Proc. of the European Safety and Reliability Conference (ESREL 2026)
Edited by José C. Matos, Paulo B. Lourenço, Michael Beer, Daniel Oliveira,
Edoardo Patelli, Hélder S. Sousa, Rui A. Silva and Monica Santamaria
 Published by Research Publishing, Singapore.
doi: 10.3850/ESREL2026061419_esrel26-p26220-cd

Dynamic Fault Tree Construction Through Minimal Cut Sequences And Causal Discovery Algorithm

Stanley Suan You Lim and Eric Gascard
Univ. Grenoble Alpes, CNRS, Grenoble INP ⋆, G-SCOP, 38000 Grenoble, France
⋆ Institute of Engineering Univ. Grenoble Alpes
E-mail: stanley-suan-you.lim@grenoble-inp.fr

Ensuring system reliability and availability is critical for maintaining operational continuity and supply chain stability in industrial systems, where unexpected downtimes can lead to significant production losses and cascading disruptions. Fault tree analysis is a well-established reliability engineering method for identifying system vulnerabilities and evaluating failure risks. While traditional fault trees are static, Dynamic Fault Trees (DFTs) extend this framework by modeling failure dependencies and sequences, which are essential for representing complex systems with redundancy and interdependent components. However, the manual construction of DFTs remains a major limitation due to its complexity, time consumption, and susceptibility to human error.

This paper proposes a data-driven framework for the automatic construction of DFTs based on the learning of Minimal Cut Sequences (MCSQs) from failure-time data. The approach extracts MCSQs and groups them according to their constituent events. To characterize temporal dependencies, the concepts of Positions of Elements and Relative Positions are introduced to qualitatively capture sequence relationships among failures. These constructs enable the inference of both static and dynamic gates that collectively represent the complete set of MCSQs in a DFT. In addition, a causal discovery algorithm is incorporated to identify causal relationships that cannot be revealed through qualitative analysis alone. The proposed framework facilitates scalable and automated DFT construction, thereby enhancing the applicability of fault tree analysis to complex, data-rich industrial systems.

Keywords: Dynamic Fault Tree Construction, Minimal Cut Sequences, Causal Discovery Algorithm, Data-Driven Approach

1. Introduction

Ensuring system reliability and availability is fundamental to maintaining smooth operations and preserving supply chain stability in industrial environments. Unexpected system downtime can cause disruptions through the supply chain network, which may cascade into significant financial losses. To prevent such cascading failures, industries place strong emphasis on reliability engineering methods that enable early detection, modeling, and mitigation of risks before they manifest as costly failures. Among the classic approaches, Fault Tree (FT) analysis (Vesely et al., 1981) is a popular method that provides a structured framework to identify design vulnerabilities and estimate system reliability.

In FT analysis, logic gates are used to express the relationships between component failures (called basic events, BEs) and how failures propagate to cause defined critical system failure (called top event, TE). A static FT is composed of static logical gates such as AND, OR, and NOT that connect BEs to the TE. While static fault trees provide a snapshot of failure combinations, they cannot represent sequential failures, spare components, or functional dependencies. To address these limitations, Dugan et al. (1990) introduced the Dynamic Fault Tree (DFT) formalism, which extends static FTs by incorporating dynamic gates such as Priority AND (PAND), SPARE, and Functional Dependency (FDEP) gates. This enriches the expressiveness of the model and increases the modeling accuracy of the actual failure mechanisms.

Despite the usefulness of FTs, their initial construction remains a demanding task. Although static FTs can often be derived directly from the structural configuration of system components and their interrelationships, DFTs require a more sophisticated treatment of event sequences, temporal dependencies, and dynamic interactions among components. As a result, manual DFT construction is time-consuming and susceptible to human error.

Meanwhile, the increasing integration of sensor technologies and advances in computational capa-

bilities have resulted in the widespread availability of operational and failure data in modern industrial systems. This abundance of data, combined with growing system complexity, has motivated our development of a data-driven approach for the automatic construction of DFTs which will be presented in this article.

The proposed approach is implemented as a Python algorithm that constructs the DFT through the sequence analysis of Minimal Cut Sequences (MCSQs) derived from failure time data. To enable such analysis, the concepts of Positions of Elements and Relative Positions will be introduced. Our approach also integrates the Peter–Clark algorithm (Spirtes et al., 2000) to discover causal dependencies, as well as a linear regression model to identify trigger-dependent relationships between events.

The remainder of this article is organized as follows. Section 2 presents a brief review of data-driven approaches to FT construction. Section 3 introduces the assumptions underlying the proposed algorithm. Based on these assumptions, qualitative sequence analysis and causal discovery are presented in Sections 4 and 5 respectively. Section 6 describes the priority of the gate discovery process. Section 7 provides illustrations of the proposed methods. Finally, Section 8 concludes the article and outlines directions for future research.

2. State of the Art in Data-Driven Fault Tree Construction

In the literature, FT construction methods are generally classified into knowledge-based, model-based, and data-driven approaches. Owing to space constraints, this section discusses only a limited number of representative data-driven methods. Readers interested in a more comprehensive overview of FT construction techniques are referred to Lim and Gascard (2025).

Data-driven FT construction techniques commonly rely on relationship-mining methods, evolutionary computation, and machine learning techniques to infer FT structures from data. Among these approaches, the LIFT algorithm (Nauta et al., 2018) using the Mantel-Haenszel statistical test is widely recognized for its capability to infer causal relationships and construct static FTs. Logical Analysis of Data (Boros et al., 2000) is a rule-based supervised learning method that extracts logical patterns from datasets. Building on this principle, Waghen and Ouali (2021) proposed the Multi-level Interpretable Logic Tree Analysis, which constructs FTs based on the patterns identified.

Inspired by biological evolution, Linard et al. (2019) developed an evolutionary algorithm that iteratively evolves static FT structures using randomized genetic operators. However, optimizing only the TE prediction often leads to structural inconsistencies and high computational costs. To address these limitations, Jimenez-Roa et al. (2022) proposed Multi-Objective evolutionary algorithm that finds optimal solutions in terms of accuracy and the generated FT structure.

A variety of machine learning techniques have been explored for FT construction. For example, Naive Bayes classifiers have been used to predict failure behaviors that may not be explicitly known a priori (Niloofar and Lazarova-Molnar, 2023). Text mining techniques have been applied to analyze company knowledge repositories to identify failure events and derive FTs (Mukherjee and Chakraborty, 2007). In addition, Bayesian Networks can be learned from failure data and subsequently transformed into FT representations (Linard et al., 2019).

Although data-driven FT construction methods exist, they remain limited to static FTs. Our proposed methods address this scientific gap by enabling automated DFT construction.

3. Assumptions and MCSQs Extraction

3.1. *Dataset assumptions*

The proposed algorithm is developed under the following assumptions regarding the input dataset:

(i) The input dataset consists of events associated with continuous failure times measured on a consistent time scale.
(ii) Each row of the dataset represents an independent failure instance.
(iii) Complete failure behaviors of the system is captured.
(iv) Events are non-repairable.
(v) For each failure instance, the failure times of events that occur prior to the TE are recorded, whereas events that do not fail are assigned a value of -1.
(vi) Basic events (BEs) model failures of non-redundant components, while spare components (SPs) model failures of redundant components. The two are distinguished by naming

prefixes in the dataset.

3.2. *MCSQs extraction*

In DFT analysis, Cut Sequences (CSQs) (Tang and Dugan, 2004) represent ordered failure sequences leading to the occurrence of the defined TE, while Minimal Cut Sequences (MCSQs) denote the shortest sequences. Our algorithm derives CSQs by first filtering the dataset to retain only rows where the TE occurs. Each remaining row thus represents a sequence of failures causing the TE. The failed events in each row are then ordered by their failure times to form a CSQ. After processing all rows, the resulting collection constitutes the set of CSQs. Please note that the operator '$\prec$' is used in this article to denote the order of occurrence.

Nevertheless, when multiple events share the same failure time (e.g., due to a common cause), the algorithm by default orders them from the leftmost to the rightmost column. Thus, to ensure meaningful ordering, a preprocessing step is applied to rearrange the columns (except the TE) in ascending order of their mean failure time before initiating the CSQs extraction process. Finally, CSQs containing shorter CSQs as subsequences in the collection are eliminated, and the remaining collection is identified as the MCSQs collection.

Table 1. Example failure dataset.

Row	BE_W	BE_X	BE_Y	BE_Z	SP_A	TE
1	-1	0.1	0.62	-1	-1	0.62
2	-1	2.42	1.05	-1	-1	2.42
3	4.77	-1	-1	-1	2.5	4.77
4	3.15	-1	-1	-1	4.21	4.21
5	-1	0.77	3.33	0.76	-1	3.33

For example, a collection of CSQs can be extracted from TABLE 1:

$$\begin{aligned}\text{CSQs} = \{&[\mathrm{BE}_X \prec \mathrm{BE}_Y] \wedge [\mathrm{BE}_Y \prec \mathrm{BE}_X] \wedge \\ &[\mathrm{SP}_A \prec \mathrm{BE}_W] \wedge [\mathrm{BE}_W \prec \mathrm{SP}_A] \wedge \\ &[\mathrm{BE}_Z \prec \mathrm{BE}_X \prec \mathrm{BE}_Y]\}\end{aligned}$$

By removing non minimal CSQs, an MCSQs collection can be obtained:

$$\begin{aligned}\text{MCSQs} = \{&[\mathrm{BE}_X \prec \mathrm{BE}_Y] \wedge [\mathrm{BE}_Y \prec \mathrm{BE}_X] \wedge \\ &[\mathrm{SP}_A \prec \mathrm{BE}_W] \wedge [\mathrm{BE}_W \prec \mathrm{SP}_A]\}\end{aligned}$$

Lastly, to systematically uncover sequence dependencies among events, MCSQs containing the exact same set of events but differing in their order of occurrence are grouped together in G_i:

$$\begin{aligned}\mathrm{G}_1 &= \{[\mathrm{BE}_X \prec \mathrm{BE}_Y] \wedge [\mathrm{BE}_Y \prec \mathrm{BE}_X]\} \\ \mathrm{G}_2 &= \{[\mathrm{SP}_A \prec \mathrm{BE}_W] \wedge [\mathrm{BE}_W \prec \mathrm{SP}_A]\}\end{aligned}$$

4. Qualitative Sequence Analysis

Within each MCSQs group, the presence or absence of specific event orderings provides qualitative evidence to identify underlying sequence relationships. This analysis will be carried out iteratively and aims to infer logical relationships between events and replace them with the corresponding logic gates.

As a result, each iteration of the analysis reduces the length of each MCSQs, while duplicating MCSQs will be removed from the group. This process ensures that the evolving MCSQs group consistently reflects the discovered logical structure and increases the depth of the DFT. Assuming that the dataset perfectly captures all failure behaviors, every MCSQs group will ultimately converge to a single logic gate representing a local sub-tree that connects all elements from the group. Finally, after processing all groups, the resulting logic gates along with the BEs from single-event groups are combined under a top OR gate to form the complete DFT and output in Galileo textual format.

In the following subsections, the concepts of Positions of Elements and Relative Positions that are used to capture logical relationships and sequential patterns between elements will be presented.

4.1. *Positions of Elements*

Before diving deeper into the concept of Positions of Elements (PoE), we would like to clarify that throughout this article, "events" are used to denote BEs and SPs, whereas "elements" refer collectively to events and identified logic gates. To identify sequential patterns, the positional order of elements must be systematically recorded and analyzed. This information is stored in an associative array (i.e., a Python dictionary) referred to as the Positions of Elements (PoE). In this structure, each element e is used as a key, and its associated value is a list of integers representing the positional indices of e across MCSQs within a given group G_i.

The PoE has several applications, including distinguishing characteristics of spare components.

Specifically, if position zero does not appear in the PoE value list of a spare component, it indicates that the spare component never fails in its dormant state before its activation triggered by the failure of its primary event. This spare component is modeled as the spare component of a Cold Spare (CSP) gate in a DFT. By contrast, it indicates that a spare component may fail while partially active without prior failure of its primary event. This type of spare component is modeled as a warm spare and represented by a Warm Spare (WSP) gate in the DFT.

Another application of PoE is the identification of exchangeable and sequence-specific elements. Elements are considered exchangeable when they can interchange their order of occurrence while still leading to the TE. They can be determined by finding exemplary MCSQs where elements swap their orders within the group. In the proposed algorithm, each element's position list in the PoE will first be sorted, elements having the same lists are then classified as exchangeable and connected through an AND gate to reflect their logical equivalence in terms of sequence order.

Conversely, the presence of lists with specific positional patterns in the PoE indicates sequence-dependent constraints, implying that the occurrence of the TE depends on the order of these elements. Such patterns typically arise after substituting spare components and primary events with spare gates, as well as replacing exchangeable elements with AND gates. This sequence dependency can be expressed in the DFT by using a PAND gate.

4.2. *Relative Positions of events*

Although the PoE is capable to identify exchageable elements and spare types, the Relative Positions (RP) serves as an effective complementary analysis to determine whether a cold spare component is shared among multiple primary events. The RP is represented as an associative array, where each key corresponds to a pair of elements, and the associated value is a Boolean value reflecting their relative ordering in the PoE. Sequential relationships between elements are identified when the value list exhibits a consistent pattern (i.e., only having '1' or '0' in the list).

A cold spare with a consistent RP value indicates that it activates and fails only after the failure of a specific event, thereby revealing its primary event. They can thus be associated with a CSP gate. In contrast, a shared cold spare component will produce inconsistent RP values, reflecting its activation and failure after different events across sequences. To determine the primary events of a shared cold spare component, one can examine the PoE and select the sequences where the spare component fails at position '1' (earliest failure order for a cold spare component by definition), the BEs that fail at position '0' in those sequences are the events that activate the spare component thus its primary events. In a shared spare configuration, a number of spare gates corresponding to the number of primary events will be generated, and the shared spare component will be linked to each of them.

5. Quantitative Analysis

While PoE and RP support associating primary events with spare components in CSPs and identifying exchangeable elements connected by AND relationships, they remain insufficient for determining primary events in WSPs whose ability to fail independently during dormancy prevents sequence-based identification of primary events, and for capturing trigger-dependent relationships that requires statistical analysis.

This section therefore discusses quantitative analysis methods that exploit the causal characteristics of changing failure rates when a warm spare component is activated (Gascard and Simeu-Abazi, 2018), as well as methods that capture trigger-dependent relationships. The scope of the quantitative analysis presented here is to identify relationships between events in order to associate them with logic gates, rather than to provide statistical information.

5.1. *Peter-Clark algorithm*

The Peter-Clark (PC) algorithm is designed to identify causal links among a group of variables by initially assuming that all variables are connected in an undirected graph. It employs numerous conditional independence tests across different combinations of conditioning sets to remove unnecessary edges, then applies logical orientation rules to detect colliders and construct a graph consistent with observed conditional independencies. This is necessary to detect a collider structure (Molak, 2023) within the causal graph as it implies multiple events can cause a common event to occur, i.e., a scenario where a warm spare component is shared among multiple primary events.

In our algorithm, we integrated the PC causal discovery algorithm from the Salesforce's CausalAI library (Arpit et al., 2023) into our framework to identify primary events for warm spares. This library is chosen for its support of nonlinear and nonparametric Kernel-based conditional independence tests. This feature holds significant importance as most traditional conditional independence tests assume data is Gaussian and relationships are linear, which is seldom true for failure data. Before applying the PC algorithm, the columns of the dataset are filtered to retain only the events present in the MCSQ, and any rows containing null values have to be removed. After running the algorithm, the 'parents' in the analysis results are the primary events for the spare component. This method can also be applied to hot spares (HSPs), which are regarded in this article as a special case of WSP with a dormancy factor equal to one.

In our controlled experiments, we observed that the p-value hyperparameter (0.05 by default) often needs to be relaxed when detecting causal relationships of a warm spare component with multiple primary events. Our hypothesis to explain this phenomenon is that as the number of contributing primary events increases, it tends to dilute the statistical strength of individual associations, which leads to higher p-values in the tests.

5.2. *Linear Causal Dependency Modeling*

In complex systems, some component failures can be caused not only by their own internal failure mechanisms but also by the failure of other components. These induced failures can be formally represented in DFTs using a FDEP gate. A key feature of a trigger–dependent event pair is that the dependent event can still fail on its own, even if the trigger event does not occur. As a result, any failure sequence that includes both the trigger and its dependent event must also include the simpler sequence where the dependent event fails alone.

This property can be used to identify possible trigger events: by finding events that appear in CSQs but are missing from MCSQs (e.g., BE_Z from TABLE 1). Once a candidate trigger is identified, all events that co-occur with it in the same CSQs can be treated as potential dependent events.

Taking the example of Fig.1, the CSQs obtain will be:

$$\mathrm{CSQs} = \left\{[\mathrm{BE}_T \prec \mathrm{BE}_D] \wedge [\mathrm{BE}_D \prec \mathrm{BE}_T] \wedge [\mathrm{BE}_D]\right\}$$

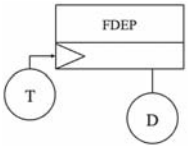

Fig. 1. FDEP gate example

Since the failure sequence $[\mathrm{BE}_D]$ is a subsequence of both $[\mathrm{BE}_T \prec \mathrm{BE}_D]$ and $[\mathrm{BE}_D \prec \mathrm{BE}_T]$, the only MCSQ here is $[\mathrm{BE}_D]$, thus BE_T appears in the CSQ set but not in the MCSQ set, making it a candidate trigger event, while BE_D as a candidate dependent event.

Now the objective is to evaluate the existence of a FDEP relationship between a trigger candidate BE_T and one of its candidate dependent event BE_D, as expressed in Eq.1 below. To assess this dependency, the failure time dataset is conditioned to include instances in which both events occur, with the candidate dependent event taking place after the trigger event. The relationship is subsequently modeled using Ordinary Least Squares (OLS) regression, wherein the trigger event's failure time is treated as the predictor variable and the dependent event's failure time as the response variable. The temporal delay between the two events is represented as a constant offset in the model. The goodness of fit is assessed using the R-squared and p-values, which together provide evidence for or against the presence of an FDEP relationship.

$$\begin{aligned} \mathrm{T}_{dependent} =& \mathrm{T}_{trigger} + \mathrm{T}_{delay}, \\ & \mathrm{T}_{dependent} \geq \mathrm{T}_{trigger} \end{aligned} \qquad \text{(Eq.1)}$$

where:

(1) $T_{dependent}$: Failure time of the dependent event
(2) $T_{trigger}$: Failure time of the trigger event
(3) T_{delay}: Delay between the failure time of the dependent event and the trigger event

A limitation of this method is that it cannot detect FDEP relationships that require the failure of multiple trigger events. For example, if the dependent event is induced only when two or more trigger events occur together, this approach fails to identify the FDEP relationship.

6. Priority of Gate Discovery

The preceding sections introduced techniques for identifying sequential patterns and logical relationships then associate the corresponding logic gate between events. However, when dealing with longer MCSQs where sequences contain three or more events, these techniques must be applied in a specific order to fully capture the complexity of event interactions.

The procedure begins by encoding the PoE. Next, the algorithm checks whether any event in the MCSQs is a spare component based on its naming prefix. If a spare component is found, its PoE is used to determine its type. If the spare is deemed as a cold spare component, the RP is analyzed to verify whether it is a shared spare component, and the PoE is used to find its primary event(s). Conversely, the PC algorithm is used to identify primary event(s) for the warm spare component.

After pairing the spare component with its primary event(s), the algorithm updates the MCSQs group by replacing involved events with spare gate in the MCSQs group. It then recalculates the PoE and identifies if any events or gates are exchangeable in the updated MCSQs group and update the group again with AND gate. Finally, if the updated MCSQs group still contains more than one event or gate and their order is fixed, a PAND gate is assigned to represent their sequence dependency.

Then, distributive laws are applied to simplify and optimize the logical structure (A.B $\vee$ A.C $\rightarrow$ A.(B $\vee$ C)) with the BEs connected to AND gates. This transformation allows the creation of OR gates that factor out common events, reducing redundancy in the representation of AND gates. For instance: AND_0: [A, B], AND_1: [A, C] will become AND_0: [A, OR_0], OR_0: [B, C].

Nevertheless, modeling linear causal dependencies does not need to follow a specific order. It can be performed independently as a separate test to identify events that exhibit trigger-dependent relationships after CSQs are distilled into MCSQs.

Finally, the logic gates representing different MCSQs groups, and the MCSQs consisting of one BE are connected to a top-level OR gate to construct the DFT whenever more than one gate is present. If only a single gate is identified from the MCSQs groups, that gate becomes the top gate directly, eliminating the need for an additional OR gate.

7. Case Studies

This section presents three case studies demonstrating the application of PoE and RP. Specifically, we illustrate how these concepts enable the identification of sequence-specific MCSQs, determination of primary events in both non-shared and shared cold spare component configurations.

7.1. *Sequence-specific and non sequence-specific MCSQs*

$$\begin{aligned} \text{G}_i = \{&[\text{BE}_A \prec \text{BE}_B \prec \text{BE}_C] \wedge \\ &[\text{BE}_A \prec \text{BE}_C \prec \text{BE}_B]\} (1) \end{aligned}$$

$$\text{PoE}_i = \{\text{BE}_A : [0,0], \text{BE}_B : [1,2], \text{BE}_C : [2,1]\}$$

$$\begin{aligned} \text{PoE}_{isorted} = \{&\text{BE}_A : [0,0], \\ &\text{BE}_B : [1,2], \\ &\text{BE}_C : [1,2]\} (2) \end{aligned}$$

$$\begin{aligned} \text{AND} &= \{\text{AND}_0 : [\text{BE}_B, \text{BE}_C]\} (2) \\ \text{G}_{iupdated} &= \{[\text{BE}_A \prec \text{AND}_0]\} (3) \\ \text{PAND} &= \{\text{PAND}_0 : [\text{BE}_A, \text{AND}_0]\} (3) \end{aligned}$$

Analysis:

(1) No spare components are found from the MCSQs group.
(2) BE_B and BE_C are exchangeable since their sorted PoE are the same, thus having an AND relationship.
(3) After updating the MCSQs group, AND_0 always occurs after BE_A as there is a lack of a counter example, indicating a PAND relationship.

7.2. *Identification of primary event of cold spare component*

$$\begin{aligned} \text{G}_i = \{&[\text{BE}_A \prec \text{BE}_B \prec \text{SP}_S] \wedge \\ &[\text{BE}_B \prec \text{SP}_S \prec \text{BE}_A] \wedge \\ &[\text{BE}_B \prec \text{BE}_A \prec \text{SP}_S]\} (1) \end{aligned}$$

$$\begin{aligned} \text{PoE}_i = \{&\text{BE}_A : [0,2,1], \\ &\text{BE}_B : [1,0,0], \\ &\text{SP}_S : [2,1,2] (2)\} \end{aligned}$$

$$\begin{aligned} \text{RP}_i = \{&(\text{BE}_A, \text{BE}_B) : [0,1,1] \\ &(\text{BE}_A, \text{SP}_S) : [0,1,0] \\ &(\text{BE}_B, \text{SP}_S) : [0,0,0] (3)\} \end{aligned}$$

$$\text{CSP} = \{\text{CSP}_0 : [\text{BE}_B, \text{SP}_S]\}(3)$$
$$\text{G}_{iupdated} = \{[\text{BE}_A \prec \text{CSP}_0] \wedge [\text{CSP}_0 \prec \text{BE}_A]\}$$
$$\text{PoE}_{iupdated} = \{\text{BE}_A : [0,1],\ \text{CSP}_0 : [1,0]\}(4)$$
$$\text{PoE}_{iupdatedsorted} = \{\text{BE}_A : [0,1],\ \text{CSP}_0 : [0,1]\}(4)$$
$$\text{AND} = \{\text{AND}_0 : [\text{BE}_A, \text{CSP}_0]\}(4)$$

Analysis:

(1) A spare component is present in this MCSQs group.
(2) The position zero is missing from the PoE of SP_S, indicating that it behaves as a cold spare component.
(3) In the RP analysis, SP_S consistently fails after BE_B, so BE_B is identified as the activating event of SP_S.
(4) BE_A and CSP_0 are exchangeable, therefore connected by an AND relationship.

7.3. *Identification of primary events of shared cold spare component*

$$\begin{aligned}\text{G}_i = \{&[\text{BE}_A \prec \text{SP}_S \prec \text{BE}_B \prec \text{BE}_C] \wedge \\ &[\text{BE}_B \prec \text{SP}_S \prec \text{BE}_A \prec \text{BE}_C] \wedge \\ &[\text{BE}_A \prec \text{BE}_B \prec \text{SP}_S \prec \text{BE}_C] \wedge \\ &[\text{BE}_B \prec \text{BE}_A \prec \text{SP}_S \prec \text{BE}_C] \wedge \\ &[\text{BE}_C \prec \text{BE}_A \prec \text{SP}_S \prec \text{BE}_B] \wedge \\ &[\text{BE}_C \prec \text{BE}_B \prec \text{SP}_S \prec \text{BE}_A] \wedge \\ &[\text{BE}_C \prec \text{BE}_A \prec \text{BE}_B \prec \text{SP}_S] \wedge \\ &[\text{BE}_C \prec \text{BE}_B \prec \text{BE}_A \prec \text{SP}_S] \wedge \\ &[\text{BE}_A \prec \text{SP}_S \prec \text{BE}_C \prec \text{BE}_B] \wedge \\ &[\text{BE}_B \prec \text{SP}_S \prec \text{BE}_C \prec \text{BE}_A] \wedge \\ &[\text{BE}_A \prec \text{BE}_C \prec \text{BE}_B \prec \text{SP}_S] \wedge \\ &[\text{BE}_B \prec \text{BE}_C \prec \text{BE}_A \prec \text{SP}_S]\}\end{aligned}$$

$$\begin{aligned}\text{PoE}_i = \{&\text{BE}_A : [0,2,0,1,1,3,1,2,0,3,0,2], \\ &\text{BE}_B : [2,0,1,0,3,1,2,1,3,0,2,0], \\ &\text{BE}_C : [3,3,3,3,0,0,0,0,2,2,1,1], \\ &\text{SP}_S : [1,1,2,2,2,2,3,3,1,1,3,3]\}(1) \\ &(3,4)\end{aligned}$$

$$\begin{aligned}\text{RP}_i = \{&(\text{BE}_A, \text{BE}_B) : [0,1,0,1,0,1,0,1,0,1,0,1], \\ &(\text{BE}_A, \text{BE}_C) : [0,0,0,0,1,1,1,1,0,1,0,1], \\ &(\text{BE}_A, \text{SP}_S) : [0,1,0,0,0,1,0,0,0,1,0,0], \\ &(\text{BE}_B, \text{SP}_C) : [0,0,0,0,1,1,1,1,0,1,0,1], \\ &(\text{BE}_B, \text{SP}_S) : [1,0,1,0,1,0,0,0,1,0,0,0], \\ &(\text{BE}_C, \text{SP}_S) : [1,1,1,1,0,0,0,0,1,1,0,0]\} \\ &(2)\end{aligned}$$

$$\begin{aligned}\text{CSP} = \{&\text{CSP}_0 : [\text{BE}_A, \text{SP}_S], \\ &\text{CSP}_1 : [\text{BE}_B, \text{SP}_S]\}(5)\end{aligned}$$

$$\begin{aligned}\text{G}_{iupdated} = \{&[\text{CSP}_0 \prec \text{CSP}_1 \prec \text{BE}_C] \wedge \\ &[\text{CSP}_1 \prec \text{CSP}_0 \prec \text{BE}_C] \wedge \\ &[\text{BE}_C \prec \text{CSP}_0 \prec \text{CSP}_1] \wedge \\ &[\text{BE}_C \prec \text{CSP}_1 \prec \text{CSP}_0] \wedge \\ &[\text{CSP}_0 \prec \text{BE}_C \prec \text{CSP}_1] \wedge \\ &[\text{CSP}_1 \prec \text{BE}_C \prec \text{CSP}_0]\}(6)\end{aligned}$$

$$\begin{aligned}\text{PoE}_{iupdated} = \{&\text{BE}_C : [2,2,0,0,1,1], \\ &\text{CSP}_0 : [0,1,1,2,0,2], \\ &\text{CSP}_1 : [1,0,2,1,2,0]\}\end{aligned}$$

$$\begin{aligned}\text{PoE}_{iupdatedsorted} = \{&\text{BE}_C : [0,0,1,1,2,2], \\ &\text{CSP}_0 : [0,0,1,1,2,2], \\ &\text{CSP}_1 : [0,0,1,1,2,2]\}(7)\end{aligned}$$

$$\text{AND} = \{\text{AND}_0 : [\text{CSP}_0, \text{CSP}_1, \text{BE}_C]\}(7)$$

Analysis:

(1) Spare component with position zero missing presents in the MSCQs group.
(2) No pattern is identified in the RP analysis, implying that the spare component is shared.
(3) According to the PoE, the earliest failures of spare component occur in 1st, 2nd, 9th, and 10th MCSQs of the group.
(4) In these MCSQs, the events at position zero are BE_A and BE_B, so these are identified as the primary events of the shared spare component.
(5) The spare component is therefore associated with each primary event through multiple CSP gates.
(6) The MCSQ group is then updated by removing the primary events and the spare event involved in these CSP gates, following the spare-claiming logic.
(7) All remaining events are exchangeable and thus are connected by an AND relationship.

8. Conclusions and Future Works

Constructing dynamic fault trees (DFTs) remains a significant challenge in reliability engineering. Unlike static fault trees, which can often be directly inferred from component structures and logical relationships, DFTs require precise modeling of temporal dependencies, event sequences, and dynamic interactions among system elements. This task becomes especially challenging as system complexity increases, leading to a combinatorial explosion in possible failure sequences and making manual construction time-consuming and highly prone to errors.

To address these difficulties, this article introduces a data-driven approach to automatically construct DFTs. Central to the proposed framework is the analysis of extracted Minimal Cut Sequences (MCSQs) from observed failure time datasets. By formalizing the concepts of Positions of Events and Relative Positions, these methods enable qualitative analysis of logical relationships between failure events. Quantitative methods such as the PC causal discovery algorithm and linear causal dependency modeling are also included in the framework for the detection of causal dependencies.

Our future work will focus on developing systematic validation methods to assess the fidelity of the learned DFT. One proposed strategy is to generate MCSQs from the learned DFT and qualitatively compare them with the original MCSQs obtained from the dataset. This round-trip validation would evaluate whether the learned tree accurately reproduces the temporal orderings, dependencies, and logical relationships encoded in the data. Metrics such as sequence coverage, precision, and structural similarity can provide meaningful indicators of the algorithm's representational accuracy and guide further refinement of the learning process. Beyond this, developing a method to detect FDEP relationships involving multiple triggering events is also an essential extension of the current framework.

References

Arpit, D., M. Fernandez, I. Feigenbaum, W. Yao, C. Liu, W. Yang, P. Josel, S. Heinecke, E. Hu, H. Wang, et al. (2023). Salesforce causalai library: A fast and scalable framework for causal analysis of time series and tabular data. *arXiv preprint arXiv:2301.10859*.

Boros, E., P. L. Hammer, T. Ibaraki, A. Kogan, E. Mayoraz, and I. Muchnik (2000). An implementation of logical analysis of data. *IEEE Transactions on knowledge and Data Engineering 12*(2).

Dugan, J. B., S. J. Bavuso, and M. A. Boyd (1990). Fault trees and sequence dependencies. In *Annual Proceedings on Reliability and Maintainability Symposium*, pp. 286–293. IEEE.

Gascard, E. and Z. Simeu-Abazi (2018). Quantitative analysis of dynamic fault trees by means of monte carlo simulations: Event-driven simulation approach. *Reliability Engineering & System Safety 180*, 487–504.

Jimenez-Roa, L. A., T. Heskes, T. Tinga, and M. Stoelinga (2022). Automatic inference of fault tree models via multi-objective evolutionary algorithms. *IEEE transactions on dependable and secure computing 20*(4), 3317–3327.

Lim, S. S. Y. and E. Gascard (2025). Fault tree construction: A survey of knowledge-based, model-based and data-driven approaches. In *Proceedings of the 35th European Safety and Reliability & the 33rd Society for Risk Analysis Europe Conference*.

Linard, A., D. Bucur, and M. Stoelinga (2019). Fault trees from data: Efficient learning with an evolutionary algorithm. In *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*. Springer.

Linard, A., M. L. P. Bueno, D. Bucur, and M. Stoelinga (2019). Induction of fault trees through bayesian networks. In *European Safety and Reliability Conference Conference (ESREL)*.

Molak, A. (2023). *Causal Inference and Discovery in Python: Unlock the secrets of modern causal machine learning with DoWhy, EconML, PyTorch and more*. Packt Publishing Ltd.

Mukherjee, S. and A. Chakraborty (2007). Automated fault tree generation: bridging reliability with text mining. In *Annual Reliability and Maintainability Symposium*. IEEE.

Nauta, M., D. Bucur, and M. Stoelinga (2018). Lift: learning fault trees from observational data. In *International Conference QEST*. Springer.

Niloofar, P. and S. Lazarova-Molnar (2023). Data-driven extraction and analysis of repairable fault trees from time series data. *Expert Systems with Applications 215*.

Spirtes, P., C. N. Glymour, and R. Scheines (2000). *Causation, prediction, and search*. MIT press.

Tang, Z. and J. B. Dugan (2004). Minimal cut set/sequence generation for dynamic fault trees. In *Annual Symposium Reliability and Maintainability, 2004-RAMS*, pp. 207–213. IEEE.

Vesely, W. E., F. F. Goldberg, N. H. Roberts, and D. F. Haasl (1981). Fault tree handbook. Technical report.

Waghen, K. and M.-S. Ouali (2021). Multi-level interpretable logic tree analysis: A data-driven approach for hierarchical causality analysis. *Expert Systems with Applications 178*.