SIAM/ASA J. UNCERTAINTY QUANTIFICATION
Vol. xx, pp. x

Sensitivity Analysis of Continuous Time Bayesian Network Reliability Models

Liessman Sturlaugson* and John W. Sheppard*

Abstract. We show how to perform sensitivity analysis on continuous time Bayesian networks (CTBNs) as applied specifically to reliability models. Sensitivity analysis of these models can be used, for example, to measure how uncertainty in the failure rates impact the reliability of the modeled system. The CTBN can be thought of as a type of factored Markov process that separates a system into a set of interdependent subsystems. The factorization allows CTBNs to model more complex systems than single Markov processes. However, the state-space of the CTBN is exponential in the number of subsystems. Therefore, existing methods for sensitivity analysis of Markov processes, when applied directly to the CTBN, become intractable. Sensitivity analysis of CTBNs, while borrowing from techniques for Markov processes, must be adapted to take advantage of the factored nature of the network if it is to remain feasible. To address this, we show how to extend the perturbation realization method for Markov processes to the CTBN. We show how to exploit the conditional independence structure of the CTBN to perform perturbation realization separately for different subnetworks, making the technique able to handle larger networks. This in turn allows the CTBN to model more complex systems while keeping sensitivity analysis of the model tractable.

Key words. Continuous time Bayesian networks, sensitivity analysis, perturbation realization, reliability

AMS subject classifications.

1. Introduction.

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.

Research has been done on sensitivity analysis for many probabilistic networks, such as Bayesian networks [11, 12], Markov chains [4, 7, 10], Markov processes [5, 6, 9, 8, 22], and queuing networks [14, 20, 34]. To the authors' knowledge, methods for sensitivity analysis have not yet been researched specifically for the continuous time Bayesian network (CTBN) model.

CTBNs have found applications in various dynamic domains. They have been used to reason about users' presence and activities in computer applications [25], to model social network dynamics [17], to detect both network and host intrusions in computer systems [35], and to diagnose cardiogenic heart failure [18]. Recently, the CTBN has been leveraged specifically for systems reliability modeling [3].

Extending sensitivity analysis to CTBNs enables one to test how changes to the network parameters affect the expected cost/reward per unit time of the modeled system. Reliability measures, such as mean time between failures (MTBF) and availability, can be attached to states of the CTBN, and sensitivity analysis can then be performed on these measures.

*Department of Computer Science, Montana State University, Bozeman, Montana. (liessman.sturlaugson@cs.montana.edu, john.sheppard@cs.montana.edu)

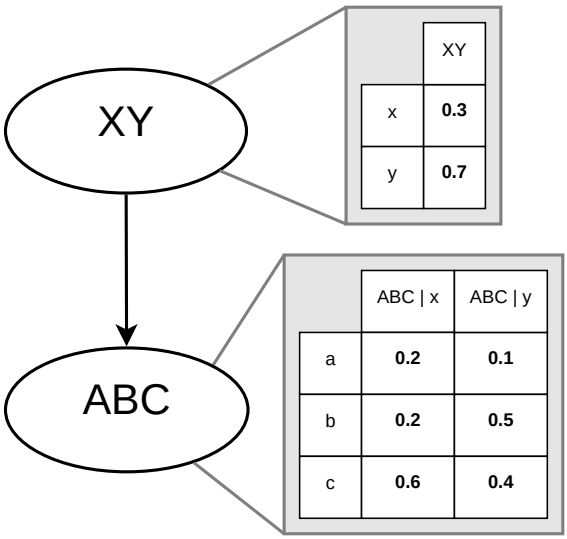

Figure 1. *Example two-node BN.*

2. Model Background. The CTBN borrows ideas from the Bayesian network model while functioning as a Markov process. In this section we introduce the background necessary for understanding the CTBN model. First, we define the Bayesian network. Second, we describe a temporal version of the BN model called the dynamic Bayesian network. Third, we describe the Markov process formulation upon which CTBNs are built. With this background in both Bayesian networks and Markov processes, we can define and describe the CTBN. We conclude the section by differentiating the CTBN from the DBN.

2.1. Bayesian Networks. A Bayesian network (BN) $\mathcal{B}$ is a probabilistic graphical model that uses nodes and arcs in a directed acyclic graph to represent a joint probability distribution over a set of random variables [21]. Let $P(\mathbf{X})$ be a joint probability distribution over n variables $X_1, \ldots, X_n \in \mathbf{X}$. Each variable X_i is represented by a node in the graph. $\mathbf{Pa}(X_i)$ denotes the parents of X_i in the graph. The structure of the network factors the joint probability distribution as:

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

The motivation of a BN is to simplify the joint probability distribution of multiple random variables into a smaller set of conditional dependencies that can be represented and reasoned over efficiently. Figure 1 shows an example two-node BN. As denoted by the arc and the conditional probability table, the probability distribution of ABC is conditionally dependent on the state of XY.

2.2. Dynamic Bayesian Network. The dynamic Bayesian network (DBN) is a special type of BN for temporal modeling and reasoning. The DBN contains a sequence of time-slices, each of which contains a copy of a regular BN $\mathbf{X}(t)$, as formulated above, but now indexed

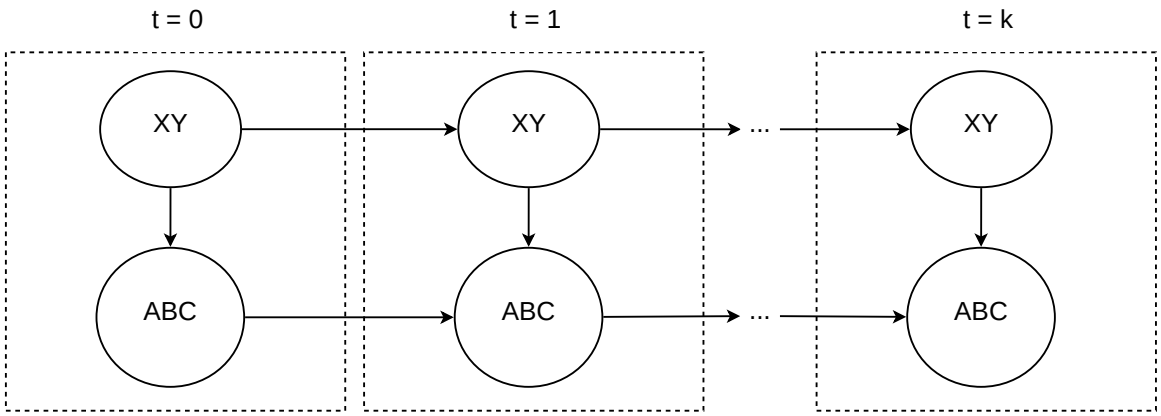

Figure 2. *Example two-node DBN.*

by time t. The probability distribution of a variable at each time-slice can be conditionally dependent on variables throughout any number of previous time-slices. In first-order DBNs, the nodes in each time-slice are not conditionally dependent on any nodes further back than the immediately previous time-slice. Therefore, the joint probability distribution for a first-order DBN factors as:

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

The conditional probability tables of the DBN can be made compact 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" into $\mathbf{X}(1), \mathbf{X}(2), \ldots, \mathbf{X}(k)$ for k time-slices. Figure 2 shows the example BN from Figure 1 extended as a DBN. The nodes in each time-slice are conditionally dependent on the nodes of the previous time-slice, allowing the probability distributions to evolve through time.

2.3. Markov Processes. Although there are variations and extensions of the Markov process model, the CTBN model uses the formulation defined in this section. A finite-state, continuous-time, homogeneous Markov process X with a state space of size n indexed by $\sigma = \{1, 2, \ldots, n\}$ is defined by:

- An initial probability distribution P_X^0 over the n states.
- An $n \times n$ transition intensity matrix $\mathbf{A}_X$, in which each entry $a_{i,j} \geq 0$, $i \neq j$ gives the transition intensity of the process moving from state i to state j, and each entry $a_{i,i} < 0$ controls the amount of time the process remains in state i.

With the diagonal entries constrained to be negative, the probability density function for the process remaining in state i is given by $|a_{i,i}| \exp(a_{i,i}t)$, 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 $1/|a_{i,i}|$. Each row is constrained to sum to zero, $\sum_j a_{i,j} = 0 \; \forall \; i$, meaning that the transition probabilities from state i can be calculated as $a_{i,j}/|a_{i,i}| \; \forall \; j, \; i \neq j$. Figure 3 shows an example three-state Markov process.

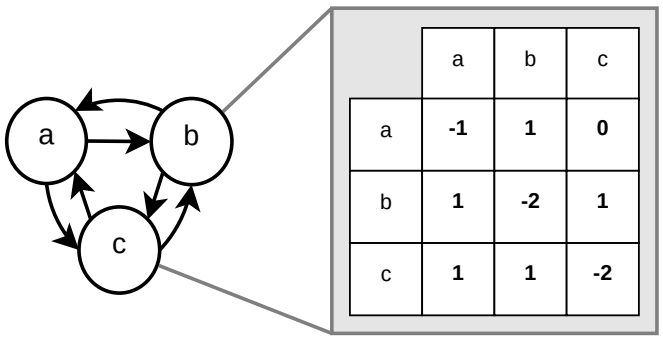

Figure 3. *Example three-state Markov process.*

2.4. Continuous Time Bayesian Networks. The CTBN was first presented in [27] and further defined in [24]. The CTBN uses a set of conditional Markov processes and is structured like a BN, in which the topology of the network encodes conditional independence/dependence relationships among the nodes. However, instead of discrete or continuous random variables, each node is a Markov process. Furthermore, child nodes are *conditional* Markov processes, a type of non-homogeneous Markov process whose transition intensities vary, not as a function of time, but based on the current states of the node's parents in the 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 states. Formally, a continuous time Bayesian network $\mathcal{C}$ over $\mathbf{X}$ consists of two components. The first is an initial distribution denoted P_X^0 over $\mathbf{X}$, which can be specified as a Bayesian network $\mathcal{B}$. This distribution P_X^0 is used for determining the probability distributions for the initial states of the process. The second is a continuous-time transition model, which describes the evolution of the process from its initial distribution, specified as:

- A directed graph $\mathcal{G}$ with nodes $X_1, X_2, \ldots, X_n$, where $\mathbf{Pa}(X_i)$ denotes the parents of X_i in $\mathcal{G}$ (likewise $\mathbf{Ch}(X_i)$ denotes the children of X_i),
- A set of conditional intensity matrices $\mathbf{A}_{X|\mathbf{Pa}(X)}$ associated with X for each possible state instantiation of $\mathbf{Pa}(X)$.

Figure 4 shows an example two-node CTBN. The child node ABC has two intensity matrices, one for each state instantiation of its parent XY.

2.5. CTBN vs. DBN. Similar to the BN/DBN, the conditional dependencies in a CTBN allow a more compact representation for the model. Like a BN, the local conditionally dependent probability tables can be combined to form the full joint probability distribution. In the case of the CTBN, this is called the full joint intensity matrix, which describes the evolution of the entire process. However, just as in the DBN, 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 nodes for the CTBN. Thus, for both the DBN and the CTBN, tractable reasoning over the models requires approaches that can overcome this problem, such as reasoning over smaller subnetworks and combining the results.

Despite the similarities in motivation and representation, the CTBN model is fundamen-

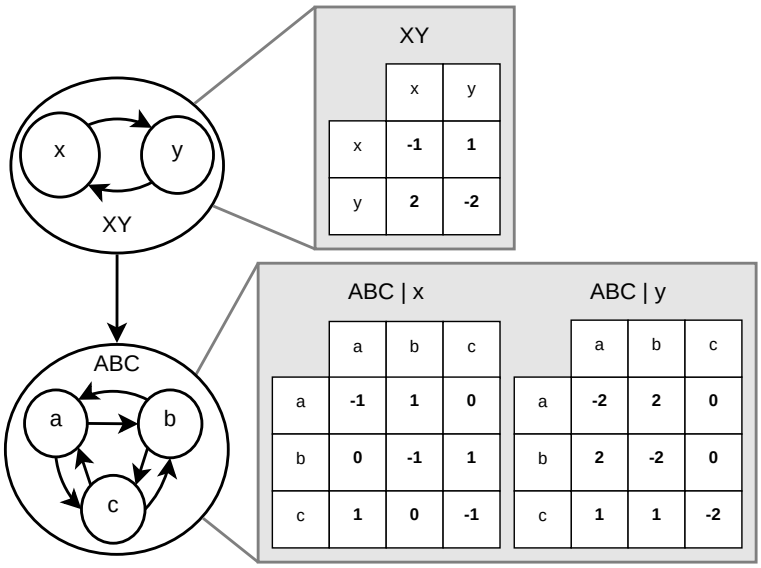

Figure 4. *Example two-node CTBN.*

tally different than the DBN model. Although the network topologies for both models encode conditional independence, the models are differentiated by what the nodes represent. Whereas the nodes in a DBN are 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 will still not be continuous with respect to time. In fact, a DBN can become asymptotically equivalent to a CTBN only as the interval of time between time-slices approaches zero [13].

2.6. CTBNs for Reliability Modeling. 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, 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 research of [3] shows how the CTBN can be specialized for reliability modeling. In particular, it is shown how the CTBN is able to encode Dynamic Fault Trees (DFTs), including intensity matrices to represent gates for AND, OR, warm spare (WSP), sequence enforcing (SEQ), probabilistic dependency (PDEP), and priority AND (PAND). Furthermore, the CTBN is able to represent multi-state interaction, whereas the DFT cannot. In that work, the CTBN models for three use cases are presented, showing different repair policies and showing how inference over the CTBN is able to perform reliability analysis.

The research presented in this paper shows how to perform such reliability analysis more efficiently, taking advantage of the factored nature of the network and calculating what are called potentials, which can be re-used for further 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 divide a network into subnetworks and only update the potentials when necessary.

3. Sensitivity Analysis Background. While the CTBN factors its state-space in much the same way as a BN, it functions as one large Markov process. Thus, sensitivity analysis algorithms for Markov processes can be applied to CTBNs. Although there are several methods for sensitivity analysis of Markov process, the research presented here builds upon perturbation realization, explained in the following section.

3.1. Perturbation Realization. Perturbation analysis using a single sample path $\mathcal{S}$ of an ergodic Markov process is discussed by Cao and Chen in [5], where they study the sensitivity of the steady-state performance of a Markov process with respect to its intensity matrix. Because they are interested in the steady-state performance, regardless of the Markov process's initial state, they only need to use a single sample path, provided the path 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.

3.1.1. Performance Measure. Let $f : \sigma \to \mathbb{R}$ (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,

$$\eta = \sum_{i \in \sigma} \pi_i f(i) = \boldsymbol{\pi}\mathbf{f}, \tag{3.1}$$

where $\boldsymbol{\pi} = (\pi_1, \pi_2, \ldots, \pi_n)$ is the row vector representing the steady-state distribution of $\mathbf{A}_X$ and $\mathbf{f} = (f(1), f(2), \ldots, f(n))^T$ 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 [31]. 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. Although the performance function is more general than the application of this paper, it can be used to represent reliability measures. For example, by setting the performance value of the failed state to 1, the performance measure matches the steady-state probability of the failed state, which represents system unavailability. Sensitivity analysis can then be used to measure the sensitivity of availability on other network parameters, such as individual component failure/repair rates.

3.1.2. Partial Derivatives. The method of perturbation realization uses the idea of partial derivatives of the Markov process's performance measure with respect to changes in the process's intensity matrix. Suppose that the intensity matrix $\mathbf{A}$ of a process changed to $\mathbf{A}_\epsilon = \mathbf{A} + \epsilon\mathbf{Q}$ with ϵ being an arbitrarily small positive number and with $\mathbf{Q}$ being an $n \times n$ matrix (referred to as the Q matrix) such that $\mathbf{A}_\epsilon$ is also a valid intensity matrix (rows sum to zero, etc.). In other words, $\mathbf{Q}$ perturbs the values of $\mathbf{A}$ to $\mathbf{A}_\epsilon$. Let X_ϵ be the Markov process

with the perturbed intensity matrix $\mathbf{A}_\epsilon$. The performance measure of X_ϵ can be decomposed as $\eta_\epsilon = \eta + \Delta\eta$. The derivative of η with respect to $\mathbf{Q}$ is then defined as

$$\frac{\partial \eta}{\partial \mathbf{Q}} = \lim_{\epsilon \to 0} \frac{\eta_\epsilon - \eta}{\epsilon}. \tag{3.2}$$

As a derivative, Equation (3.2) can be thought of as the sensitivity of η with respect to the changes in $\mathbf{A}$, i.e., in the direction of $\mathbf{Q}$.

Perturbation realization calculates what are called performance potentials for each state of the process, forming the potential vector $\mathbf{g} = (g_1, g_2, \ldots, g_n)$. 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 by capturing the expected performance gains/losses that are reachable from that state as well. Thus, some states might be assigned zero value from the performance function, but they still might have non-zero performance potential because the system has a probability of transitioning *to* a cost/reward state *from* that state. The potential vector gives a quantity by which we can calculate the derivative of the performance measure [5],

$$\frac{\partial \eta}{\partial \mathbf{Q}} = \boldsymbol{\pi} \mathbf{Q} \mathbf{g}. \tag{3.3}$$

The benefit of perturbation realization, as revealed by Equation (3.3), is that once $\boldsymbol{\pi}$ and $\mathbf{g}$ are computed from a single sample path, $\frac{\partial \eta}{\partial \mathbf{Q}}$ can be calculated for any number of user-defined Q matrices by simple matrix multiplications.

3.1.3. Algorithms for Single Sample Path. The perturbation matrix $\mathbf{Q}$ of Equation (3.3) is supplied by the user, but the other quantities must be calculated to determine $\frac{\partial \eta}{\partial \mathbf{Q}}$. Algorithms for computing $\boldsymbol{\pi}$ and $\mathbf{g}$ based on a single sample path, summarized below, are provided in [6].

Let T_k be the kth transition epoch of X_t (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 $I_i(X_k)$ is 1 if $X_k = i$ for state i and 0 otherwise. Then the steady-state probability π_i and potential g_i of each state i can be estimated from a single sample path of N transitions as:

$$\pi_i \approx \frac{1}{T_N} \left(\sum_{k=0}^{N-1} I_i(X_k) S_k \right) \text{ and} \tag{3.4}$$

$$g_i \approx \frac{\sum_{k=0}^{N} \left(I_i(X_k) \int_{T_k}^{T_k+T} f(X_t) dt \right)}{\sum_{k=0}^{N} I_i(X_k)}. \tag{3.5}$$

Equation (3.4) 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 (3.5) averages the performance gain from the visits to each state across the sample path, giving the potential for that state within the limit. The parameter T is a properly chosen positive value controlling the amount of time used to estimate the potentials. We found that our heuristic of setting T equal to the longest sojourn time in the sample path gave consistently accurate results across the different networks and sample paths.

Note that closed-form solutions exist for calculating $\boldsymbol{\pi}$ without having to estimate it from the sample path using Equation (3.4). We found that the accuracy of the $\frac{\partial \eta}{\partial \mathbf{Q}}$ estimates were improved by calculating $\boldsymbol{\pi}'\mathbf{Q}\mathbf{g}$ where $\boldsymbol{\pi}'$ is the steady-state distribution of the *perturbed* intensity matrix rather than the original intensity matrix. The closed-form solution for the perturbed steady-state distribution is found by calculating the normalized left eigenvector of $\mathbf{A}_\epsilon$ corresponding to the zero eigenvalue [33],

$$\boldsymbol{\pi}'\mathbf{A}_\epsilon = \mathbf{0}. \tag{3.6}$$

3.2. Other Methods for Sensitivity Analysis. In addition to perturbation realization, there are two other approaches to performing sensitivity analysis on Markov processes, namely, the likelihood ratio method and the reduced augmented chain method.

3.2.1. Likelihood Ratio Method. The likelihood ratio (LR) method [29, 23, 19] takes the ratio of the likelihood of a sample path from the original process with the likelihood of one that incorporates small changes into 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 sample path. 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 sample path 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.

3.2.2. Reduced Augmented Chain Method. The reduced augmented chain (RAC) method [8, 10] also uses a single sample path, 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 nominal system, and can be used for on-line estimation of the steady-state probability sensitivities.

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 sample path 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.

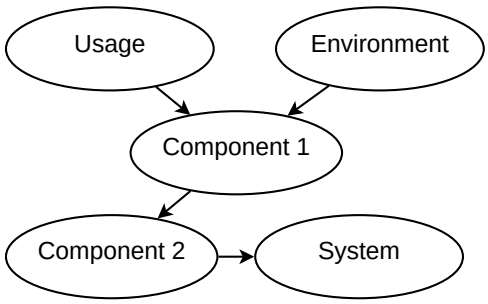

Figure 5. *Example CTBN representing reliability of a simple system.*

Note that LR and RAC could also be incorporated into the following approach. Even so, we chose perturbation realization for use in the experiments. Our goal is efficient sensitivity analysis of CTBNs, and perturbation realization's ability to calculate and retain potentials for different subnetworks is directly applicable to that end.

4. Perturbation Realization on CTBNs. Because a CTBN represents a Markov process (although in compact, factored form), the sample path method of perturbation realization for Markov processes is a natural candidate for CTBN sensitivity analysis. However, a straightforward application of perturbation realization to CTBNs would result in simply flattening the entire network into one large Markov process, with the states being the Cartesian product of all the nodes' states. 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 sample paths 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 Q matrices tested–all separately.

For a CTBN sensitivity analysis example, consider the example network shown in Figure 5. Suppose that a performance function is attached to the two states of the *System* 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 3.3 for calculating the change in performance for given perturbations. Our method for creating subnetworks, as with perturbation realization, follows a sample path approach.

4.1. Forward Sampling. Algorithm 1, adapted from [16], shows the pseudocode for creating a sample path 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 3-6 choose the starting states of the sample path and initialize the variables. Lines 7-23 are repeated until the sample path is of desired length, which could be the total time of the sample path or the total number of transitions. Lines 8-14 ensure that all variables have a proposed

Algorithm 1 Sample($\mathcal{C}$) // Create sample path of CTBN [16]

1: **Input:** $\mathcal{C}$ as CTBN
2: **Output:** sample path $\mathcal{S}$ as sequence of pairs of states and arrival times
3: **for each** $X \in \mathcal{C}$
4: choose $X(0)$ by sampling from $\mathcal{B}$
5: **end for**
6: $t \leftarrow 0$, $\mathcal{S} \leftarrow \emptyset$
7: **repeat** until termination
8: **for each** $X \in \mathcal{C}$
9: **if** $time(X) \neq null$ **then** continue **end if**
10: $\mathbf{A}_X \leftarrow \mathbf{A}_{X|\mathbf{Pa}(X)}$
11: $i \leftarrow X(t)$
12: $\Delta t \sim$ Exponential$(a_{i,i})$ // recall $a_{i,i}$ is entry i, i of $\mathbf{A}_X$
13: $time(X) \leftarrow t + \Delta t$
14: **end for**
15: $X' \leftarrow \text{argmin}_{X \in \mathcal{C}} (time(X))$
16: $t \leftarrow time(X')$
17: $X(t) \sim$ Multinomial$(\mathbf{A}_{X'}, X(t))$
18: append $\langle X(t), t \rangle$ to $\mathcal{S}$
19: $time(X') = null$
20: **for each** $Y \in \mathbf{Ch}(X')$
21: $time(Y) = null$
22: **end for**
23: **end repeat**
24: **return** $\mathcal{S}$

sojourn time, drawn from an exponential distribution whose parameter depends on the current states of the parents of each variable. Line 15 selects the variable with the soonest proposed transition time, and line 16 updates the current time of the sample path. Line 17 chooses the next state for the transitioning variable, according to a multinomial distribution whose parameters are derived from the current row of the variable's conditional intensity matrix. The state transition and time of the transition are recorded in the sample path. Finally, lines 19-22 reset the proposed sojourn times for that variable and all its children so that they will be re-sampled. The proposed sojourn times of the variable's children are reset because their conditional intensity matrices changed when their parent transitioned to a new state.

This sample path is used in Equation 3.5 to estimate the potential vector. We can also use a sample path to estimate an unconditional intensity matrix for any node in the network, as shown in the next section.

4.2. Subnetwork Isolation. For a node with parents in the network, the node will have conditional intensity matrices for every possible state instantiation of its parents. Isolating a subnetwork entails projecting a node's conditional intensity matrices onto a single unconditional intensity matrix, thereby removing the conditional dependence of a node on its parents.

This is analogous to marginalizing out variables in a Bayesian network.

We note that expectation propagation (EP) [26] or belief propagation (BP) [15] for CTBNs could also be used to calculate an unconditional intensity matrix for a CTBN node, provided the size of the window over which these methods integrate is sufficiently large to capture the long-term behavior of the node being isolated. Each of these methods is projecting onto a single unconditional Markov process but in different ways. Whereas EP and BP are given an interval of time over which to calculate the moments of an unconditional Markov process, node isolation is given a sample path. The length of the sample path is not interchangeable with the size of the interval. Using this sample-based method has an advantage in our application, because we are already computing sample paths from which to derive performance potentials. As we iterate over the sample path to estimate g, we can easily and simultaneously estimate unconditional intensity matrices, instead of also performing numerical integration for EP or BP.

Our novel method approximates the unconditional intensity matrix from a sample path, aggregating the average sojourn times and transition counts between subsystems. While combinations of nodes in a CTBN are able to model more complex behavior, this sampling approach approximates the behavior of a subsystem with a single node. While some accuracy is lost from this approximation, in many cases, the exponential distribution is sufficient to approximate the behavior of the subsystem while reducing the network complexity. Even in the simplest case, in which a two-state node can be removed from the network, this still reduces the state-space over the entire network in half.

Algorithm 2 shows the pseudocode for how node isolation can be accomplished for an arbitrary node in the network. The algorithm accepts a sample path, as generated from Algorithm 1, and the node to be isolated and returns the unconditional intensity matrix of the isolated node. First, lines 3-5 of the algorithm initialize the variables. Although the states of ancestor nodes may be changing throughout the sample path, lines 6-13 are concerned *only* with state changes of the node to isolate. Specifically, line 8 calculates the total amount of time spent in each state of the node, line 9 counts how many times the system has transitioned between each state of the node, and line 10 counts the total number of transitions that have occurred between states of the node. Next, lines 14-24 transform these statistics into an unconditional intensity matrix. Line 15 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 16 takes the negative reciprocal for use in the exponentially decreasing probability function. The relative number of transitions from a state to the remaining states represent the transition probabilities, which lines 17-23 normalize in accordance with the intensity matrix row constraints. In summary Algorithm 2 computes the maximum likelihood estimate of the parameters for X as an unconditional Markov process. Lines 6-13 gather the sufficient statistics for an unconditional intensity matrix, while lines 14-24 transform the sufficient statistics into the maximum likelihood estimates of $\mathbf{A}_X$.

4.3. Sufficient Conditions for CTBN Ergodicity. We need to ensure continued ergodicity when dividing the network into different subnetworks, as assumed 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,

Algorithm 2 IsolateNode($\mathcal{S}$, X) // Estimate unconditional intensity matrix of CTBN node

Input: $\mathcal{S}$ as sample path, X as node in $\mathcal{S}$ to isolate
Output: $\mathbf{A}_X$ as unconditional intensity matrix of X with entries $a_{i,j}$
$\mathbf{A}_X \leftarrow \mathbf{0}$, $\mathbf{v} \leftarrow \mathbf{0}$
$\langle X(t), t\rangle \leftarrow$ initial state of X in $\mathcal{S}$
$i \leftarrow X(t)$
for each transition $\langle X(t), t'\rangle$ of X in $\mathcal{S}$
    $j \leftarrow X(t)$
    $a_{i,i} \leftarrow a_{i,i} + (t' - t)$
    $a_{i,j} \leftarrow a_{i,j} + 1$
    $v_i \leftarrow v_i + 1$
    $i \leftarrow X(t)$
    $t \leftarrow t'$
end for
for each state i of X
    $a_{i,i} \leftarrow a_{i,i}$ / v_i
    $a_{i,i} \leftarrow -(1/a_{i,i})$
    $z \leftarrow 0$
    for each state $j \neq i$ of X
        $z \leftarrow z + a_{i,j}$
    end for
    for each state $j \neq i$ of X
        $a_{i,j} \leftarrow a_{i,j} \cdot (|a_{i,i}|/z)$
    end for
end for
return $\mathbf{A}_X$

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 4.1 (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*

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

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

Definition 4.2 (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:

$$(a_{ij}b_{kl}) = \begin{cases} a_{ij}|b_k & \text{if } i \neq j \text{ and } k = l \\ b_{kl}|a_i & \text{if } i = j \text{ and } k \neq l \\ a_{ij}|b_k + b_{kl}|a_i & \text{if } i = j \text{ and } k = l \\ 0 & \text{otherwise} \end{cases} \tag{4.1}$$

Lemma 4.3. *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_ib_{kl}) and $(a_{ij}b_l)$ are non-zero. In other words, there exists $t' > t$ such that

$$P(AB(t') = (a_jb_l)|AB(t) = (a_ib_k)) > 0 \text{ and } P(AB(t') = (a_ib_k)|AB(t) = (a_jb_l)) > 0.$$

■

Lemma 4.4. *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 4.1, 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_ib_k) to (a_jb_k) and from (a_ib_k) to (a_ib_l) for all states in $a_i, a_j \in A$ and $b_k, b_l \in B$. This implies that

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

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

Theorem 4.5. *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 4.3 and lemma 4.4, amalgamation of two nodes with ergodic conditional intensity matrices will produce another node with ergodic conditional intensity matrices. Therefore, by induction this process can be repeated to amalgamate any number of nodes in the CTBN and the resulting nodes will still have ergodic conditional intensity matrices. ■

4.4. Estimating CTBN Performance Derivative. Using perturbation realization and the method for isolating subnetworks (having ensured continued ergodicity), 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 sample paths may 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 (3.3) becomes inaccurate. Thus, being able to create and analyze sample paths from smaller subnetworks, and then being able to combine the results, becomes essential.

Algorithm 3 shows the pseudocode for performing sensitivity analysis. The algorithm accepts a CTBN and a set of Q 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 the intensity matrices. It calls four helper methods in addition to Algorithms 1 and 2. CollapseCycles detects cycles in $\mathcal{G}$, amalgamates each cycle's intensity matrices, and replaces each cycle with a single node. AmalgamateIntensityMatrices($\mathcal{C}$) expands the full joint intensity matrix of the CTBN $\mathcal{C}$ by repeatedly combining pairs of nodes as per Equation (4.1). CalculateSteadyStateProbabilities($\mathbf{A}_X$) calculates the steady-state probabilities $\boldsymbol{\pi}$ of the intensity matrix $\mathbf{A}_X$ by calculating the normalized eigenvector of corresponding with the zero eigenvalue as per Equation (3.6). CalculatePotentialVector($\mathcal{S}$) parses the sample path $\mathcal{S}$ and calculates the potential vector $\mathbf{g}$ as per Equation (3.5).

First, line 3 collapses the cycles of the CTBN. This is necessary before the top-down isolation begins, because node isolation requires a sample path 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 4 initializes the performance measure derivative estimate, which will be incrementally updated during the top-down isolation process. Lines 5-29 repeat until the condition of line 7 is satisfied, that is, when every node has been isolated. Line 6 finds the roots of the network, those without any parents. Lines 8-10 apply any perturbations to those root nodes. Line 11 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 12-28 iterate over the children. Line 13 creates a sample path for the child node, while line 14 uses that sample path to isolate it. Line 15 applies any perturbations to the child's conditional intensity matrices, and lines 16-17 isolate the perturbed child node. Line 18 adds the difference between the unconditional intensity matrices to the child perturbations. Lines 19-20 calculate the steady-state probabilities of the child node and its parents. Line 21 calculates the Q matrix for the child node and its parents. Line 22 calculates the potential vector from the perturbed sample

Algorithm 3 EstimateDerivative($\mathcal{C}$, $\mathbf{Q}$)

1: **Input:** $\mathcal{C}$ as CTBN, $\mathbf{Q}$ as set of Q matrices for the nodes of $\mathcal{C}$
2: **Output:** $\partial\eta/\partial\mathbf{Q}$ as performance measure derivative
3: $\mathcal{G}' \leftarrow$ CollapseCycles($\mathcal{G}$)
4: $\partial\eta/\partial\mathbf{Q} \leftarrow 0$
5: **repeat** until termination
6: $\quad\mathbf{X} \leftarrow \bigcup_{X\in\mathcal{G}'} X$ where $\mathbf{Pa}(X) = null$
7: $\quad$**if** $\mathbf{X} = null$ **then** terminate **end if**
8: $\quad$**for** $X \in \mathbf{X}$ **do**
9: $\quad\quad\mathbf{A}_X \leftarrow \mathbf{A}_X + \mathbf{Q}_X$
10: $\quad$**end for**
11: $\quad\mathbf{Y} \leftarrow \bigcup_{X\in\mathbf{X}} \mathbf{Ch}(X)$ where $\mathbf{Pa}(\mathbf{Ch}(X)) \subseteq \mathbf{X}$
12: $\quad$**for** $Y \in \mathbf{Y}$ **do**
13: $\quad\quad\mathcal{S} \leftarrow$ Sample($Y \cup \mathrm{Pa}(Y)$)
14: $\quad\quad Y' \leftarrow$ IsolateNode($\mathcal{S}, Y$)
15: $\quad\quad\mathbf{A}_{Y|\mathbf{Pa}(Y)} \leftarrow \mathbf{A}_{Y|\mathbf{Pa}(Y)} + \mathbf{Q}_{Y|\mathbf{Pa}(Y)}$
16: $\quad\quad\mathcal{S}' \leftarrow$ Sample($Y \cup \mathrm{Pa}(Y)$)
17: $\quad\quad Y'' \leftarrow$ IsolateNode($\mathcal{S}', Y$)
18: $\quad\quad\mathbf{Q}_Y \leftarrow \mathbf{Q}_Y + (\mathbf{A}_{Y''} - \mathbf{A}_{Y'})$
19: $\quad\quad\mathbf{A}_Y \leftarrow$ AmalgamateIntensityMatrices($Y \cup \mathbf{Pa}(Y)$)
20: $\quad\quad\boldsymbol{\pi} \leftarrow$ CalculateSteadyStateProbabilities($\mathbf{A}_Y$)
21: $\quad\quad\mathbf{Q} \leftarrow$ AmalgamateIntensityMatrices($\mathbf{Q}_Y \cup \mathbf{Q}_{\mathbf{Pa}(Y)}$)
22: $\quad\quad\mathbf{g} \leftarrow$ CalculatePotentialVector($\mathcal{S}'$)
23: $\quad\quad\partial\eta/\partial\mathbf{Q} \leftarrow \partial\eta/\partial\mathbf{Q} + \boldsymbol{\pi}\mathbf{Q}\mathbf{g}$
24: $\quad\quad$**for** $X \in \mathbf{Pa}(Y)$ **do**
25: $\quad\quad\quad$remove edge (X, Y) in $\mathcal{G}'$
26: $\quad\quad$**end for**
27: $\quad\quad$replace Y with Y' in $\mathcal{G}'$
28: $\quad$**end for**
29: **end repeat**
30: **return** $\partial\eta/\partial\mathbf{Q}$

path. Line 23 calculates the performance measure derivative for the subnetwork consisting of the child and its parents. Lines 24-27 finishes 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 14. After reaching the leaves of the CTBN, the aggregated performance measure derivative is returned in line 30.

Algorithm 3 avoids handling the whole CTBN at once. Sample paths 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. If dealing with an entire cycle is intractable, the node isolation can still be used following an iterative approach like the one described in [27], in which nodes are successively marginalized around the cycle until convergence.

4.5. CTBN Reliability Measures. While our approach for sensitivity analysis applies to general CTBNs and general performance functions [31], 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.

4.5.1. 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:

$$\frac{1}{|a_{0,0}|}. \tag{4.2}$$

In fact, our algorithm for sensitivity analysis provides the MTBF as a direct result of its top-down isolation of the nodes. 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.

4.5.2. Point Availability. The availability measure makes use of the performance function. 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:

$$f(i) = \begin{cases} 1 & i = n-1 \\ 0 & \text{otherwise} \end{cases}. \tag{4.3}$$

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

$$1 - \eta. \tag{4.4}$$

Sensitivity analysis allows us to measure how perturbations in the ancestors will be expected to impact availability.

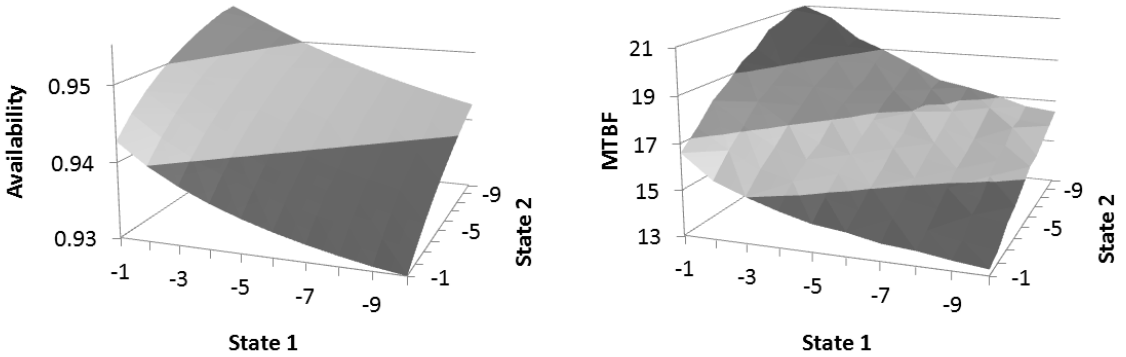

Figure 6. *Availability and MTBF estimates from pairs of perturbations on simple network.*

5. 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 a DFT encoded as a CTBN. The third is a model of a milling machine as learned from run-to-failure data.

For each network, 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 transitions generated in the samples.

5.1. Simple Network. The simple network for the first set of experiments is shown in Figure 5. Each node has 2 or 3 states, for a total state-space size of 72 over the entire network. This network is simple enough that the exact solutions can be computed easily using Equation 3.1.

The *System* node has two states, representing failed and non-failed states. Suppose that we want to observe how the parameters of *Component 2* affect system reliability. The naïve way would be to flatten 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. We tested all combinations of perturbations $\lambda = \{-1, -2, \ldots, -10\}$ to the two states of the *Component 2* node when *Component 1* is in state 1. We computed the exact availability and MTBF for each perturbation and compared to the estimates returned by Algorithm 3.

Before the perturbations are applied, the closed-form solution yields an availability of 94.30%, and an MTBF of 16.55 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 sample paths of 100,000 transitions in the node to isolate. The minimum and maximum availability resulting

from the perturbations was 93.0% and 95.5%, respectively, and the minimum and maximum MTBF was 13.3 and 21.5, respectively. The MTBF and point availability estimates for all perturbations are shown in Figure 6. For all perturbations, the sensitivity analysis method yielded less than 1% relative error.

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 the *System* node. By isolating each level separately, we reached 100,000 transitions in all of the isolated nodes with around 2.6 million total transitions generated across all levels. Our sensitivity analysis algorithm restricted the largest state-space size of any subnetwork to only 18 states (the *Usage-Environment-Component 1* subnetwork). The brute-force approach was 4 times as large, sampling from the whole network with 72 states. Furthermore, the brute-force method with 12.3 million transitions estimated the impact of a single perturbation. The brute-force method would need to have been run a total of 100 times to try all combinations of perturbations in λ. By reusing the potentials, the sensitivity analysis method could test any number of perturbations with few extra calculations.

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.

5.2. Cardiac Assist System. Specially constructed CTBNs have been used to represent dynamic fault trees specifically for reliability modeling, in which each of the gates of a DFT can be mapped onto CTBN nodes [3]. In this experiment, we use the CTBN representation of a DFT for a cardiac assist system (CAS) as given in [3]. This DFT is broadly used in the literature and based on a real-world system [2, 28]. The network is shown in Figure 7. Of the various repair policies evaluated in [3], 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. We can test how different combinations of perturbations to the different subsystems affect the reliability of 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 2688 hours and the point availability to be 99.43%. For our sensitivity analysis, we test all combinations of the following perturbations: $\lambda_{CS} = \{0, -0.0001, -0.0002\}$, $\lambda_{MS} = \{0, -0.001, -0.002\}$, and $\lambda_{PA} = \{0, -0.01, -0.02\}$.

Using our sensitivity analysis technique, we can isolate the three nodes for the different subsystems: *CPU*, *Pump*, and *Motor*. The brute-force approach is sampling from a state-space size of over 6.6 million over the whole network. When isolating individual subsystems, the

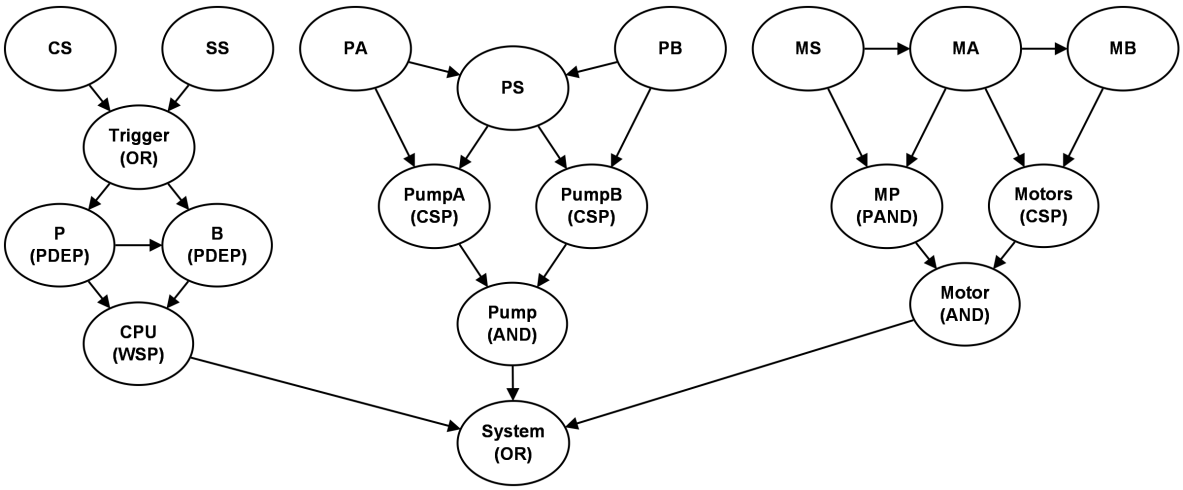

Figure 7. *Cardiac assist system model.*

state-space sizes are 144, 144, and 160 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. Furthermore, by saving the potentials and the unconditional intensity matrices of the three subsystem nodes, we can apply the different perturbations to different subnetworks, 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 every time. All of the perturbations tested decreases in system reliability. The minimum availability resulting from the perturbations was 99.41%, and the minimum MTBF was 1799 hours. For all perturbations, the sensitivity analysis method yielded less than 1% relative error with respect to the brute-force approach estimates.

The brute-force approach had to generate more than 9 billion transitions in order to generate 27 million transitions in the *System* node (1 million transitions for each of the 27 different combinations of the perturbations). The sensitivity analysis method can take advantage of the fact that perturbations in subnetworks are reused multiple times between these combinations. Around 10 million transitions were generated for each perturbation of a subnetwork, for a total of around 90 million transitions (3 perturbations for the 3 subnetworks with 10 million transitions each). Another 10 million transitions were generated to estimate the potentials for the lower network. In other words, the brute-force approach scaled exponentially with respect to the number of perturbation combinations, while the sensitivity analysis method scaled linearly because it could work with each subnetwork separately as enabled by the CTBN representation of the system.

5.3. 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 measuring the flank wear of the cutting tool [1]. 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

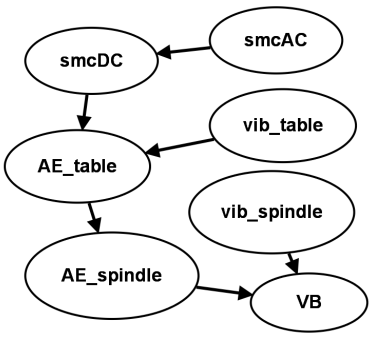

Figure 8. *Milling machine model.*

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) [30]. 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 8. 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. Therefore, a repair rate of 1 hour 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 have an estimated MTBF of 55.5 minutes and an estimated point availability of 47.9%. Now suppose we apply perturbations to the intensity matrix of $smcAC$ as shown in Table 1. Using the brute-force approach, the new estimated MTBF changes at $i = 1.0$ to 55.2 minutes while the estimated point availability changes to 47.7%.

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 can estimate the MTBF and the point availability. The relative error of the point availability estimates were less than 1% for all cases. The relative error of the MTBF estimates were roughly between 1% and 2%. The MTBF estimates and relative error of our sensitivity analysis approach are shown in Figure 9.

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 from the lower subnetwork. By dividing the network into upper and lower subnetworks, the size of the state-space for sampling is 81 and 54 states, respectively, instead of 1458 states when sampling over the entire network. The sizes of the Q 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 Q 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 in order to manage complexity.

The brute-force approach had to generate 4.8 billion transitions on average in order to get

Table 1

Intensity Matrix (left) for smcAC and Q Matrices (right) for $i = \{0.0, 0.1, 0.2, \ldots, 1.0\}$

-0.134	0.132	0.002
0.134	-0.267	0.133
0.002	0.131	-0.133

-i	i/2	i/2
0.0	0.0	0.0
0.0	0.0	0.0

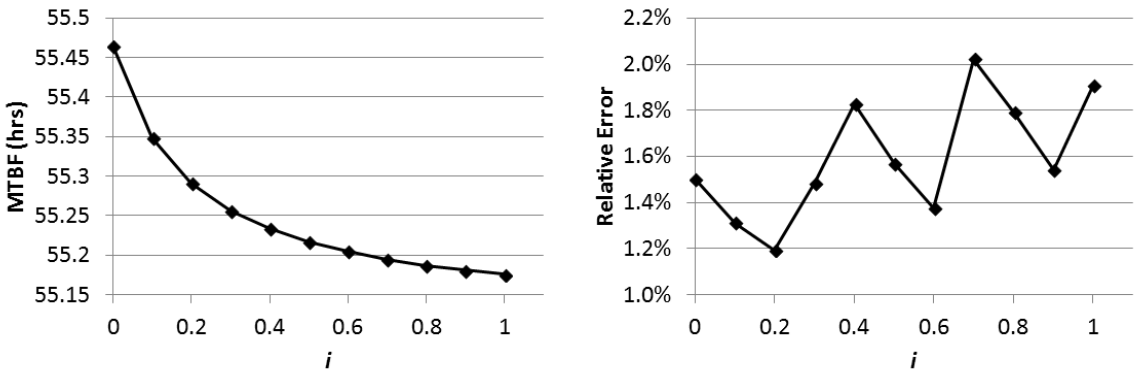

Figure 9. *MTBF estimates (left) and relative error (right) for perturbations on milling machine network.*

100,000 transitions in *VB* for each perturbation. The sensitivity analysis approach generated less than 2.8 million transitions on average to isolate *AE_table* and then around 3.6 billion transitions to get 100,000 transitions in *VB*. Each perturbation of *smcAC* entails recalculating the unconditional intensity matrix for *AE_table* and then reusing the potentials of the lower network. Therefore, to test the 10 perturbations, the sensitivity analysis needed to generate less than 4 billion transitions, while the brute-force approach needed to generate around 48 billions transitions.

6. Discussion. For each of these three experiments, we have shown an advantage of our approach over the brute-force approach. The advantage becomes especially apparent when the user wants to evaluate multiple perturbations, as is common with sensitivity analysis. Even with a single Q matrix, our approach allows for more control over the sampling of different subnetworks. Our approach allows sensitivity analysis to be performed on larger networks instead of single Markov processes, in which exact solutions cannot be computed due to the exponential state-space size of CTBNs.

For the simple network, we were able to use top-down node isolation to more efficiently estimate the change in reliability resulting from multiple perturbations. Even for a single perturbation, the total number of transitions generated by the sensitivity analysis method was fewer than the brute-force approach by a factor of 4 with the same level of accuracy as compared to the exact solution.

For the CAS network, we were able to reuse potentials between the three different subsystems to test out combinations of perturbations much more efficiently than the brute-force approach. Furthermore, the brute-force approach had to sample from the entire network all at once, even though the three subsystems were evolving at different rates. This meant the brute-

force approach was disproportionately sampling from different subsystems. The brute-force approach scaled exponentially with respect to the number of combinations of perturbations between the three subsystems. By isolating each subnetwork separately, our sensitivity analysis method scaled linearly.

For the milling machine network, we were able to focus on perturbations of the upper network and cascade the effect of these changes down to the lower subnetwork. Sampling from the upper subnetwork alone and reusing the potentials from the lower subnetwork was an order of magnitude more efficient than sampling from the entire network. In this case, the relative error increased as the magnitude of the perturbations increased, even with longer sample paths. Also, even without perturbations, the node isolation introduced a bias into the estimate. This illustrates the fact that a set of conditional intensity matrix can only be approximated by a single unconditional intensity matrix.

While our method has relatively high accuracy for these choices of networks, there is no guarantee for all cases. Inference in arbitrary CTBNs has been shown to be NP-hard [32], and this result also applies to the sensitivity analysis presented here. Thus, we cannot guarantee the performance of the sensitivity method on arbitrary networks. The node isolation method approximates a multiple-node subnetwork with a single node, but for some subnetworks this single-node approximation may not be accurate enough (EP and BP face the same problem). However, we have shown that the sensitivity analysis method can be applied successfully to real-world networks for which the exact solution is intractable and the brute-force approach is inefficient.

7. 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 subnetwork isolation method can be used to counteract the exponential blow-up in the size of the state-space while simultaneously yielding sample paths for re-usable performance potential estimates. 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 specifically for CTBNs.

One direction for future work is to find constrained networks which admit guaranteed tractable inference. For perturbation realization, the relationship between the length of the sample paths (computational complexity) and the accuracy of the estimates, both of the performance measure derivatives and the unconditional intensity matrices, is also an area future work.

REFERENCES

[1] A. Agogino and 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).

[2] Hichem Boudali, Pepijn Crouzen, and Marielle Stoelinga, *Dynamic fault tree analysis using input/output interactive Markov chains*, in 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks, IEEE, 2007, pp. 708–717.

[3] D. Cao, *Novel models and algorithms for systems reliability modeling and optimization*, PhD thesis, Wayne State University, 2011.
[4] Xi-Ren Cao, *Potential based sensitivity analysis of Markov chains*, in Proceedings of the 35th IEEE Decision and Control, vol. 1, Dec. 1996, pp. 568–569.
[5] Xi-Ren Cao and Han-Fu Chen, *Perturbation realization, potentials, and sensitivity analysis of Markov processes*, IEEE Transactions on Automatic Control, 42 (1997), pp. 1382–1393.
[6] Xi-Ren Cao and Yat-Wah Wan, *Algorithms for sensitivity analysis of Markov systems through potentials and perturbation realization*, IEEE Transactions on Control Systems Technology, 6 (1998), pp. 482–494.
[7] Xi-Ren Cao, Xue-Ming Yuan, and Li Qiu, *A single sample path-based performance sensitivity formula for Markov chains*, IEEE Transactions on Automatic Control, 41 (1996), pp. 1814–1817.
[8] Christos Cassandras and Stephen Strickland, *An "augmented chain" approach for on-line sensitivity analysis of Markov process*, in 26th IEEE Conference on Decision and Control, vol. 26, Dec. 1987, pp. 1873–1878.
[9] C. Cassandras and S. Strickland, *Observable augmented systems for sensitivity analysis of Markov and semi-Markov processes*, IEEE Transactions on Automatic Control, 34 (1989), pp. 1026–1037.
[10] C.G. Cassandras and S.G. Strickland, *On-line sensitivity analysis of Markov chains*, IEEE Transactions on Automatic Control, 34 (1989), pp. 76–86.
[11] E. Castillo, J. Gutierrez, and A. Hadi, *Sensitivity analysis in discrete Bayesian networks*, IEEE Transactions on Systems, Man and Cybernetics, Part A: Systems and Humans, 27 (1997), pp. 412–423.
[12] Hei Chan and Adnan Darwiche, *Sensitivity analysis in Bayesian networks: From single to multiple parameters*, in Proceedings of the Twentieth Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-04), Arlington, Virginia, 2004, AUAI Press, pp. 67–75.
[13] I. Cohn, T. El-Hay, N. Friedman, and R. Kupferman, *Mean field variational approximation for continuous-time Bayesian networks*, in Proceedings of the 25th Annual Conference on Uncertainty in Artificial Intelligence (UAI), AUAI Press, 2009, pp. 91–100.
[14] Liyi Dai and Yu-Chi Ho, *Structural infinitesimal perturbation analysis (SIPA) for derivative estimation of discrete-event dynamic systems*, IEEE Transactions on Automatic Control, 40 (1995), pp. 1154–1166.
[15] T. El-Hay, I. Cohn, N. Friedman, and R. Kupferman, *Continuous-time belief propagation.*, in International Conference on Machine Learning (ICML), J. Frnkranz and T. Joachims, eds., 2010, pp. 343–350.
[16] Y. Fan and C.R. Shelton, *Sampling for approximate inference in continuous time Bayesian networks*, in Tenth International Symposium on Artificial Intelligence and Mathematics, 2008.
[17] Yu Fan and Christian Shelton, *Learning continuous-time social network dynamics*, in Proceedings of the Twenty-Fifth Conference on Uncertainty in Artificial Intelligence, UAI '09, Arlington, Virginia, United States, 2009, AUAI Press, pp. 161–168.
[18] E. Gatti, D. Luciani, and F. Stella, *A continuous time Bayesian network model for cardiogenic heart failure*, Flexible Services and Manufacturing Journal, (2011), pp. 1–20.
[19] Paul Glasserman, *Derivative estimates from simulation of continuous-time markov chains*, Oper. Res., 40 (1992), pp. 292–308.
[20] Yu-Chi Ho and Jian-Qiang Hu, *An infinitesimal perturbation analysis algorithm for a multiclass G/G/1 queue*, Operations Research Letters, 9 (1990), pp. 35–44.
[21] D. Koller and N. Friedman, *Probabilistic Graphical Models: Principles and Techniques*, MIT Press, 2009.
[22] Zikuan Liu and Fengsheng Tu, *Single sample path-based sensitivity analysis of Markov processes using uniformization*, IEEE Transactions on Automatic Control, 44 (1999), pp. 872–875.
[23] Marvin K. Nakayama, Ambuj Goyal, and Peter W. Glynn, *Likelihood ratio sensitivity analysis for Markovian models of highly dependable systems*, Operations Research, 42 (1994), pp. pp. 137–157.
[24] Uri Nodelman, *Continuous Time Bayesian Networks*, PhD thesis, Stanford University, Stanford, California, 2007.
[25] Uri Nodelman and Eric Horvitz, *Continuous time Bayesian networks for inferring users' presence and activities with extensions for modeling and evaluation*, tech. report, a, 2003.

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

[27] U. Nodelman, C. Shelton, and D. Koller, *Continuous time Bayesian networks*, in Proceedings of the Eighteenth Conference on Uncertainty in Artificial Intelligence (UAI), 2002, pp. 378–387.

[28] Luigi Portinale, Daniele Codetta Raiteri, and Stefania Montani, *Supporting reliability engineers in exploiting the power of dynamic Bayesian networks*, International journal of approximate reasoning, 51 (2010), pp. 179–195.

[29] Martin Reiman and Alan Weiss, *Sensitivity analysis via likelihood ratios*, in Proceedings of the 18th conference on Winter simulation, WSC '86, New York, NY, USA, 1986, ACM, pp. 285–289.

[30] Christian R. Shelton, Yu Fan, William Lam, Joon Lee, and Jing Xu, *Continuous time Bayesian network reasoning and learning engine*, Journal of Machine Learning Research, 11 (2010), pp. 1137–1140.

[31] Liessman Sturlaugson and John W. Sheppard, *Factored performance functions with structural representation in continuous time Bayesian networks*, in The Twenty-Seventh International FLAIRS Conference, 2014.

[32] L. Sturlaugson and J. W. Sheppard, *Inference complexity in continuous time Bayesian networks*, in Proceedings of the 30th Annual Conference on Uncertainty in Artificial Intelligence (UAI), AUAI Press, 2014, pp. 772–779.

[33] Howard M. Taylor and Samuel Karlin, *An introduction to stochastic modeling*, Academic press, 1998.

[34] Li Xia and Xi-Ren Cao, *Relationship between perturbation realization factors with queueing models and Markov models*, IEEE Transactions on Automatic Control, 51 (2006), pp. 1699–1704.

[35] Jing Xu and Christian Shelton, *Intrusion detection using continuous time Bayesian networks*, J. Artif. Int. Res., 39 (2010), pp. 745–774.