Deriving Prognostic Continuous Time Bayesian Networks from Fault Trees

Logan Perreault, Monica Thornton, John W. Sheppard

Montana State University, Bozeman, MT, 59717, United States
logan.perreault@montana.edu
monica.thornton@montana.edu
john.sheppard@montana.edu

Abstract

Probabilistic graphical models have been applied successfully to a number of Prognostic and Health Management (PHM) applications. Continuous time Bayesian networks (CTBNs) are one such model, and they are capable of representing discrete systems that evolve in continuous time. In this work, we propose a method for constructing a CTBN from a fault tree, a model often used for evaluating system reliability. Additionally, we provide a method for reducing the number of required CTBN parameters by pruning unnecessary portions of the fault tree. Furthermore, we take advantage of the information encoded in the remaining gates of the tree and make use of the Noisy-OR model, offering additional reductions in the number of parameters needed to specify the CTBN model. We show how a CTBN derived from a fault tree can be combined with a CTBN derived from a D-matrix to form a unified model. This allows for a description of faults and effects that evolve in continuous time based on test outcomes. We demonstrate the derivation and parameterization processes using a running example, and show how the resulting model can be queried to obtain information about the state of the system over time.

1. Introduction

In a variety of applications, complex systems have become increasingly important, and their significance in fields as diverse as aerospace and medicine cannot be overstated. The complexity of these systems, combined with the severity of consequences in the event of system failure, underscores the importance of diagnostics and prognostics in these critical domains. Probabilistic graphical models (PGMs) provide a convenient method for performing diagnostics and prognostics in complex systems, as reasoning about the model reflects reasoning about uncertainties in the actual system. There are a number of viable models that can be selected for these tasks, but in selecting a model, a balance needs to be struck between finding one that can capture the most critical details of the domain while still retaining its transparency and tractability.

As our work focuses primarily on prognostics, we utilize a framework that is equipped for modeling system change over time, the continuous time Bayesian network (CTBN). A CTBN is a PGM with demonstrated utility as a prognostic model that can compactly describe complicated systems. As a result, CTBNs have been applied successfully in a number of diverse domains. Despite their numerous advantages, however, the PHM community has been slow to adopt these models. We believe that this reticence is due, in part, to the difficulty associated with constructing and parameterizing the networks manually, or obtaining the necessary data required to use a machine learning algorithm. To remedy this, we propose a method for simplifying CTBN model construction through an automatic derivation process where the CTBN is derived from fault trees. This has the advantage of making CTBNs easier to use while also reducing the potential for human error in building the network.

In this work, we describe the CTBN network structure that corresponds to the structure of the fault tree and explain how to parameterize the model using the logic gates found in the fault tree, as well as other commonly available diagnostic information. The resulting CTBN contains a node for every fault and effect. The CTBN's structure describes the interaction between faults and effects through time and works as a prognostic model capable of predicting the likelihood of effects based on the occurrence of a set of faults. To aid in prognostic tasks, we formally define how the derivation process is performed and support this description by working through an example derivation for a simple fault tree.

In addition, we also address some practical concerns related to the use of CTBNs, and propose solutions intended to prevent models from becoming intractable. These solutions involve introducing a pruning process that eliminates redundant

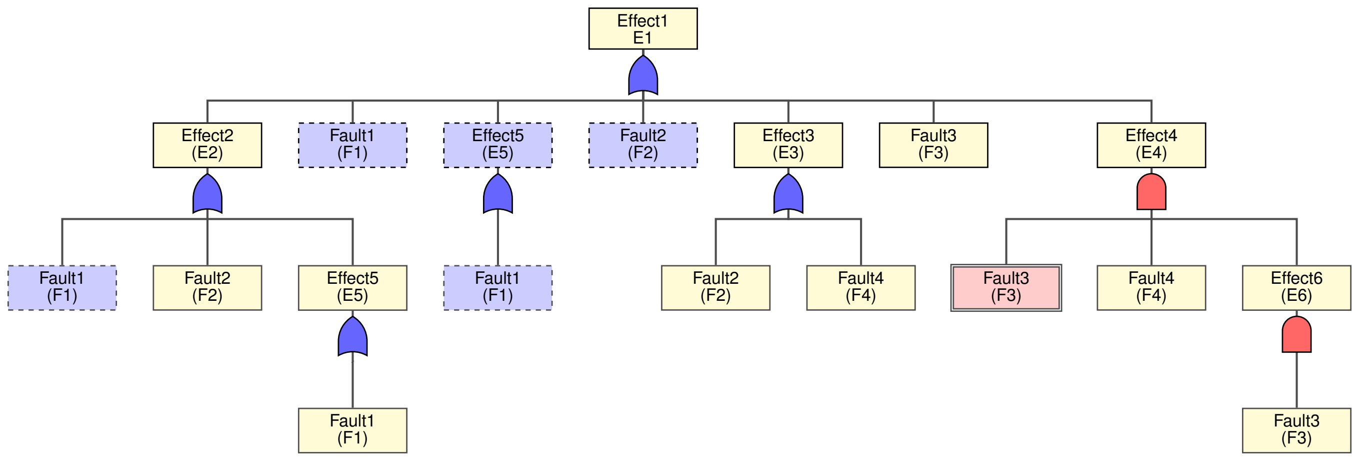

Figure 1. Example fault tree with 6 distinct effects and 4 distinct faults.

information from the fault tree, and use a method that reduces the space complexity of a CTBN node from exponential to linear. Finally, we discuss how this fault tree derivation process relates to previous work in constructing CTBNs from D-Matrices (Perreault, Thornton, Strasser, & Sheppard, 2015). The fault tree derivation process detailed in this work, along with the D-matrix derivation process introduced in our previous work, will assist in the construction of prognostic CTBN models. In simplifying this process, our goal is to eliminate the barriers to entry associated with incorporating CTBNs into the PHM community.

2. BACKGROUND

Before describing our proposed approach for deriving CTBNs from fault trees, we first provide a brief overview of the concepts on which this work was founded.

2.1. Fault Trees

Fault tree analysis (FTA) is a powerful and well-established technique for evaluating system design in a reliability context. In a number of critical domains, fault trees encode knowledge about the system in a manner that is intuitive and easy to interpret. A fault tree provides a graphical representation that depicts the ways in which the failure of one or more system components can lead to system failure. As shown in Figure 1, a fault tree is a directed acyclic graph (DAG) consisting of a set of events and a set of logic gates. The events depicted in Figure 1 consist of faults and their associated effects, and the logic gates are either AND or OR gates. These logic gates determine how faults at lower levels of the tree can contribute to system failure.

Some fault tree representations may allow for redundancies in the system that can cause the same component to appear in multiple parts of the tree. These redundancies can also be introduced when the system of interest is not represented easily as a polytree. The process of converting the system to a corresponding polytree representation often involves duplicating portions of the tree and placing them appropriately to avoid cycles. The fault F_1 in Figure 1 is an example of this kind of duplication. While these redundant components do provide valid information about the system, in many cases the information they provide is already encoded at another location in the tree. Furthermore, when these redundant components are removed from the fault tree with a valid pruning process, the resulting fault tree is easier to store, update and analyze. Fortunately, the pruned tree remains functionally equivalent to the original fault tree.

A related diagnostic model is the Fault Isolation Manual (FIM), which often plays a significant role in the maintenance of large systems (Simpson & Sheppard, 1994). FIMs are derived from decision trees and are typically referred to as a fault tree[1]. The internal nodes of the FIM correspond to a series of tests, and the leaves correspond to a fault. To use the FIM, the user first performs the test specified by the root of the tree and follows the branch specified by the test outcome until reaching the leaf node corresponding to the diagnosed fault. In this way, a FIM enables the isolation of individual faults or ambiguity groups through the application of a structured sequence of tests (Strasser & Sheppard, 2013).

2.2. Continuous Time Bayesian Networks

While fault trees provide an overall snapshot of the diagnostic dependencies of a system, they are unable to capture how the system may change over time. A continuous time Markov process (CTMP) is a model that describes a set of discrete state variables X that evolve in continuous time, providing a method for modeling domains that do not conform to a predefined time granularity (Nodelman, Shelton, & Koller, 2002). There are two major components that define the CTMP, an

[1]Note that FIM-based fault trees are different from fault trees developed through FTA.

initial distribution over the state space $P(X(0))$ and an associated intensity matrix Q describing the state transition behavior. An entry $q_{i,j}$ in Q provides the intensity with which X is expected to transition from state x^i to state x^j, and is drawn from an exponential distribution with a rate corresponding to $q_{i,j}$. Each diagonal entry $q_{i,i}$ is defined as the negative sum of the entries in row i, thus ensuring that each row in the matrix sums to zero.

A row containing a nonzero entry represents a transient state, a state that will transition to another in time. Conversely, a row consisting of all zeroes is an absorbing state, one in which there is no way to transition to another state. In the event an absorbing state exists, a Markov process will transition to it, and will remain in that state permanently.

While CTMPs do provide a framework for modeling time directly, there are inherent limitations that prevent this model from being applied to complex systems. In a CTMP, the sizes of the initial distribution $P(X(0))$ and the intensity matrix Q are exponential in the number of variables in the system, and this exponential increase in the size of the state space makes reasoning tasks difficult, if not intractable.

The continuous time Bayesian network (CTBN) mitigates this problem by factoring the CTMP, taking advantage of the conditional independencies among the variables in the system. These conditional independencies are encoded in a directed network structure G, where the nodes of G correspond to the variables of the system and the edges connecting nodes represent a relationship between the variables. Specifically, an edge from node A to node B indicates that the behavior of variable B depends on the state of A. There are significant modeling advantages associated with adopting this factored representation, namely that rather than needing to parameterize the model with a single exponentially sized probability distribution and intensity matrix, local probability distributions and intensity matrices are defined over each variable individually. These matrices are conditioned on the states of the parents of the node and are referred to as conditional intensity matrices (CIMs).

To use the CTBNs within a diagnostic or prognostic context, the state of the faults and effects are inferred via queries that are applied over the course of time. In the event that there is existing knowledge about the system, evidence can be set to indicate the known states. Diagnostic and prognostic models typically consist of a network with fault and test nodes, meaning that the evidence set on the test nodes describes the test outcomes, and the state of the faults is inferred based on this information. Queries about the faults, tests, and effects can be used to request the associated probabilities over time periods of interest.

3. RELATED WORK

Given the advantages afforded by PGMs, it is not surprising that a fair amount of work has been done to extend fault trees through the use of PGMs. Bobbio *et al.* provide a method to map any static fault tree to a Bayesian network (BN), and they demonstrate that this framework results in increased representational power, the relaxation of some of the typical fault tree constraints, and performance gains at the analysis level (Bobbio, Portinale, Minichino, & Ciancamerla, 2001).

BNs are commonly used in the dependability analysis of safety critical systems, because they provide a robust probabilistic method of reasoning in the presence of uncertainty. By design, however, BNs reason about static processes, which makes them ineffective on the kind of structured stochastic processes that evolve over continuous time, and these problems frequently occur in the real world. To address this concern, work has been done to map dynamic fault trees to dynamic Bayesian networks (DBNs) (Montani, Portinale, & Bobbio, 2006). A DBN can model time-sliced stochastic functions by discretizing time at some predetermined granularity, effectively extending the Bayesian network to dynamic processes by representing the state of the system at different points in time. The method used by Montani *et al.* translates the basic dynamic gates into a corresponding DBN model, and the DBN is evaluated at different points in time.

In a DBN, the variables in the different time steps are related to each other, and allowing these nodes to have probabilities conditional on their state in a previous timestep allows the model to account for dynamic behavior (Ruijters & Stoelinga, 2015). This ability to model dynamic behavior, the authors contend, is something that would prove useful in modeling dependability applications. DBNs, however, are not appropriate for use in all domains. If the application has a strong temporal component but lacks natural time slices, either relevant data will be excluded from the model, or all variables in the system will need to modeled at the finest possible granularity. The shortcomings of the BN and DBN framework in time-sensitive complex domains motivates the idea that mapping fault trees to CTBN models could be an attractive alternative.

Previous work has been done to develop CTBNs that correspond to fault trees. Specifically, a method was formulated to translate the basic logical gates in a dynamic fault tree to the corresponding CTBN model (Cao, 2011). Cao constructed CTBN nodes that were deterministic by forcing instantaneous transitions using infinite rate values. We relax this assumption to allow for effects that occur at some point after the preceding faults. The resulting structure describes the interaction between faults and effects through time, and works as a prognostic model capable of predicting the likelihood of effects based on the occurrence of a set of faults. While Cao's method does provide an effective method for relating fault trees and CTBNs, it fails to capture the temporal dynamics necessary to model complex systems that change through time.

The fault tree derivation process described in this work is related to our previous work in constructing CTBNs from D-Matrices (Perreault et al., 2015). Like fault trees, D-matrices are commonly used in diagnostic contexts and are employed in a number of modeling tools. A D-matrix is an adjacency matrix that represents explicit relationships between tests and faults, and work has been done to use them in diagnostic contexts (Sheppard & Butcher, 2007). Within a prognostic context, the CTBNs obtained from D-Matrices contain a set of fault and test nodes. This set of faults is the same as the set obtained during the fault tree derivation process, and in Section 7 we show how a single CTBN can be formed by combining the two derived networks into a single network capable of predicting fault and risk events based on the application of test evidence through time.

4. PRUNING PROCESS

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

$$(\mathbf{E2.1}) \qquad F_1 = 1 \rightarrow E_2 = 1$$
$$(\mathbf{E2.2}) \qquad F_2 = 1 \rightarrow E_2 = 1$$
$$(\mathbf{E2.3}) \qquad E_5 = 1 \rightarrow E_2 = 1$$

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

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

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

Algorithm 1 Fault Tree Pruning Algorithm

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

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

A similar idea can also be applied to nested AND gates in a fault tree. Consider effect E_4 in Figure 1. The output of E_4 is determined by the rule $F_3 \wedge F_4 \wedge E_6 \rightarrow E_4$. Effect E_6 is in turn determined by the rule $F_3 \rightarrow E_6$. To obtain the implied rules for E_4, we can simply replace node E_6 with the logical expression that determines it. This results in the rule $F_3 \wedge F_4 \wedge (F_3) \rightarrow E_4$. Here we can see that F_3 is redundant in the logical expression. We say that effect E_6 accounts for F_3, and therefore E_4 does not require F_3 as a direct descendant. The pruned F_3 is highlighted in Figure 1 with a double outline. Note that in more complex cases where subeffects have more than one child, implied rules may involve lengthy chains of ANDed variables.

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

5. DERIVING CTBN STRUCTURE

The structure of a CTBN takes the form of a directed graph $G = (V, E)$, where V is a set of nodes v_i, and E is a set of edges e_{ij} connecting node v_i to v_j. We can derive this structure directly using a fault tree as follows. First we obtain the set of nodes V by extracting them directly from the faults and effects in the fault tree. Note that a fault or effect may occur in multiple locations of the fault tree, but the corresponding node in the CTBN will occur only once. We must therefore insure that no duplicates are added to the set V when iterating through the fault tree components.

Next we introduce edges between the nodes to form the set E. These can be obtained directly from the structure of the fault tree. First note that faults only occur as leaves in the fault tree. This means that faults occur on their own accord, and do not depend on any other modeled variables. The corresponding nodes in the CTBN therefore have no parents. Next we consider the effects in the fault tree, whose states are determined by its inputs. We represent this dependence on the inputs by adding an edge from the nodes corresponding to each input u_i to the node corresponding to the effect. The resulting CTBN has the same structure as the fault-tree after it has been pruned, but is generally shown reversed with the faults on top.

Network Construction Example

Recall the fault tree shown in Figure 1. We can derive the network structure for a CTBN corresponding to this fault tree by using the process described in the previous section. Figure 2 shows the constructed CTBN, consisting of the four faults and six effects contained in the original fault tree. The dashed lines indicate cases where edges can be removed using the pruning process for OR gates, while the dotted line shows the edge that is removed when pruning the nested AND gates. Note that aside from the alterations due to the pruning process, the structure of the CTBN is not determined by the type of gates. In other words, parents are added to an effect regardless of whether it is an AND or an OR gate. In the next section, we show how these gate types are used to determine the parameterization for a CTBN.

6. PARAMETERIZING THE CTBN

The task of parameterizing a CTBN involves populating the initial distribution and the rates for the CIMs associated with each node in the network. In general, this set of parameters can be quite large. Let n_X be the number of states in the domain of variable X. Then for some node X, the number of parameters required to populate all the associated CIMs is $(n_X(n_X - 1)) \cdot \prod_{A \in Pa(X)} n_A$. In system reliability, the prior probability that each components starts in a failing state is often known, and in this work we assume that all variables start in a working state with a probability of 1.0. For our purposes, we can assume the variables are binary, consisting of a failing state and a non-failing state. In this case, each node X requires $2^{(|Pa(X)|+1)}$ rates, which can make identifying parameters for even relatively small models a difficult task. We aim to reduce the number of rates required to parameterize a node by taking advantage of common reliability information, and by exploiting behavioral information obtained from the underlying fault tree.

6.1. Parameterizing Fault Nodes

To begin, consider a node F_i representing a fault variable in the CTBN. By construction, F_i contains no parents, and therefore has only a single unconditional intensity matrix. This single CIM describes how quickly we expect a fault to transition to a failing state, and back to a non-failing state. This equates to failure and repair rates, which we denote using λ and μ respectively. As a CTBN is parameterized using transition rates, we can insert these values directly into the cells of a CIM.

Note that failure rates are often readily available for a system for which a fault tree has been constructed. Alternatively, the information may be represented in terms of the mean time between failure (MTBF), which is simply $MTBF = 1/\lambda$. The mean time to repair (MTTR) is also commonly available for many diagnostic models, or may simply be set to 0 if no repair policy exists. In this case the failing state is absorbing, and the MTBF is instead referred to as the mean time to failure (MTTF).

The intensity matrix for a fault F is shown below, where λ_i and μ_i are the failure and repair rates respectively for fault F.

$$Q_F = \begin{array}{c} \\ f^0 \\ f^1 \end{array} \begin{array}{c} \begin{array}{cc} f^0 & f^1 \end{array} \\ \begin{pmatrix} -\lambda_f & \lambda_f \\ \mu_f & -\mu_f \end{pmatrix} \end{array}$$

6.2. Parameterizing AND Effects

Although we wish to model state transitions that occur over time, eventually we expect the behavior of an effect node in a CTBN to match the behavior of the corresponding static logic gate. Let F_X be the discrete function corresponding to the gate from which variable X was derived. Then we expect that $\lim_{t\to\infty} P(X(t) = F_X(Pa(X))) = 1.0$. We ensure this behavior for each CIM $Q_{X|Pa(X)}$ by guaranteeing a non-zero transition rate *to* the state produced by $F_X(Pa(X))$, and a zero rate *from* the state produced by $F_X(Pa(X))$. This results in an absorbing state that will eventually be reached,

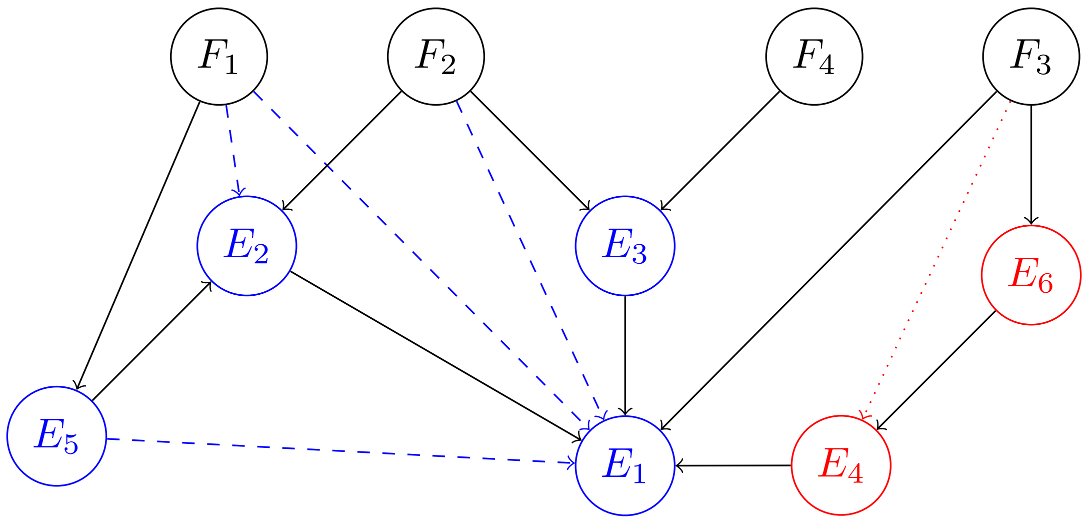

Figure 2. Example CTBN constructed from fault tree.

at which point the variable will remain in this state until a change in at least one of the parents occurs.

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

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

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

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

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

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

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

6.3. Parameterizing OR Effects

When a variable X is created from an OR gate, F_X is defined as the logical OR function. In this case, we expect F_X to produce a value of 0 if and only if all the parents of node X are 0. Again, we ensure that this CTBN node eventually reaches state 0, we parameterize the CIM such that state 0 is an absorbing state:

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

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

Next we look at the cases where F_X evaluates to 1. Since we are dealing with a logical OR gate, this occurs whenever at least a single parent is in state 1. In this case, we desire the opposite behavior for node X, and need to parameterize the CIMs such that the node eventually transitions to state 1:

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

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

Here again we are required to specify a rate parameter for every possible state instantiation of the parents where at least one parent is in state 1. This means that we require $2^{|Pa(X)|}$ parameters for node X. As in the case of the AND node in the previous section, we can reduce this by making a simplifying assumption. Specifically, we are able to make use of the Noisy-OR model, since we interpreting the model as causal (Perreault, Strasser, Thornton, & Sheppard, 2016). This allows us to specify rates λ_{X_i} only for the cases where a single parent is in state 1. This reduces the number of required parameters to be linear in the number of parents rather than exponential. The remainder of the parameters for the CIMs where multiple parents are in state 1 are accounted for within the Noisy-OR model.

6.4. Parameterization Example

We demonstrate how a CTBN constructed from a fault tree can be parameterized by providing an example of the rates that are required to specify the CIMs for the CTBN shown in Figure 2. This specified CTBN is later used in our query demonstrations in Section 8. We start with the four fault nodes F_1, F_2, F_3, and F_4. In general these can be parameterized using 8 rates, corresponding to the failure rates and repair rates. For this example, we assume that there is no repair policy in place, therefore all repair rates are 0. This reduces the number of necessary parameters, which now consist of the four parameters for each fault. The MTBF for each fault F_1, F_2, F_3, and F_4 is 1, 2, 4, and 8 respectively, measured in thousands of hours. This equates to failure rates of $\lambda_{F1} = 1.0$, $\lambda_{F2} = 0.5$, $\lambda_{F3} = 0.25$ and $\lambda_{F4} = 0.125$. This, along with the 0 valued repair rates, can be used to parameterize the faults as specified in Section 6.1.

Next, we parameterize the effects that were derived from AND gates in the fault tree. This consists of E_4 and E_6 (the dotted nodes in Figure 2). Each of these nodes can be parameterized using two rates. These values indicate the time it takes to transition to 1 when all parents are on, and the time it takes to transition to 0 otherwise. After the pruning process is applied, each node has only a single parent. These parameters $\lambda_{E4} = 0.3$, $\mu_{E4} = 1.1$, $\lambda_{E6} = 0.7$, and $\lambda_{E6} = 0.5$ are used to parameterize the four CIMs for E_4 and E_6 as shown in Section 6.2.

Finally, we must parameterize the effects that correspond to the OR gates in the fault tree. Specifically, we will need to parameterize E_1, E_2, E_3, and E_5 (the nodes outlined in dashed lines in Figure 2). Using the Noisy-OR model, we need only specify the rate for transitioning to state 0 given all parents are in state zero, as well as the rates for transitioning to state 1 given each individual parent being in state 1. Let $\lambda_{X|Y}$ be the rate at which X transitions to 1 given that only parent Y is in state 1. After pruning, E_1 has only four parents, resulting in the five required parameters $\mu_{E1} = 1.8$, $\lambda_{E1|E2} = 2.0$, $\lambda_{E1|E3} = 1.2$, $\lambda_{E1|F3} = 1.6$, $\lambda_{E1|E4} = 0.9$. E_2 is parameterized with rates $\mu_{E2} = 0.2$, $\lambda_{E2|F2} = 0.3$, $\lambda_{E2|E5} = 0.1$. E_3 requires rates $\mu_{E3} = 0.8$, $\lambda_{E3|F2} = 2.2$, $\lambda_{E3|F4} = 0.7$. Finally, E_5 has only a single parent, meaning it can be parameterized using only the rates $\mu_{E5} = 1.5$ and $\lambda_{E5|F1} = 1.8$.

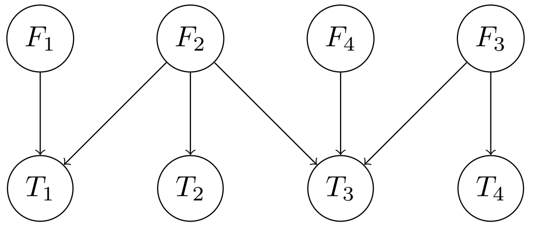

Figure 3. Example CTBN constructed from D-matrix.

These rates can be used to generate the $16 + 4 + 4 + 2 = 26$ CIMs required for the four OR-gate effect nodes by using the process described in Section 6.3.

7. COMBINING FAULT TREES AND D-MATRICES

We have shown how a CTBN can be derived from a fault tree. The resulting network consists of a set of fault nodes, as well as a set of effect nodes that depend on these faults. The constructed CTBN allows us to answer queries about the expected behavior of effects over time, given information we might know about the faults in the system.

In previous work, we have shown how to construct a CTBN from D-matrices, which results in a network consisting of fault and test nodes. Given a D-matrix D_S and a fault tree T_S that both describe a single system S, the set of faults is the same for both diagnostic models. A CTBN that models system S as a whole can be obtained by merging the two CTBNs derived from D_S and T_S.

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

We demonstrate by way of example. Let us refer to the fault tree used to construct the network in Figure 2 as F_S. Now let us assume that we also have a D-matrix available for the same system S, which we refer to as D_S. Figure 3 shows a CTBN that could have been created from D-matrix D_S. The networks in Figures 2 and 3 can be merged to form a single CTBN describing system S. This new CTBN consists of four faults, four tests, and six effects. No additional parameterization is required since all parent sets remain the same.

Table 1. Effect Parameters

	CIMs	Total Params	Noisy-OR Reduction	Gate Reduction	Reduced Total
Original	148	296	-250	-23	23
Pruned	30	60	-26	-17	**17**

8. QUERY DEMONSTRATIONS

In the following subsections, we demonstrate how a derived CTBN may be queried for information. To begin, we provide a brief description of the model that was constructed, and discuss the observed reductions to the required number of parameters. We then ensure that the constructed model adheres to the logical definitions provided by the original fault tree. Finally, we provide an example of how a technician might employ the constructed model to aid in the task of predicting the occurrence of effects.

8.1. Required Parameters

We ran Algorithm 1 on the fault tree from Figure 1. We then derived a CTBN as described in Section 5, and parameterized the model using the rates from Section 6.4. Table 1 describes this network and provides attributes that demonstrate the benefits of the pruning process and parameter reduction techniques. The constructed network is shown by Figure 2, and contains four faults and six effects. Each of the four faults has a single intensity matrix associated with it, which can be parameterized using basic reliability information. Since the complexity of the model is driven by the number of parameters required for the effects, we restrict our focus to these CIMs only. Table 1 shows how many parameters are necessary to specify the effects in the network, and how the number of parameters is affected by the techniques discussed so far.

Note that the total number of parameters required for the CTBN is double the number of CIMs. This is because we are working with binary variables, and as such each CIM only requires two off-diagonal rates. Next we observe that although there is a significant difference between the number of CIMs in the original fault tree as compared to the pruned version, the application of the Noisy-OR model narrows the gap between the number of required parameters considerably. While this may be the case for this particular model, it does not hold in general. When a fault tree has an effect determined by an AND gate, the Noisy-OR model cannot be applied, in which case the pruning process may be crucial in reducing the number CIMs. Finally, we observe that by making use of the information encoded by the gates in the fault tree, the number of required parameters is halved. This is because the gate structure requires rates of 0.0 for states that do not conform to the output of the corresponding discrete function, leaving only a single required parameter per CIM.

8.2. Behavior Validation

To validate the network structure derivation and parameterization process, we perform a series of queries that test for the expected behavior already discussed. Recall that a fault will transition between failing and non-failing states according to failure rates and repair rates, with no additional dependencies on other variables in the network. For this reason we are more interested in the effects in the network. A model that properly encodes the information of a fault tree will ensure that effects eventually reach a state that corresponds to the output of the discrete gate function it is associated with. As discussed in Section 6, with a fixed state instantiation of parents, $\lim_{t\to\infty} P(X(t) = F_X(Pa(X))) = 1.0$.

To demonstrate this behavior, we assign evidence E to the parents of an effect X. We do this for all combinations of parent instantiations, and in each case assign a deterministic initial distribution to X that conforms to the gate output $F_X(E)$. We then query for the probability of X at several discrete timesteps to observe its behavior. We used Importance Sampling as the underlying inference algorithm, which uses weighted samples to provide approximate solutions to the submitted queries (Fan & Shelton, 2008). In all cases, we ran inference until $10,000$ samples were generated.

Table 2 shows these queries for effects E_3 and E_4. The first column indicates which variable is being queried, while the second column describes the evidence that was applied to the parents over the interval of interest. The third column shows the output of the logical gate function F_X, which is dependent on the evidence as well as the gate type for the effect. The last four columns show the probability that effect X is in the state corresponding to the gate output at the specified times, or $P(X(t) = F_X(E))$. For this experiment, we queried this probability distribution at times $t = 0.1, 1, 10,$ and 100. Note that due to the conflicting prior distribution, the probability at time $t = 0$ is always 0.0.

Note that in all cases, the probabilities start near 0.0, and approach 1.0 as time increases. Although the table only shows queries performed on effects E_3 and E_4, similar results were obtained for the remaining four effects. By time $t = 100$, all effects in the network had a 1.0 probability of being in the state corresponding to the logical gate output. The differences in the time it takes to reach this level of certainty stems from the differences in the rates that were used to specify the corresponding CIMs. This experiment demonstrates that the model exhibits the desired behavior, which supports the correctness of the structure derivation and parameterization process.

Table 2. Queries of Synthetic Network

Effect	Evidence	Gate Output	$t = 0.1$	$t = 1$	$t = 10$	$t = 100$
E_3 (OR)	$F_2 = 0, F_4 = 0$	F	0.073	0.531	0.999	1.0
E_3 (OR)	$F_2 = 0, F_4 = 1$	T	0.067	0.499	0.997	1.0
E_3 (OR)	$F_2 = 1, F_4 = 0$	T	0.198	0.901	1.0	1.0
E_4 (AND)	$E_6 = 0, F_3 = 0$	F	0.105	0.668	1.0	1.0
E_4 (AND)	$E_6 = 0, F_3 = 1$	F	0.111	0.671	1.0	1.0
E_4 (AND)	$E_6 = 1, F_3 = 0$	F	0.083	0.687	1.0	1.0
E_4 (AND)	$E_6 = 1, F_3 = 1$	T	0.030	0.255	0.935	1.0

8.3. Use Case

Although applying evidence to the parent set of an effect is able to demonstrate the desired behavior, in practice the faults and effects cannot be observed directly. For this reason, we turn our attention to the CTBN obtained from merging the models derived from D-matrices and fault trees. This process introduces tests into the model, thereby incorporating directly observable variables. Evidence can be applied to these new nodes based on the test outcomes, and queries can be constructed that determine the probability of faults and effects through time.

To start, we derive a CTBN from the D-matrix shown below. This results in the network structure shown in Figure 3. We parameterized the model using failure and repair rates, along with false alarm and non-detect probabilities as described in previous work (Perreault et al., 2015). All tests and faults are assumed to start in a working state at time 0, therefore the initial probability distribution is set to $(1.0, 0.0)$. We use the failure and repair rates that were discussed in Section 6.4. The non-detect probabilities for the tests are set to 0.02, 0.02, 0.01, and 0.01, while the false alarm probabilities are 0.01, 0.04, 0.02, and 0.01.

$$D = \begin{array}{c} \\ F_1 \\ F_2 \\ F_3 \\ F_4 \end{array} \begin{array}{c} \begin{array}{cccc} T_1 & T_2 & T_3 & T_4 \end{array} \\ \begin{pmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 \end{pmatrix} \end{array}$$

The resulting CTBN is merged with the fault tree derived CTBN from the previous experiment as described in Section 7. This model is similar to the fault tree derived network, except that there are now four additional nodes corresponding to the tests defined in the D-matrix. Without running any tests, we can query the model to observe the expected behavior of each test, fault, or effect. For example, a technician may first query the probability of each effect at time $t = 1.0$ in the future. These probabilities are shown in the left-most group in Figure 4. Note that E_4 has been omitted from this chart for clarity, since each probability was approximately 0.0.

Based on these results, there is no one effect that appears particularly likely. Given this, the technician may decide to run test T_3 and observe a failed outcome. By applying evidence that T_3 is in a failed state at time $t = 0$, we obtain the new probabilities for each effect shown in the second group in Figure 4. These new probabilities do not offer any additional clarity, as there appears to be a fairly uniform increase in each effect's likelihood.

Next the technician runs test T_1 and determines that it passes. After applying this evidence in addition to the evidence for T_3, we are left with the new distribution shown in the third group in Figure 4. Given this evidence, we see that E_1, and to a lesser extent E_6, are likely to occur at time $t = 1.0$ in the future. The remaining effects have all become relatively unlikely.

Finally, the technician may run test T_4 and observe a passing outcome. This drastically changes the distribution again, showing E_3 as being very probable, with all remaining effects being extremely unlikely. The magnitude of the change is due to the fact that the passing outcome for T_4 was itself unlikely given the previous evidence. This highlights the necessity of probabilistic models like CTBNs, which are capable of encoding more information than a D-matrix or a fault tree alone.

9. CONCLUSION

In this work, we have demonstrated how the structure of a fault tree can be mapped to the network structure of a CTBN. Furthermore, we show that the resulting CTBN can be specified using a reduced set of parameters. This is accomplished by taking advantage of attributes of the associated fault tree. We show how tests can be introduced into this model and demonstrate the way in which the model might be used in practice. For future work, we hope to use the structure of a multi-level fault tree to encode the behavioral relationship between a child and its parents. We intend to use this as a parameter reduction technique similar to Noisy-OR, which would allow for more compact CTBN representations.

CTBNs have been applied successfully to several practical

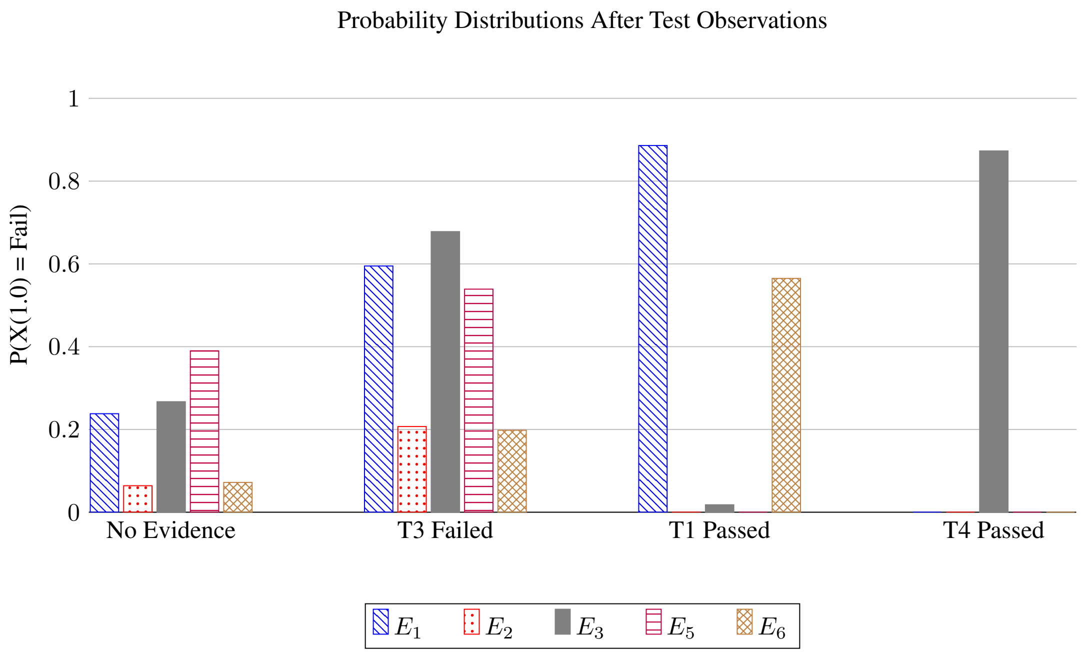

Figure 4. Use Case Query Results

domains where discrete systems evolve in continuous time. To date, use of CTBNs has been limited within the PHM community, and we contend that practitioners are hesitant to adopt this model due to the labor-intensive process associated with the construction and parameterization of the model. By automating the structure generation process and reducing the number of required parameters, the process of constructing a prognostic CTBN is greatly simplified. We hope that this work will encourage the adoption of CTBNs for the task of modeling discrete state systems.

ACKNOWLEDGMENTS

This work was funded in part by a NASA STTR under contract NNX14CJ45C. We would also like to thank the members of the Numerical Intelligent Systems Laboratory (NISL) at Montana State University for their support.

REFERENCES

Bobbio, A., Portinale, L., Minichino, M., & Ciancamerla, E. (2001). Improving the analysis of dependable systems by mapping fault trees into Bayesian networks. *Reliability Engineering & System Safety*, *71*(3), 249–260.

Cao, D. (2011). *Novel models and algorithms for systems reliability modeling and optimization* (Dissertation). Wayne State University.

Fan, Y., & Shelton, C. R. (2008). Sampling for approximate inference in continuous time Bayesian networks. In *Tenth international symposium on artificial intelligence and mathematics.*

Montani, S., Portinale, L., & Bobbio, A. (2006). Dynamic Bayesian networks for modeling advanced fault tree features in dependability analysis. In *Proceedings of the European safety and reliability conference* (p. 1415-1422).

Nodelman, U., Shelton, C. R., & Koller, D. (2002). Continuous time Bayesian networks. In *Proceedings of the eighteenth conference on uncertainty in artificial intelligence* (pp. 378–387).

Perreault, L., Strasser, S., Thornton, M., & Sheppard, J. W. (2016). A noisy-or model for continuous time Bayesian networks. In *Proceedings of the Florida artificial intelligence symposium.* (To Appear)

Perreault, L., Thornton, M., Strasser, S., & Sheppard, J. (2015). Deriving prognostic continuous time Bayesian networks from D-matrices. In *IEEE AUTOTESTCON Conference Record.*

Ruijters, E., & Stoelinga, M. (2015). Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, *1516*, 29 - 62.

Sheppard, J., & Butcher, S. (2007). A formal analysis of fault diagnosis with D-matrices. *Journal of Electronic Testing*, *23*(4), 309–322.

Simpson, W. R., & Sheppard, J. W. (1994). *System test and diagnosis*. Norwell, MA: Kluwer Academic Publishers.

Strasser, S., & Sheppard, J. (2013). An empirical evaluation of Bayesian networks derived from fault trees. In *Proceedings of the IEEE aerospace conference* (pp. 1–13).