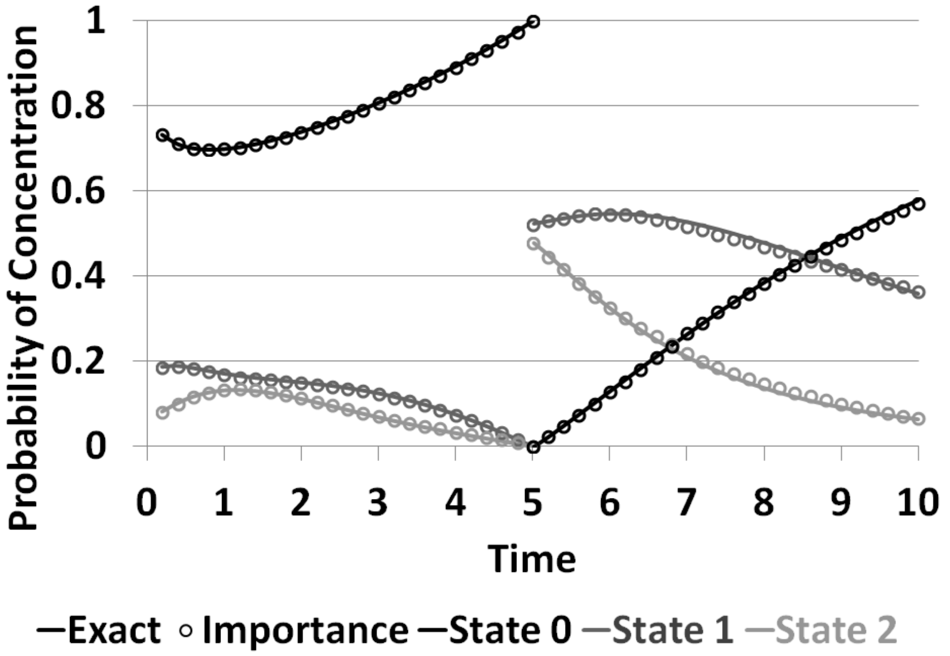


(a) Probability of c_0 through time.

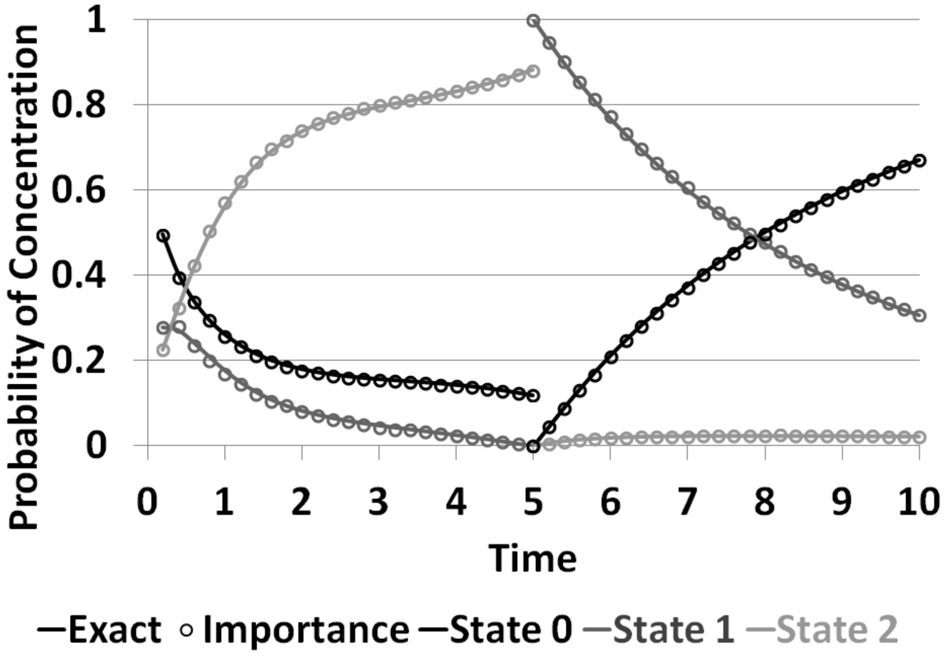


(b) Probability of d_0 through time.

Figure 7.7: Varying duration of uncertain interval evidence.



(a) State probabilities of Concentration through time.



(b) State probabilities of Concentration through time.

Figure 7.8: Uncertain transitions *to* and *from* a given state.

7.4.4 Uncertain Transition Evidence

Lastly, we give an example of uncertain transition evidence. Figure 7.8 shows the state probabilities of Concentration when an uncertain transition is observed at time 5.0. Figure 7.8a shows the probabilities when the state is known to transition *from* state c_0, but the destination state is unknown with $\lambda_{c_1} = \lambda_{c_2} = 0.5$. Figure 7.8b shows the probabilities when the state is known to transition *to* state c_1, but the source state is unknown with $\lambda_{c_0} = \lambda_{c_2} = 0.5$.

The transition probabilities between the states of Concentration are determined by the current states of its parents (Uptake and Full stomach). Thus, Figure 7.8a takes into account the probable states of the parents at time 5.0, in addition to Concentration's transition probabilities. Likewise, Figure 7.8b shows that, reasoning over all the state combinations of Concentration's parents, a transition to c_1 most likely originated from c_2.

7.5 Conclusion

Continuous-time systems allow for much greater variety in the types of evidence (as opposed to DBNs, for example) that would be useful to know about the system. For CTBNs, evidence can be over an interval or at a single point, corresponding to the system being observed in a particular state. The types of continuous-time evidence can be further generalized to include uncertain and negative evidence. Uncertain and negative evidence in continuous-time systems represents a novel and useful generalization of evidence that makes the models more applicable and versatile.

We presented the first definitions for uncertain and negative variations with point, transition, and interval evidence in CTBNs and showed the relationships between these evidence types. We showed how to extend the CTBN importance sampling algorithm to support all of these evidence types, how to extend the forward-backward algorithm in CTBNs to support uncertain positive evidence and certain negative evidence, and how to use a rejection sampling technique in CTBNs for uncertain negative evidence. As discussed in Section 2.3.6, there are several other CTBN inference algorithms that have been developed, each with their own strengths and weaknesses. Thus, to give the greatest flexibility to CTBN modelers who have evidence that is uncertain or negative (or both), it would be useful to extend these algorithms to support uncertain and negative evidence as well.

CHAPTER 8

CONCLUSION

We conclude with a summary of our contributions and directions for future work.

8.1 Summary

In this dissertation, we described five novel and useful extensions to CTBN modeling and inference.

First, we extended the complexity theory of inference in CTBNs. While the NP-hardness results of exact and approximate CTBN inference given the initial states are not surprising (considering what we know about Bayesian networks), such results did not exist previously. These results for the CTBN are now proved explicitly. The proof for exact inference justifies the focus of researchers on approximate methods for CTBN inference. However, the proofs for approximate inference warn modelers that even approximate inference may not be tractable, regardless of whether the maximum number of parents is constrained to be less than or equal to three. Our experiments demonstrated the reduction and showed the exponential behavior of the sampling algorithm as predicted by the theorem. The complexity proofs for the general case may motivate efforts toward finding efficient exact or provably bounded approximate inference methods for useful special cases of CTBNs.

Second, we introduced and formalized factored performance functions over the nodes of the CTBN. These performance functions place user-defined values on various behaviors of the model, allowing for a much greater variety in the types of queries that can be asked of the model. The factorization of the performance functions allows individual performance functions to be assigned to individual nodes, which makes them easier for a modeler to interpret and easier to implement in a CTBN reasoner. For values placed on complex interactions between multiple states of multiple nodes, we show how to augment the network structure with synergy nodes. For modelers these synergy nodes provide a visual representation of the dependence between performance functions in much the same way that the CTBN arcs show dependence between the behaviors of subsystems. They also provide a way to compute synergistic functions directly during the inference process. We showed an application of the performance functions and synergy nodes to a real-world network consisting of fleet of vehicles. Through the use of a synergy node, the sampling algorithm could compute the expected performance of the system after evaluating fewer samples than the brute-force approach.

Third, we presented two novel node isolation algorithms for node marginalization that builds on previous node marginalization methods. Node isolation provides a more nuanced treatment of the entry and exit distributions of the subsystem being marginalized. By weighting the entrance and exit distributions of the subsystems with entries from the steady-state distribution, node isolation tends to be better at estimating the long-term behavior of the unconditional Markov process than the previous methods. We compared the existing linearization and subsystem methods with two variations of the node isolation approach. Whereas the linearization and subsystem methods rely on the immediate behavior of the CTBN for their approximations, the experiments demonstrated that the node isolation concept is better able to describe the long-term behavior of the CTBN.

Fourth, we leveraged both the factored performance functions and the node isolation method to show how sensitivity analysis can be applied to the CTBN model. Sensitivity analysis is a new form of inference for CTBNs, allowing modelers to measure how changes in model parameters affect the expected performance of the system. Extending beyond a straightforward application of Markov process sensitivity analysis to CTBNs, we showed how to use node isolation to follow the factorization of the network in order to perform sensitivity analysis on different subnetworks independently, without having to confront the exponential state-space of the system as a whole. We demonstrated sensitivity analysis on three reliability models—one synthetic network and two real-world networks. The experiments demonstrated the benefit of subdividing the CTBN into independent subnetworks on which to perform sensitivity analysis.

Fifth, we defined, for the first time, uncertain and negative continuous-time evidence for CTBNs. We showed how the new "dimensions" of uncertainty and negativity can be applied to the existing point, transition, and interval evidence types. We categorized the different combinations of evidence and proved the generalization and specialization relationships between them. We extended the CTBN importance sampling algorithm to be able to reason over all of these new types of evidence. Thus, modelers are now able to include a wider variety of available observations and reason about uncertainty in the observations, as well as uncertainty in the system's behavior. We demonstrated the impact of different types of evidence on the probability distributions of observed and unobserved nodes through time, showing variations in the magnitude of the uncertainty in the state, duration, and transition.

8.2 Future Work

As with the BN model when it was introduced, the CTBN model continues to be extended, refined, and applied to new domains. While we have advanced several areas, there are other directions for CTBN modeling, learning, and inference that have yet to be explored.

In complexity theory, the CTBN has several potential special-case complexity results. For example, there could be constrained structures and/or parameters that admit tractable inference algorithms, analogous to polytree-based inference in BNs. The complexity theory behind CTBN structure learning and parameter estimation could be extended beyond the proof for finding the k highest-scoring parents in polynomial-time. Our experiments seemed to indicate that the CTBN for deciding the satisfiability of a Boolean expression could also give the fraction of satisfying truth assignments by examining the convergence of the probabilities of the D_m node. Formally proving this is left as future work but would most likely proceed by examining the limiting distribution of the full joint intensity matrix in relation to the corresponding clause.

We would like to explore further applications for our factored performance functions. In particular we would like to demonstrate an application that uses families of performance functions over the nodes of the CTBN. The families could represent trade-offs in the behavior of the system, such as cost vs. availability. From there one could perform multi-objective optimization over the CTBN. For example, our approach for CTBN sensitivity analysis could be used to find model parameters that meet acceptable trade-off criteria for competing performance functions.

Our node isolation methods show promise over the previous linearization and subsystem methods for filtering and prediction with point evidence by better approximating the long-term behavior of the nodes. Future work for the node isolation methods would be to extend these methods to be able to include interval evidence, as well as perform smoothing. More research can be done on the quality of the approximation of the node isolation method, especially in the presence of cycles.

We have shown how perturbation realization and node isolation can be used to perform more efficient sensitivity analysis of CTBNs. However, the node isolation methods produce approximating unconditional intensity matrices from which perturbation realization generates trajectories to approximate potentials. Analyzing the correlation between the quality of approximation of the potentials and the quality of the approximation of the unconditional intensity matrices is an area for future work.

Our definitions for uncertain and negative evidence in CTBNs expand the types of observations that CTBN inference algorithms can incorporate. While we extended the exact forward-backward algorithm and the importance sampling algorithm to be able to support these new types of evidence, extending other CTBN inference algorithms to also support uncertain and negative evidence is future work. Other sampling algorithms, such as Gibbs sampling, might be able to use some of the concepts used in the importance sampling algorithm, such as sampling from uncertain observations to determine the evidence for each sample. Methods that use the forward-backward algorithm on a marginalized subnetwork, such as expectation propagation, might be able to use our extensions for exact inference to include uncertain and negative evidence separately. As we do not yet have an exact formulation for uncertain negative evidence, solving this problem in the exact case may enable this approximation method to handle uncertain negative evidence as well.

We have shown CTBN sensitivity analysis as a new form of inference, which calculates expected changes in system performance given changes in system parameters rather than calculating expected system behavior given system observations. There are other types of inference that could be useful for CTBNs. For example, adaptive inference is the task of recalculating the results of inference after the inference problem has been slightly changed from previously calculated results. Suppose we have used CTBN importance sampling to compute $\widehat{E}(f|\mathbf{e}'_1)$. Can we calculate $\widehat{E}(f|\mathbf{e}'_2)$ for some $\mathbf{e}_2'$ that is similar to $\mathbf{e}_1'$ by reusing some of the calculations from $\widehat{E}(f|\mathbf{e}_1')$? For another example, abductive inference is the problem of computing the most probable explanation for a set of observations. That is, given a set of observations about the states of the nodes, find states for (at least some of) the remaining nodes that best explains the observations. One of the first problems in extending abductive inference to the CTBN is to formulate exactly what this means in continuous-time setting.

Lastly, we have focused on CTBN modeling and inference. As a data-driven model, CTBNs can be learned from data. This includes learning the structure of the network, the dependencies between subsystems, and learning the parameters, the entries in the conditional intensity matrices. The first CTBN structure learning algorithm used a scoring-based metric to determine the parents of a node. Algorithms for BN structure learning include these types of algorithms but also constraint-based algorithms that test for conditional dependence and independence in the data and construct a structure that matches the results of these tests. One of the first problems in extending constraint-based structure learning for CTBNs is to extend the dependence and independence tests for continuous-time data. There are only a few CTBN learning algorithms that exist for learning when the data is incomplete and for learning the parameters of hidden nodes (additional nodes that are not directly represented in the data but are inferred from other values in the data). These algorithms rely on heuristics and provide only approximate solutions. Thus, advances to CTBN learning that improve these approximations would also be valuable future work.

These are just some examples, inspired by a similar progression in modeling, learning, and inference in BNs. Given the prevalence of continuous-time systems, the application of CTBNs to real-world problems will likely increase as well, motivating further research to make the CTBN more powerful and more applicable to solving these problems.

REFERENCES CITED

[1] H. Boudali, J. B. Dugan. A continuous-time Bayesian network reliability modeling, and analysis framework. *IEEE Transactions on Reliability*, 55(1):86–97, 2006.

[2] Y. Gong. Speech recognition in noisy environments: a survey. *Speech Communication*, 16(3):261–291, 1995.

[3] R. Plamondon, S. N. Srihari. Online and off-line handwriting recognition: a comprehensive survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 22(1):63–84, 2000.

[4] J. C. Augusto. Temporal reasoning for decision support in medicine. *Artificial Intelligence in Medicine*, 33(1):1–24, 2005.

[5] K. Adlassnig, C. Combi, A. K. Das, E. T. Keravnou, G. Pozzi. Temporal representation and reasoning in medicine: Research directions and challenges. *Artificial Intelligence in Medicine*, 38(2):101–113, 2006.

[6] M. Schwabacher, K. Goebel. A survey of artificial intelligence for prognostics. *AAAI Fall Symposium*, pages 107–114, 2007.

[7] A. K. S. Jardine, D. Lin, D. Banjevic. A review on machinery diagnostics and prognostics implementing condition-based maintenance. *Mechanical Systems and Signal Processing*, 20(7):1483–1510, 2006.

[8] D. T. Gillespie. Stochastic simulation of chemical kinetics. *Annual Review of Physical Chemistry*, 58:35–55, 2007.

[9] S. S. Carroll, D. L. Pearson. Detecting and modeling spatial and temporal dependence in conservation biology. *Conservation Biology*, 14(6):1893–1897, 2000.

[10] C. K. Wikle. Hierarchical Bayesian models for predicting the spread of ecological processes. *Ecology*, 84(6):1382–1394, 2003.

[11] G. Jacob, H. Debar, E. Filiol. Behavioral detection of malware: from a survey towards an established taxonomy. *Journal in Computer Virology*, 4(3):251–266, 2008.

[12] Z. Chen, C. Ji. Spatial-temporal modeling of malware propagation in networks. *IEEE Transactions on Neural Networks*, 16(5):1291–1303, 2005.

[13] M. P. Clements, P. H. Franses, N. R. Swanson. Forecasting economic and financial time-series with non-linear models. *International Journal of Forecasting*, 20(2):169–183, 2004.

[14] L. Bauwens, S. Laurent, J. V. K. Rombouts. Multivariate GARCH models: a survey. *Journal of Applied Econometrics*, 21(1):79–109, 2006.

[15] D. Kempe, J. Kleinberg, A. Kumar. Connectivity and inference problems for temporal networks. *Proceedings of the 32nd Annual ACM Symposium on Theory of Computing*, pages 504–513. ACM, 2000.

[16] L. Vila. A survey on temporal reasoning in artificial intelligence. *AI Communications*, 7(1):4–28, 1994.

[17] U. Nodelman, C. Shelton, D. Koller. Continuous time Bayesian networks. *Proceedings of the 18th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 378–387, 2002.

[18] U. Nodelman. *Continuous Time Bayesian Networks.* Doctoral Dissertation, Stanford University, Stanford, California, 2007.

[19] N. A. Gershenfeld. *The Nature of Mathematical Modeling.* Cambridge University Press, 1999.

[20] G. Welch. An introduction to Kalman filtering, TR95-041. Technical Report, University of North Carolina Chapel Hill, 2002.

[21] J. D. Hamilton. State-space models. Volume 4 series *Handbook of Econometrics*, pages 3039–3080. Elsevier, 1994.

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

[23] M. S. Arulampalam, S. Maskell, N. Gordon, T. Clapp. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing*, 50(2):174–188, 2002.

[24] H. M. Taylor, S. Karlin. *An Introduction to Stochastic Modeling.* Academic press, 1998.

[25] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[26] K. P. Murphy. *Dynamic Bayesian networks: representation, inference and learning.* Doctoral Dissertation, University of California Los Angeles, 2002.

[27] J. T. Connor, R. D. Martin, L. E. Atlas. Recurrent neural networks and robust time series prediction. *IEEE Transactions on Neural Networks*, 5(2):240–254, 1994.

[28] K. Funahashi, Y. Nakamura. Approximation of dynamical systems by continuous time recurrent neural networks. *Neural Networks*, 6(6):801–806, 1993.

[29] U. Nodelman, C. R. Shelton, D. Koller. Expectation maximization and complex duration distributions for continuous time Bayesian networks. *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[30] D. Geiger, T. Verma, J. Pearl. Identifying independence in Bayesian networks. *Networks*, 20(5):507–534, 1990.

[31] R. Setiono, H. Liu. Understanding neural networks via rule extraction. *International Joint Conferences on Artificial Intelligence (IJCAI)*, Volume 1, pages 480–485, 1995.

[32] M. W. Craven, J. W. Shavlik. Extracting tree-structured representations of trained networks. *Advances in Neural Information Processing Systems*, pages 24–30, 1996.

[33] D. Cao. *Novel models and algorithms for systems reliability modeling and optimization.* Doctoral Dissertation, Wayne State University, 2011.

[34] D. H. Wolpert. The lack of a priori distinctions between learning algorithms. *Neural Computation*, 8(7):1341–1390, 1996.

[35] D. H. Wolpert, W. G. Macready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1):67–82, 1997.

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

[37] M. A. J. van Gerven, B. G. Taal, P. J. F. Lucas. Dynamic Bayesian networks as prognostic models for clinical patient management. *Journal of Biomedical Informatics*, 41(4):515–529, 2008.

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

[39] F. Camci, R. B. Chinnam. Dynamic Bayesian networks for machine diagnostics: hierarchical hidden Markov models vs. competitive learning. *Proceedings of the IEEE International Joint Conference on Neural Networks (IJCNN)*, Volume 3, pages 1752–1757, 2005.

[40] A. Muller, M. Suhner, 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.

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

[42] K. Medjaher, J. Moya, N. Zerhouni. Failure prognostic by using dynamic Bayesian networks. *Dependable Control of Discrete Systems*, 1:291–296, 2009.

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

[44] C. Moler, C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1):3–49, 2003.

[45] I. Cohn. *Mean Field Variational Approximations in Continuous-Time Markov Processes*. Doctoral Dissertation, The Hebrew University, 2009.

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

[47] J. C. Weiss, S. Natarajan, C. D. Page, Jr. Learning when to reject an importance sample. *AAAI (Late-Breaking Developments)*, Volume WS-13-17 series *AAAI Workshops*. AAAI, 2013.

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

[49] V. Rao, Y. W. Teh. Fast MCMC sampling for Markov jump processes and extensions. *Journal of Machine Learning Research (JMLR)*, 14(1):3295–3320, 2013.

[50] B. Miasojedow, W. Niemiro, J. Noble, K. Opalski. Metropolis-type algorithms for continuous time Bayesian networks. *arXiv preprint arXiv:1403.4035*, 2014.

[51] U. Nodelman, D. Koller, C. R. Shelton. Expectation propagation for continuous time Bayesian networks. *Proceedings of the 21st Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 431–440. AUAI Press, 2005.

[52] S. Saria, U. Nodelman, D. Koller. Reasoning at the right time granularity. *Proceedings of the 23rd Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, 2007.

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

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

[55] E. B. Celikkaya, C. R. Shelton, W. Lam. Factored filtering of continuous-time systems. Fabio Gagliardi Cozman, Avi Pfeffer, editors, *UAI*, pages 61–68. AUAI Press, 2011.

[56] B. Ng, A. Pfeffer, R. Dearden. Continuous time particle filtering. *International Joint Conference on Artificial Intelligence (IJCAI)*, Volume 19, pages 1360–1365, 2005.

[57] U. Nodelman, C. R. Shelton, D. Koller. Learning continuous time Bayesian networks. *Proceedings of the 19th Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 451–458. Morgan Kaufmann Publishers Inc., 2002.

[58] D. Shi, J. You. Update rules for parameter estimation in continuous time Bayesian network. *PRICAI 2006: Trends in Artificial Intelligence*, pages 140–149. Springer, 2006.

[59] U. Nodelman, E. Horvitz. Continuous time Bayesian networks for inferring users' presence and activities with extensions for modeling and evaluation. *Microsoft Research*, 2003.

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

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

[62] Y. Fan. *Continuous Time Bayesian Network Approximate Inference and Social Network Applications.* Doctoral Dissertation, University of California Riverside, 2009.

[63] D. Shi, X. Tang, J. You. An intelligent system based on adaptive CTBN for uncertainty reasoning in sensor networks. *Intelligent Automation & Soft Computing*, 16(3):337–351, 2010.

[64] J. Xu, C. R. Shelton. Intrusion detection using continuous time Bayesian networks. *Journal of Artificial Intelligence Research (JAIR)*, 39(1):745–774, 2010.

[65] J. Xu. *A Continuous Time Bayesian Network Approach for Intrusion Detection.* Doctoral Dissertation, University of California Riverside, 2010.

[66] J. Xu, C. R. Shelton. Continuous Time Bayesian Networks for Host Level Network Intrusion Detection. W. Daelemans, B. Goethals, K. Morik, editors, *Machine Learning and Knowledge Discovery in Databases*, Volume 5212 series *Lecture Notes in Computer Science*, pages 613–627. Springer Berlin / Heidelberg, 2008.

[67] S. Qiao, C. Tang, H. Jin, T. Long, S. Dai, Y. Ku, M. Chau. PutMode: prediction of uncertain trajectories in moving objects databases. *Applied Intelligence*, 33(3):370–386, 2010.

[68] E. Gatti, D. Luciani, F. Stella. A continuous time Bayesian network model for cardiogenic heart failure. *Flexible Services and Manufacturing Journal*, pages 1–20, 2011.

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

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

[71] L. Portinale, D. Codetta-Raiteri. Generalizing continuous time Bayesian networks with immediate nodes. *Workshop on Graph Structures for Knowledge Representation and Reasoning*, 2009.

[72] F. Stella, Y. Amer. Continuous time Bayesian network classifiers. *Journal of Biomedical Informatics*, 2012.

[73] D. Codecasa, F. Stella. Conditional log-likelihood for continuous time Bayesian network classifiers. *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.

[74] S. Villa, F. Stella. A continuous time Bayesian network classifier for intraday FX prediction. *Quantitative Finance*, 2014.

[75] J. Weiss, S. Natarajan, D. Page. Multiplicative forests for continuous-time processes. *Advances in Neural Information Processing Systems 25*, pages 467–475, 2012.

[76] N. Friedman, M. Goldszmidt. Learning Bayesian networks with local structure. *Learning in graphical models*, pages 421–459. Springer, 1998.

[77] K. Gopalratnam, H. Kautz, D. S. Weld. Extending continuous time Bayesian networks. *Proceedings of the National Conference on Artificial Intelligence*, Volume 20, pages 981–986, 2005.

[78] A. Pfeffer. Asynchronous dynamic Bayesian networks. *Proceedings of the 21st Conference on Uncertainty in Artificial Intelligence (UAI)*, 2005.

[79] D. M. Chickering. Learning Bayesian networks is NP-complete. *Learning from Data*, pages 121–130. Springer, 1996.

[80] G. F. Cooper. The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial Intelligence*, 42(2):393–405, 1990.

[81] P. Dagum, M. Luby. Approximating probabilistic inference in Bayesian belief networks is NP-hard. *Artificial Intelligence*, 60(1):141–153, 1993.

[82] H. Boudali, P. Crouzen, M. Stoelinga. Dynamic fault tree analysis using input/output interactive Markov chains. *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*, pages 708–717. IEEE, 2007.

[83] L. Portinale, D. C. Raiteri, S. Montani. Supporting reliability engineers in exploiting the power of dynamic Bayesian networks. *International Journal of Approximate Reasoning*, 51(2):179–195, 2010.

[84] J. Bechta Dugan, S. J. Bavuso, M. A. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Transactions on Reliability*, 41(3):363–377, 1992.

[85] E. Castillo, J. Gutierrez, A. Hadi. Sensitivity analysis in discrete Bayesian networks. *IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans*, 27(4):412–423, 1997.

[86] H. Chan, A. Darwiche. Sensitivity analysis in Bayesian networks: From single to multiple parameters. *Proceedings of the 20th Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI)*, pages 67–75. AUAI Press, 2004.

[87] X. Cao. Potential based sensitivity analysis of Markov chains. *Proceedings of the 35th IEEE Decision and Control*, Volume 1, pages 568–569, 1996.

[88] X. Cao, X. Yuan, L. Qiu. A single sample path-based performance sensitivity formula for Markov chains. *IEEE Transactions on Automatic Control*, 41(12):1814–1817, 1996.

[89] C. G. Cassandras, S. G. Strickland. On-line sensitivity analysis of Markov chains. *IEEE Transactions on Automatic Control*, 34(1):76–86, 1989.

[90] X. Cao, H. Chen. Perturbation realization, potentials, and sensitivity analysis of Markov processes. *IEEE Transactions on Automatic Control*, 42(10):1382–1393, 1997.

[91] X. Cao, Y. Wan. Algorithms for sensitivity analysis of Markov systems through potentials and perturbation realization. *IEEE Transactions on Control Systems Technology*, 6(4):482–494, 1998.

[92] C. Cassandras, S. Strickland. Observable augmented systems for sensitivity analysis of Markov and semi-Markov processes. *IEEE Transactions on Automatic Control*, 34(10):1026–1037, 1989.

[93] C. Cassandras, S. Strickland. An "augmented chain" approach for on-line sensitivity analysis of Markov process. *26th IEEE Conference on Decision and Control*, Volume 26, pages 1873–1878, 1987.

[94] Z. Liu, F. Tu. Single sample path-based sensitivity analysis of Markov processes using uniformization. *IEEE Transactions on Automatic Control*, 44(4):872–875, 1999.

[95] L. Dai, Y. Ho. Structural infinitesimal perturbation analysis (SIPA) for derivative estimation of discrete-event dynamic systems. *IEEE Transactions on Automatic Control*, 40(7):1154–1166, 1995.

[96] Y. Ho, J. Hu. An infinitesimal perturbation analysis algorithm for a multiclass G/G/1 queue. *Operations Research Letters*, 9(1):35–44, 1990.

[97] L. Xia, X. Cao. Relationship between perturbation realization factors with queueing models and Markov models. *IEEE Transactions on Automatic Control*, 51(10):1699–1704, 2006.

[98] A. Geraci, F. Katki, L. McMonegal, B. Meyer, J. Lane, P. Wilson, J. Radatz, M. Yee, H. Porteous, F. Springsteel. *IEEE Standard Computer Dictionary: Compilation of IEEE Standard Computer Glossaries.* IEEE Press, 1991.

[99] M. Reiman, A. Weiss. Sensitivity analysis via likelihood ratios. *Proceedings of the 18th Conference on Winter Simulation*, WSC '86, pages 285–289, New York, NY, USA, 1986. ACM.

[100] M. K. Nakayama, A. Goyal, P. W. Glynn. Likelihood ratio sensitivity analysis for Markovian models of highly dependable systems. *Operations Research*, 42(1):137–157, 1994.

[101] P. Glasserman. Derivative estimates from simulation of continuous-time Markov chains. *Operations Research*, 40:292–308, 1992.

[102] L. Sturlaugson, J. W. Sheppard. Factored performance functions with structural representation in continuous time Bayesian networks. *The Twenty-Seventh International FLAIRS Conference*, 2014.

[103] A. Agogino, K. Goebel. Mill Data Set. *BEST lab, UC Berkeley. NASA Ames Prognostics Data Repository, [http://ti.arc.nasa.gov/project/prognostic-data-repository], NASA Ames, Moffett Field, CA*, 2007.

[104] C. R. Shelton, Y. Fan, W. Lam, J. Lee, J. Xu. Continuous time Bayesian network reasoning and learning engine. *Journal of Machine Learning Research (JMLR)*, 11:1137–1140, 2010.

[105] H. Chan, A. Darwiche. On the revision of probabilistic beliefs using uncertain evidence. *Artificial Intelligence*, 163(1):67–90, 2005.

[106] R. Pan, Y. Peng, Z. Ding. Belief update in Bayesian networks using uncertain evidence. *18th IEEE International Conference on Tools with Artificial Intelligence*, pages 441–444. IEEE, 2006.

[107] J. Bilmes. On virtual evidence and soft evidence in Bayesian networks, UWEETR-2004-0016. Technical Report, Department of Electrical Engineering, University of Washington, 2004.

APPENDICES

APPENDIX A

PUBLICATIONS

Each of the five major contributions in this thesis forms the basis for a paper that has either been accepted, submitted, or will be submitted in the near future to a peer-reviewed conference or journal. The list of papers is as follows.

- L. Sturlaugson, J. W. Sheppard. Inference Complexity in Continuous Time Bayesian Networks. Submitted to *The Thirtieth Conference on Uncertainty in Artificial Intelligence (UAI)*, 2014.

- L. Sturlaugson, J. W. Sheppard. Factored Performance Functions with Structural Representation in Continuous Time Bayesian Networks. Accepted to *The Twenty-Seventh International FLAIRS Conference*, 2014.

- L. Sturlaugson, J. W. Sheppard. Node Isolation for Approximate Inference in Continuous Time Bayesian Networks. To be submitted to *Advances in Neural Information Processing Systems 26 (NIPS)*, 2014.

- L. Sturlaugson, J. W. Sheppard. Sensitivity Analysis of Continuous Time Bayesian Network Reliability Models. Submitted to *SIAM/ASA Journal on Uncertainty Quantification*, 2014.

- L. Sturlaugson, J. W. Sheppard. Uncertain and Negative Evidence in Continuous Time Bayesian Networks. Submitted to *Journal of Artificial Intelligence Research*, 2014.

APPENDIX B

DRUG EFFECT MODEL

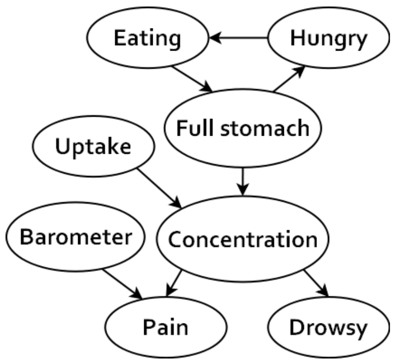


Figure B.1: Drug effect network

These are the initial distributions and conditional intensity matrices for the drug effect network, shown in Figure B.1 and referenced in Sections 2.3.1, 5.3.2, and 7.4. Each node is abbreviated to its first letter.

$$\boldsymbol{P}_E = \begin{pmatrix} 0.1 & 0.9 \end{pmatrix} \quad \boldsymbol{Q}_{E|h_0} = \begin{pmatrix} -0.01 & 0.01 \\ 10 & -10 \end{pmatrix} \quad \boldsymbol{Q}_{E|h_1} = \begin{pmatrix} -2 & 2 \\ 1 & -1 \end{pmatrix}$$

$$\boldsymbol{P}_H = \begin{pmatrix} 0.6 & 0.4 \end{pmatrix}$$

$$\boldsymbol{Q}_{H|f_0} = \begin{pmatrix} -2 & 2 \\ 0.01 & -0.01 \end{pmatrix} \quad \boldsymbol{Q}_{H|f_1} = \begin{pmatrix} -0.01 & 0.01 \\ 0.01 & -0.01 \end{pmatrix} \quad \boldsymbol{Q}_{H|f_2} = \begin{pmatrix} -0.01 & 0.01 \\ 10 & -10 \end{pmatrix}$$

$$\boldsymbol{P}_F = \begin{pmatrix} 0.7 & 0.2 & 0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{F|e_0} = \begin{pmatrix} -0.02 & 0.01 & 0.01 \\ 0.3 & -0.31 & 0.01 \\ 0.01 & 1.0 & -1.01 \end{pmatrix} \quad \boldsymbol{Q}_{F|e_1} = \begin{pmatrix} -6.01 & 6 & 0.01 \\ 0.01 & -3.01 & 3 \\ 0.01 & 0.01 & -0.02 \end{pmatrix}$$

$$\boldsymbol{P}_U = \begin{pmatrix} 0.45 & 0.55 \end{pmatrix} \quad \boldsymbol{Q}_U = \begin{pmatrix} 0 & 0 \\ 0.5 & -0.5 \end{pmatrix}$$

$$\boldsymbol{P}_C = \begin{pmatrix} 0.7 & 0.2 & 0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{C|u_0,f_0} = \begin{pmatrix} -0.02 & 0.01 & 0.01 \\ 0.25 & -0.26 & 0.01 \\ 0.01 & 0.5 & -0.51 \end{pmatrix} \quad \boldsymbol{Q}_{C|u_1,f_0} = \begin{pmatrix} -2.01 & 2 & 0.01 \\ 0.01 & -1.01 & 1 \\ 0.01 & 0.01 & -0.02 \end{pmatrix}$$

$$\boldsymbol{Q}_{C|u_0,f_1} = \begin{pmatrix} -0.02 & 0.01 & 0.01 \\ 0.25 & -0.26 & 0.01 \\ 0.01 & 0.5 & -0.51 \end{pmatrix} \quad \boldsymbol{Q}_{C|u_1,f_1} = \begin{pmatrix} -1.01 & 1 & 0.01 \\ 0.01 & -2.01 & 2 \\ 0.01 & 0.01 & -0.02 \end{pmatrix}$$

$$\boldsymbol{Q}_{C|u_0,f_2} = \begin{pmatrix} -0.02 & 0.01 & 0.01 \\ 0.25 & -0.26 & 0.01 \\ 0.01 & 0.5 & -0.51 \end{pmatrix} \quad \boldsymbol{Q}_{C|u_1,f_2} = \begin{pmatrix} -0.51 & 0.5 & 0.01 \\ 0.01 & -4.01 & 4 \\ 0.01 & 0.01 & -0.02 \end{pmatrix}$$

$$\boldsymbol{P}_B = \begin{pmatrix} 0.3 & 0.7 & 0 \end{pmatrix} \quad \boldsymbol{Q}_B = \begin{pmatrix} -0.21 & 0.2 & 0.01 \\ 0.05 & -0.1 & 0.05 \\ 0.01 & 0.2 & -0.21 \end{pmatrix}$$

$$\boldsymbol{P}_P = \begin{pmatrix} 0.1 & 0.9 \end{pmatrix}$$

$$\boldsymbol{Q}_{P|c_0,b_0} = \begin{pmatrix} -6 & 6 \\ 0.1 & -0.1 \end{pmatrix} \quad \boldsymbol{Q}_{P|c_1,b_0} = \begin{pmatrix} -1 & 1 \\ 0.1 & -0.1 \end{pmatrix} \quad \boldsymbol{Q}_{P|c_2,b_0} = \begin{pmatrix} -6 & 6 \\ 0.1 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{P|c_0,b_1} = \begin{pmatrix} -1 & 1 \\ 0.3 & -0.3 \end{pmatrix} \quad \boldsymbol{Q}_{P|c_1,b_1} = \begin{pmatrix} -0.3 & 0.3 \\ 0.3 & -0.3 \end{pmatrix} \quad \boldsymbol{Q}_{P|c_2,b_1} = \begin{pmatrix} -1 & 1 \\ 0.3 & -0.3 \end{pmatrix}$$

$$\boldsymbol{Q}_{P|c_0,b_2} = \begin{pmatrix} -0.01 & 0.01 \\ 2 & -2 \end{pmatrix} \quad \boldsymbol{Q}_{P|c_1,b_2} = \begin{pmatrix} -0.01 & 0.01 \\ 2 & -2 \end{pmatrix} \quad \boldsymbol{Q}_{P|c_2,b_2} = \begin{pmatrix} -0.01 & 0.01 \\ 2 & -2 \end{pmatrix}$$

$$\boldsymbol{P}_D = \begin{pmatrix} 0.1 & 0.9 \end{pmatrix}$$

$$\boldsymbol{Q}_{D|c_0} = \begin{pmatrix} -0.17 & 0.17 \\ 0.5 & -0.5 \end{pmatrix} \quad \boldsymbol{Q}_{D|c_1} = \begin{pmatrix} -0.33 & 0.33 \\ 0.33 & -0.33 \end{pmatrix} \quad \boldsymbol{Q}_{D|c_2} = \begin{pmatrix} -1 & 1 \\ 0.1 & -0.1 \end{pmatrix}$$

APPENDIX C

VEHICLE RELIABILITY MODEL

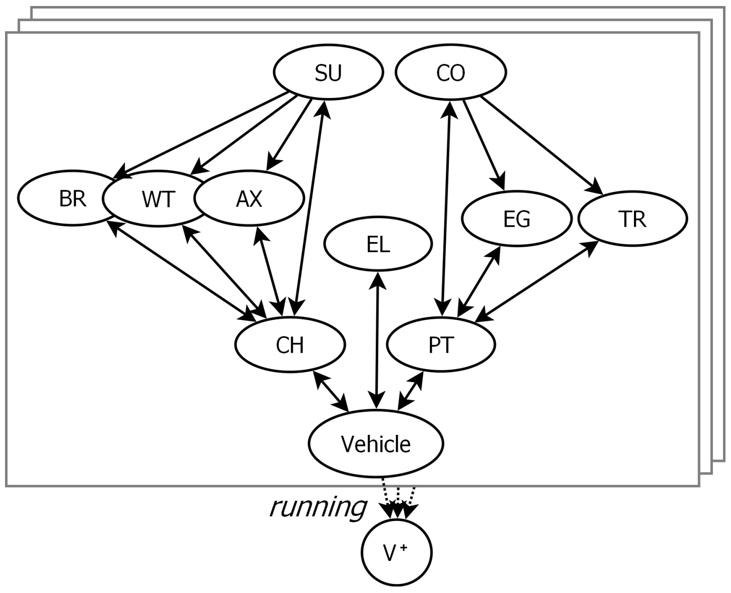


Figure C.1: CTBN for fleet of vehicles with synergy node

These are conditional intensity matrices for the vehicle reliability network, shown in Figure C.1 and referenced in Section 4.3. The notation $\mathbf{Q}_{X|*}$ is used to specify the conditional intensity matrices of X for all state combinations of $\mathbf{Pa}(X)$ that are not specified previously. The initial state of each node is state 0.

$$\boldsymbol{Q}_{CH|br_0,wt_0,ax_0,su_0,v_0} = \begin{pmatrix} 0 & 0 & 0 \\ 1\text{E}10 & -1\text{E}10 & 0 \\ 1\text{E}10 & 0 & -1\text{E}10 \end{pmatrix}$$

$$\boldsymbol{Q}_{CH|br_0,wt_0,ax_0,su_0,v_1} = \begin{pmatrix} -1\text{E}10 & 0 & 1\text{E}10 \\ 0 & -1\text{E}10 & 1\text{E}10 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{CH|*} = \begin{pmatrix} -1\text{E}10 & 1\text{E}10 & 0 \\ 0 & 0 & 0 \\ 0 & 1\text{E}10 & -1\text{E}10 \end{pmatrix}$$

$$\boldsymbol{Q}_{PT|tr_0,eg_0,co_0,v_0} = \begin{pmatrix} 0 & 0 & 0 \\ 1\text{E}10 & -1\text{E}10 & 0 \\ 1\text{E}10 & 0 & -1\text{E}10 \end{pmatrix}$$

$$\boldsymbol{Q}_{PT|tr_0,eg_0,co_0,v_1} = \begin{pmatrix} -1\text{E}10 & 0 & 1\text{E}10 \\ 0 & -1\text{E}10 & 1\text{E}10 \\ 0 & 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{PT|*} = \begin{pmatrix} -1\text{E}10 & 1\text{E}10 & 0 \\ 0 & 0 & 0 \\ 0 & 1\text{E}10 & -1\text{E}10 \end{pmatrix}$$

$$\boldsymbol{Q}_{EL|v_0} = \begin{pmatrix} -0.12\text{E-}3 & 0.12\text{E-}3 \\ 0.8 & -0.8 \end{pmatrix} \quad \boldsymbol{Q}_{EL|v_1} = \begin{pmatrix} 0 & 0 \\ 0.8 & -0.8 \end{pmatrix}$$

$$\boldsymbol{Q}_{SU|ch_0} = \begin{pmatrix} -0.5\text{E-}4 & 0.5\text{E-}4 & 0 \\ 0 & -0.1\text{E-}3 & 0.1\text{E-}3 \\ 3 & 0 & -3 \end{pmatrix}$$

$$\boldsymbol{Q}_{SU|ch_1} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 3 & 0 & -3 \end{pmatrix} \quad \boldsymbol{Q}_{SU|ch_2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 3 & 0 & -3 \end{pmatrix}$$

$$\boldsymbol{Q}_{CO|pt_0} = \begin{pmatrix} -0.75\text{E-}4 & 0.75\text{E-}4 & 0 \\ 0 & -0.15\text{E-}3 & 0.15\text{E-}3 \\ 3.5 & 0 & -3.5 \end{pmatrix}$$

$$\boldsymbol{Q}_{CO|pt_1} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 3.5 & 0 & -3.5 \end{pmatrix} \quad \boldsymbol{Q}_{CO|pt_2} = \begin{pmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 3.5 & 0 & -3.5 \end{pmatrix}$$

$$\boldsymbol{Q}_{TR|pt_0} = \begin{pmatrix} -0.83\text{E-4} & 0.83\text{E-4} \\ 4 & -4 \end{pmatrix}$$

$$\boldsymbol{Q}_{TR|pt_1} = \begin{pmatrix} 0 & 0 \\ 4 & -4 \end{pmatrix} \quad \boldsymbol{Q}_{TR|pt_2} = \begin{pmatrix} 0 & 0 \\ 4 & -4 \end{pmatrix}$$

$$\boldsymbol{Q}_{BR|ch_0,su_0} = \begin{pmatrix} -0.5\text{E-3} & 0.5\text{E-3} \\ 4 & -4 \end{pmatrix} \quad \boldsymbol{Q}_{BR|ch_0,su_1} = \begin{pmatrix} -0.8\text{E-3} & 0.8\text{E-3} \\ 4 & -4 \end{pmatrix}$$

$$\boldsymbol{Q}_{BR|ch_0,su_2} = \begin{pmatrix} -0.8\text{E-3} & 0.8\text{E-3} \\ 4 & -4 \end{pmatrix} \quad \boldsymbol{Q}_{BR|*} = \begin{pmatrix} 0 & 0 \\ 4 & -4 \end{pmatrix}$$

$$\boldsymbol{Q}_{WT|ch_0,su_0} = \begin{pmatrix} -0.16\text{E-3} & 0.16\text{E-3} \\ 1.2 & -1.2 \end{pmatrix} \quad \boldsymbol{Q}_{WT|ch_0,su_1} = \begin{pmatrix} -0.2\text{E-3} & 0.2\text{E-3} \\ 1.2 & -1.2 \end{pmatrix}$$

$$\boldsymbol{Q}_{WT|ch_0,su_2} = \begin{pmatrix} -0.2\text{E-3} & 0.2\text{E-3} \\ 1.2 & -1.2 \end{pmatrix} \quad \boldsymbol{Q}_{WT|*} = \begin{pmatrix} 0 & 0 \\ 1.2 & -1.2 \end{pmatrix}$$

$$\boldsymbol{Q}_{AX|ch_0,su_0} = \begin{pmatrix} -0.5\text{E-3} & 0.5\text{E-3} \\ 8 & -8 \end{pmatrix} \quad \boldsymbol{Q}_{AX|ch_0,su_1} = \begin{pmatrix} -0.7\text{E-3} & 0.7\text{E-3} \\ 8 & -8 \end{pmatrix}$$

$$\boldsymbol{Q}_{AX|ch_0,su_2} = \begin{pmatrix} -0.2\text{E-3} & 0.2\text{E-3} \\ 8 & -8 \end{pmatrix} \quad \boldsymbol{Q}_{AX|*} = \begin{pmatrix} 0 & 0 \\ 8 & -8 \end{pmatrix}$$

$$\boldsymbol{Q}_{EG|pt_0,co_0} = \begin{pmatrix} -0.23\text{E-4} & 0.23\text{E-4} \\ 10 & -10 \end{pmatrix} \quad \boldsymbol{Q}_{EG|pt_0,co_1} = \begin{pmatrix} -0.35\text{E-4} & 0.35\text{E-4} \\ 10 & -10 \end{pmatrix}$$

$$\boldsymbol{Q}_{EG|pt_0,co_2} = \begin{pmatrix} -0.35\text{E-4} & 0.35\text{E-4} \\ 10 & -10 \end{pmatrix} \quad \boldsymbol{Q}_{EG|*} = \begin{pmatrix} 0 & 0 \\ 10 & -10 \end{pmatrix}$$

$$\boldsymbol{Q}_{V|ch_0,pt_0,el_0} = \begin{pmatrix} 0 & 0 \\ 1\text{E}10 & -1\text{E}10 \end{pmatrix} \quad \boldsymbol{Q}_{V|ch_0,pt_0,el_0} = \begin{pmatrix} 0 & 0 \\ 1\text{E}10 & -1\text{E}10 \end{pmatrix}$$

$$\boldsymbol{Q}_{V|ch_2,pt_0,el_0} = \begin{pmatrix} 0 & 0 \\ 1\text{E}10 & -1\text{E}10 \end{pmatrix} \quad \boldsymbol{Q}_{V|ch_2,pt_2,el_0} = \begin{pmatrix} 0 & 0 \\ 1\text{E}10 & -1\text{E}10 \end{pmatrix}$$

$$\boldsymbol{Q}_{V|*} = \begin{pmatrix} -1\text{E}10 & 1\text{E}10 \\ 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{V^+|\text{all good}} = \begin{pmatrix} 0 & 0 \\ 1\text{E}10 & -1\text{E}10 \end{pmatrix} \quad \boldsymbol{Q}_{V^+|*} = \begin{pmatrix} -1\text{E}10 & 1\text{E}10 \\ 0 & 0 \end{pmatrix}$$

APPENDIX D

SIMPLE SYNTHETIC MODEL

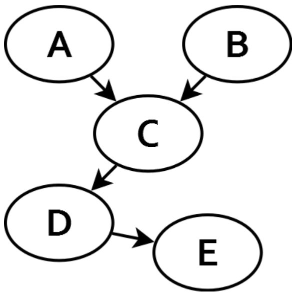


Figure D.1: Example CTBN of a simple system

These are conditional intensity matrices for the simple, synthetic reliability network, shown in Figure D.1 and referenced in Section 5.3.1 and Section 6.4.1. Each node uses a uniform initial distribution.

$$\boldsymbol{Q}_A = \begin{pmatrix} -3 & 1 & 2 \\ 1 & -2 & 1 \\ 2 & 0 & -2 \end{pmatrix} \qquad \boldsymbol{Q}_B = \begin{pmatrix} -1 & 1 \\ 2 & -2 \end{pmatrix}$$

$$\boldsymbol{Q}_{C|a_0,b_0} = \begin{pmatrix} -2 & 1 & 1 \\ 2 & -3 & 1 \\ 1 & 1 & -2 \end{pmatrix} \quad \boldsymbol{Q}_{C|a_1,b_0} = \begin{pmatrix} -3 & 2 & 1 \\ 2 & -4 & 2 \\ 2 & 1 & -3 \end{pmatrix}$$

$$\boldsymbol{Q}_{C|a_2,b_0} = \begin{pmatrix} -5 & 4 & 1 \\ 2 & -5 & 3 \\ 3 & 1 & -4 \end{pmatrix} \quad \boldsymbol{Q}_{C|a_0,b_1} = \begin{pmatrix} -6 & 4 & 2 \\ 2 & -6 & 4 \\ 4 & 1 & -5 \end{pmatrix}$$

$$\boldsymbol{Q}_{C|a_1,b_1} = \begin{pmatrix} -7 & 6 & 1 \\ 2 & -7 & 5 \\ 5 & 1 & -6 \end{pmatrix} \quad \boldsymbol{Q}_{C|a_2,b_1} = \begin{pmatrix} -2 & 1 & 1 \\ 2 & -3 & 1 \\ 1 & 6 & -7 \end{pmatrix}$$

$$\boldsymbol{Q}_{D|c_0} = \begin{pmatrix} -1 & 1 \\ 2 & -2 \end{pmatrix} \quad \boldsymbol{Q}_{D|c_1} = \begin{pmatrix} -3 & 3 \\ 1 & -1 \end{pmatrix} \quad \boldsymbol{Q}_{D|c_2} = \begin{pmatrix} -3 & 3 \\ 2 & -2 \end{pmatrix}$$

$$\boldsymbol{Q}_{E|d_0} = \begin{pmatrix} -0.01 & 0.01 \\ 1 & -1 \end{pmatrix} \quad \boldsymbol{Q}_{E|d_1} = \begin{pmatrix} -0.1 & 0.1 \\ 1 & -1 \end{pmatrix}$$

APPENDIX E

CARDIAC ASSIST SYSTEM MODEL

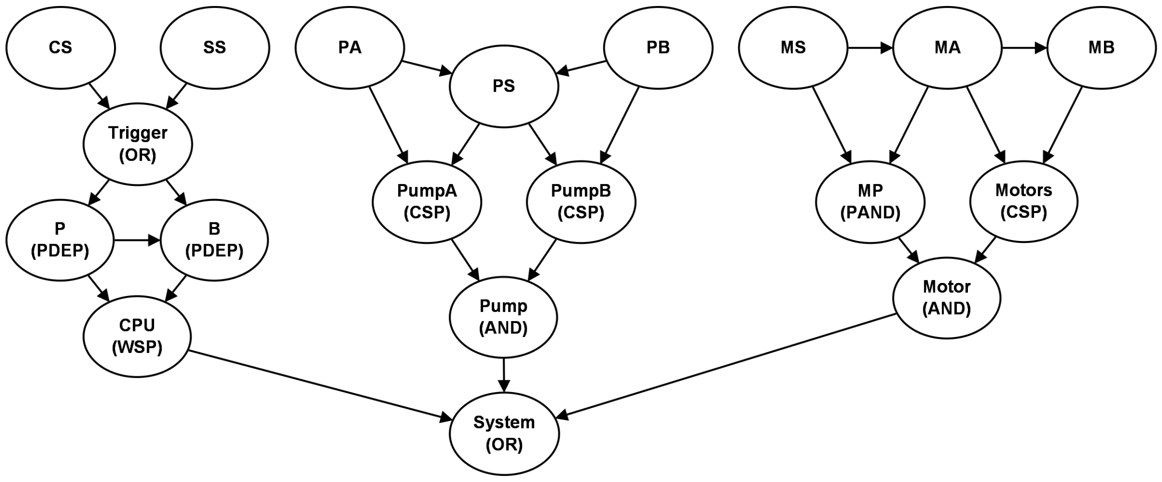


Figure E.1: Cardiac assist system model

These are conditional intensity matrices for the reliability model of the cardiac assist system shown in Figure E.1 and referenced in Sections 5.3.4 and 6.4.2. The notation $\mathbf{Q}_{X|*}$ is used to specify the conditional intensity matrices of X for all state combinations of $\mathbf{Pa}(X)$ that are not specified previously. Because the model is used for long-term reliability analysis, uniform initial distributions were used for all nodes.

$$\boldsymbol{Q}_{B|}\begin{Bmatrix} p_1, trigger_0 \\ p_1, trigger_2 \end{Bmatrix} = \begin{pmatrix} -5\text{E-}4 & 5\text{E-}4 \\ 0.1 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{B|}\begin{Bmatrix} p_0, trigger_1 \\ p_1, trigger_1 \\ p_2, trigger_1 \end{Bmatrix} = \begin{pmatrix} -1\text{E+}20 & 1\text{E+}20 \\ 0.1 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{B|*} = \begin{pmatrix} -2.5\text{E-}4 & 2.5\text{E-}4 \\ 0.1 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{CPU|}\begin{Bmatrix} b_0, p_0 \\ b_0, p_1 \\ b_1, p_0 \end{Bmatrix} = \begin{pmatrix} 0 & 0 \\ 1\text{E}{+}20 & -1\text{E}{+}20 \end{pmatrix}$$

$$\boldsymbol{Q}_{CPU|*} = \begin{pmatrix} -1\text{E}{+}20 & 1\text{E}{+}20 \\ 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{CS} = \begin{pmatrix} -2\text{E-}4 & 2\text{E-}4 \\ 0.1 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{MA|ms_0} = \begin{pmatrix} -1\text{E-}3 & 1\text{E-}3 & 0 & 0 & 0 \\ 0.1 & -0.1 & 0 & 0 & 0 \\ 0.1 & 0 & -0.1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{MA|*} = \begin{pmatrix} -1\text{E-}3 & 0 & 0 & 1\text{E-}3 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0.1 & 0 & 0 & -0.1 & 0 \\ 0.1 & 0 & 0 & 0 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{MB|}\begin{Bmatrix} ma_1 \\ ma_3 \end{Bmatrix} = \begin{pmatrix} -1\text{E-}3 & 1\text{E-}3 \\ 0.1 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{MB|*} = \begin{pmatrix} 0 & 0 \\ 0.1 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{Motor|motors_1,mp_1} = \begin{pmatrix} -1\text{E}{+}20 & 1\text{E}{+}20 \\ 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{Motor|*} = \begin{pmatrix} 0 & 0 \\ 1\text{E}{+}20 & -1\text{E}{+}20 \end{pmatrix}$$

$$\boldsymbol{Q}_{Motors|}\left\{\begin{matrix} ma_0, mb_0 \\ ma_0, mb_1 \\ ma_1, mb_0 \\ ma_3, mb_0 \end{matrix}\right\} = \begin{pmatrix} 0 & 0 \\ 1\text{E}{+}20 & -1\text{E}{+}20 \end{pmatrix}$$

$$\boldsymbol{Q}_{Motors|*} = \begin{pmatrix} -1\text{E}{+}20 & 1\text{E}{+}20 \\ 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{MP|ma_0,ms_0} = \begin{pmatrix} 0 & 0 \\ 1\text{E}{+}20 & -1\text{E}{+}20 \end{pmatrix}$$

$$\boldsymbol{Q}_{MP|}\left\{\begin{matrix} ma_1, ms_1 \\ ma_2, ms_1 \end{matrix}\right\} = \begin{pmatrix} -1\text{E}{+}20 & 1\text{E}{+}20 \\ 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{MP|*} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{MS} = \begin{pmatrix} -1\text{E-}5 & 1\text{E-}5 \\ 0.1 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{P|trigger_1} = \begin{pmatrix} -1\text{E+}20 & 9\text{E+}19 & 1\text{E+}19 \\ 0.1 & -0.1 & 0 \\ 0.1 & 0 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{P|*} = \begin{pmatrix} -0.1 & 0.09 & 0.01 \\ 0.1 & -0.1 & 0 \\ 0.1 & 0 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{PA} = \begin{pmatrix} -1\text{E-}3 & 8\text{E-}4 & 2\text{E-}4 \\ 0.1 & -0.1 & 0 \\ 0.1 & 0 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{PB} = \begin{pmatrix} -1\text{E-}3 & 8\text{E-}4 & 2\text{E-}4 \\ 0.1 & -0.1 & 0 \\ 0.1 & 0 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{PS|}\begin{Bmatrix} pa_0, pb_0 \\ pa_0, pb_2 \\ pa_1, pb_2 \end{Bmatrix} = \begin{pmatrix} -5\text{E-}4 & 5\text{E-}4 \\ 1\text{E-}4 & -1\text{E-}4 \end{pmatrix}$$

$$\boldsymbol{Q}_{PS|}\left\{\begin{matrix} pa_0, pb_1 \\ pa_1, pb_0 \\ pa_1, pb_1 \end{matrix}\right\} = \begin{pmatrix} -1\text{E-}3 & 1\text{E-}3 \\ 1\text{E-}4 & -1\text{E-}4 \end{pmatrix}$$

$$\boldsymbol{Q}_{PS|*} = \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{Pump|pumpa_1,pumpb_1} = \begin{pmatrix} -1\text{E+}20 & 1\text{E+}20 \\ 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{Pump|*} = \begin{pmatrix} 0 & 0 \\ 1\text{E+}20 & -1\text{E+}20 \end{pmatrix}$$

$$\boldsymbol{Q}_{PumpA|}\left\{\begin{matrix} pa_0, ps_0 \\ pa_0, ps_1 \\ pa_1, ps_0 \end{matrix}\right\} = \begin{pmatrix} 0 & 0 \\ 1\text{E+}20 & -1\text{E+}20 \end{pmatrix}$$

$$\boldsymbol{Q}_{PumpA|*} = \begin{pmatrix} -1\text{E+}20 & 1\text{E+}20 \\ 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{PumpB|}\left\{\begin{matrix} pb_0, ps_0 \\ pb_0, ps_1 \\ pb_1, ps_0 \end{matrix}\right\} = \begin{pmatrix} 0 & 0 \\ 1\text{E+}20 & -1\text{E+}20 \end{pmatrix}$$

$$\boldsymbol{Q}_{PumpB|*} = \begin{pmatrix} -1\text{E}{+}20 & 1\text{E}{+}20 \\ 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{SS} = \begin{pmatrix} -2\text{E-}4 & 2\text{E-}4 \\ 0.1 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{System|cpu_0,motor_0,pump_0} = \begin{pmatrix} 0 & 0 \\ 1\text{E}{+}20 & -1\text{E}{+}20 \end{pmatrix}$$

$$\boldsymbol{Q}_{System|*} = \begin{pmatrix} -1\text{E}{+}20 & 1\text{E}{+}20 \\ 0 & 0 \end{pmatrix}$$

$$\boldsymbol{Q}_{Trigger|cs_0,ss_0} = \begin{pmatrix} 0 & 0 & 0 \\ 1\text{E}{+}20 & -1\text{E}{+}20 & 0 \\ 1\text{E}{+}20 & 0 & -1\text{E}{+}20 \end{pmatrix}$$

$$\boldsymbol{Q}_{Trigger|*} = \begin{pmatrix} -1\text{E}{+}20 & 9\text{E}{+}19 & 1\text{E}{+}19 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{pmatrix}$$

APPENDIX F

MILLING MACHINE MODEL

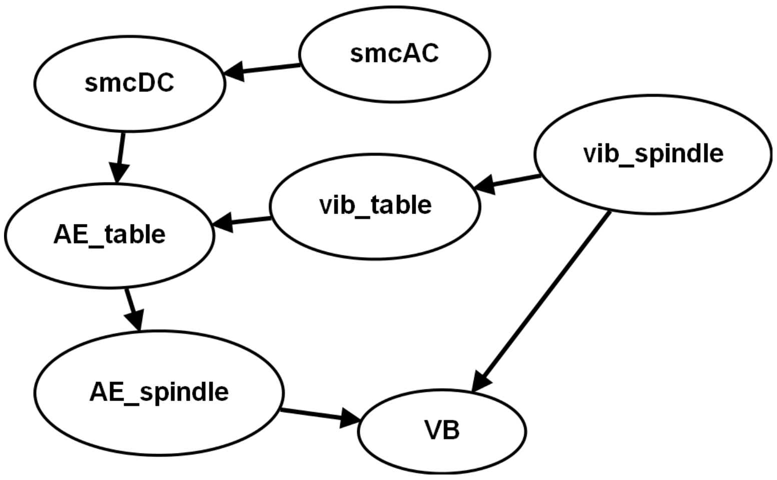


Figure F.1: Milling machine model.

These are conditional intensity matrices learned for the milling machine reliability network, shown in Figure F.1 and referenced in Section 6.4.3. The notation $\mathbf{Q}_{X|*}$ is used to specify the conditional intensity matrices of X for all state combinations of $\mathbf{Pa}(X)$ that are not specified previously. Because the model is used for long-term reliability analysis, uniform initial distributions were used for all nodes.

$$\boldsymbol{Q}_{smcAC} = \begin{pmatrix} -0.1336 & 0.1316 & 0.0020 \\ 0.1340 & -0.2669 & 0.1329 \\ 0.0022 & 0.1309 & -0.1331 \end{pmatrix}$$

$$\boldsymbol{Q}_{smcDC|smcAC_0} = \begin{pmatrix} -0.0095 & 0.0095 & 0 \\ 0.0043 & -0.0145 & 0.0102 \\ 0 & 0.0091 & -0.0091 \end{pmatrix}$$

$$\boldsymbol{Q}_{smcDC|smcAC_1} = \begin{pmatrix} -0.0017 & 0.0017 & 0 \\ 0.0048 & -0.0100 & 0.0052 \\ 0.0003 & 0.0075 & -0.0078 \end{pmatrix}$$

$$\boldsymbol{Q}_{smcDC|smcAC_2} = \begin{pmatrix} -0.1444 & 0.1302 & 0.0142 \\ 0.1707 & -0.3207 & 0.1500 \\ 0.0019 & 0.2636 & -0.2655 \end{pmatrix}$$

$$\boldsymbol{Q}_{vib\_table|vib\_spindle_0} = \begin{pmatrix} -0.1444 & 0.1302 & 0.0142 \\ 0.1707 & -0.3207 & 0.1500 \\ 0.0019 & 0.2636 & -0.2655 \end{pmatrix}$$

$$\boldsymbol{Q}_{vib\_table|vib\_spindle_1} = \begin{pmatrix} -0.1650 & 0.1330 & 0.0320 \\ 0.1679 & -0.3391 & 0.1712 \\ 0.0044 & 0.2363 & -0.2408 \end{pmatrix}$$

$$\boldsymbol{Q}_{vib\_table|vib\_spindle_2} = \begin{pmatrix} -0.2114 & 0.1644 & 0.0470 \\ 0.1318 & -0.3374 & 0.2056 \\ 0.0026 & 0.1516 & -0.1542 \end{pmatrix}$$

$$\boldsymbol{Q}_{vib\_spindle} = \begin{pmatrix} -0.1897 & 0.1784 & 0.0113 \\ 0.1846 & -0.3399 & 0.1553 \\ 0.0033 & 0.1651 & -0.1684 \end{pmatrix}$$

$$\boldsymbol{Q}_{AE\_table|smcDC_0,vib\_table_0} = \begin{pmatrix} -0.0266 & 0.0238 & 0.0028 \\ 0.4166 & -0.4919 & 0.0753 \\ 0.0078 & 0.6708 & -0.6786 \end{pmatrix}$$

$$\boldsymbol{Q}_{AE\_table|smcDC_0,vib\_table_1} = \begin{pmatrix} -0.1680 & 0.1630 & 0.0050 \\ 0.1279 & -0.3249 & 0.1970 \\ 0 & 0.2450 & -0.2450 \end{pmatrix}$$

$$\boldsymbol{Q}_{AE\_table|smcDC_0,vib\_table_2} = \begin{pmatrix} -0.0776 & 0.0634 & 0.0142 \\ 0.0639 & -0.3264 & 0.2625 \\ 0 & 0.0891 & -0.0891 \end{pmatrix}$$

$$\boldsymbol{Q}_{AE\_table|smcDC_1,vib\_table_0} = \begin{pmatrix} -0.1505 & 0.1403 & 0.0102 \\ 0.3203 & -0.3783 & 0.0580 \\ 0.0027 & 0.4714 & -0.4741 \end{pmatrix}$$

$$\boldsymbol{Q}_{AE\_table|smcDC_1,vib\_table_1} = \begin{pmatrix} -0.2397 & 0.2314 & 0.0083 \\ 0.1648 & -0.3047 & 0.1399 \\ 0.0001 & 0.3013 & -0.3014 \end{pmatrix}$$

$$\boldsymbol{Q}_{AE\_table|smcDC_1,vib\_table_2} = \begin{pmatrix} -0.5263 & 0.3723 & 0.1540 \\ 0.0961 & -0.3549 & 0.2588 \\ 0.0001 & 0.1511 & -0.1512 \end{pmatrix}$$

$$\boldsymbol{Q}_{AE\_table|smcDC_2,vib\_table_0} = \begin{pmatrix} -0.3948 & 0.3439 & 0.0509 \\ 0.1905 & -0.2923 & 0.1017 \\ 0.0004 & 0.2823 & -0.2827 \end{pmatrix}$$

$$\boldsymbol{Q}_{AE\_table|smcDC_2,vib\_table_1} = \begin{pmatrix} -0.4187 & 0.3683 & 0.0504 \\ 0.0997 & -0.2679 & 0.1682 \\ 0 & 0.2837 & -0.2837 \end{pmatrix}$$

$$\boldsymbol{Q}_{AE\_table|smcDC_2,vib\_table_2} = \begin{pmatrix} -0.7074 & 0.6061 & 0.1013 \\ 0.0807 & -0.3982 & 0.3175 \\ 0 & 0.1230 & -0.1230 \end{pmatrix}$$

$$\boldsymbol{Q}_{AE\_spindle|AE\_table_0} = \begin{pmatrix} -0.0410 & 0.0400 & 0.0010 \\ 0.4733 & -0.5141 & 0.0408 \\ 0.0037 & 0.9298 & -0.9335 \end{pmatrix}$$

$$\boldsymbol{Q}_{AE\_spindle|AE\_table_1} = \begin{pmatrix} -0.4894 & 0.4692 & 0.0202 \\ 0.1164 & -0.2021 & 0.0857 \\ 0.0001 & 0.4311 & -0.4312 \end{pmatrix}$$

$$\boldsymbol{Q}_{AE\_spindle|AE\_table_2} = \begin{pmatrix} -3.8803 & 2.9045 & 0.9758 \\ 0.0147 & -0.4459 & 0.4312 \\ 0 & 0.07183 & -0.07183 \end{pmatrix}$$

$$\boldsymbol{Q}_{VB|vib\_spindle_0,AE\_spindle_0} = \begin{pmatrix} -1.30062\text{E-}4 & 1.30062\text{E-}4 \\ 0.1 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{VB|vib\_spindle_0,AE\_spindle_1} = \begin{pmatrix} -5.46581\text{E-}5 & 5.46581\text{E-}5 \\ 0.1 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{VB|vib\_spindle_0,AE\_spindle_2} = \begin{pmatrix} -2.07019\text{E-}5 & 2.07019\text{E-}5 \\ 0.1 & -0.1 \end{pmatrix}$$

$$\boldsymbol{Q}_{VB|*} = \begin{pmatrix} -1\text{E-}6 & 1\text{E-}6 \\ 0.1 & -0.1 \end{pmatrix}$$