

# LIFT: Learning Fault Trees from Observational Data

Meike Nauta, Doina Bucur, and Mariëlle Stoelinga

University of Twente, Enschede, The Netherlands  
{m.nauta, d.bucur, m.i.a.stoelinga}@utwente.nl

**Abstract.** Industries with safety-critical systems increasingly collect data on events occurring at the level of system components, thus capturing instances of system failure or malfunction. With data availability, it becomes possible to automatically learn a model describing the failure modes of the system, i.e., how the states of individual components combine to cause a system failure. We present LIFT, a *machine learning method* for *static fault trees* directly out of *observational datasets*. The fault trees model probabilistic causal chains of events ending in a global system failure. Our method makes use of the Mantel-Haenszel statistical test to narrow down possible causal relationships between events. We evaluate LIFT with synthetic case studies, show how its performance varies with the quality of the data, and discuss practical variants of LIFT.

## 1 Introduction

Fault tree (FT) analysis [1] is a widely applied method to analyse the safety of high-tech systems, such as self-driving cars, drones and robots. FTs model how system failures occur as a result of component failures: the leaves of the tree model different failure modes, while the fault tree gates model how failure modes propagate through the system and lead to system failures. A wide number of metrics, such as the system reliability and availability, can then be computed to evaluate whether a system meets its dependability and safety requirements.

A key bottleneck is the construction of the FT. This requires domain knowledge, and the number of potential *failure causes* and contributing factors can be overwhelming: age, system loads, usage patterns and environmental conditions can all influence the failure mechanisms. It is thus appealing to learn FTs automatically from data, to assist reliability engineers in tackling the complexity of today’s systems. This paper is a first step in this direction: we *learn* static FTs from *observational records*.

**The fault-tree formalism.** The nodes in an FT are either events or logical gates. Fig. 1 shows an example FT and the graphical notation. A part of the system is modelled by an *intermediate event*; a special intermediate event is the root node of the tree, called the *top event* or *outcome*, which models the global system failure. A set of *basic events*, distinct from the intermediate events, marks the most elementary faults in system components, may be annotated with a probability of occurrence, and form the leaves of the FT. Intermediate

events form the inputs and the output of any gate, and are the output of any basic event. The basic gates, AND and OR, denoted by the standard logic-gate symbols, model their standard logic meaning, in terms of causal relationships between the events in the input and the event in the output of any gate.

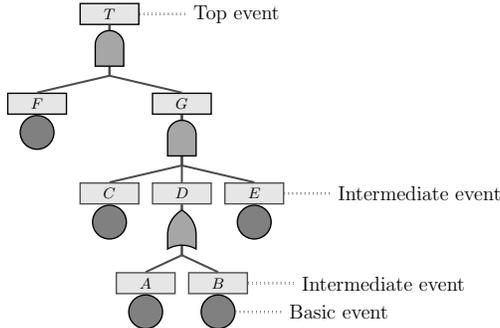


Fig. 1: Example fault tree with annotations

**Summary of contribution.** We learn static FTs with Boolean event variables (where an event variable has value True or 1 if that fault occurs in the system), n-ary AND/OR gates, and annotated with event failure probabilities. The input to the algorithm consists of raw, untimed observational data over the system under study, i.e., a dataset where each row is a single *observation* over the entire system, and each column *variable* records the value of a system *event*. All intermediate events to be included in the FT must be present in the dataset, but not all of those events in the dataset may be needed in the FT. We do not know what the basic events will be, nor which gates form the FT, nor which intermediate events are attached to the gates. We know the top event: the system failure of interest. Our main result is an algorithm that learns a statistically significant FT; we allow for a user-specified amount of noise, assumed uniformly distributed in the data. We evaluate the algorithm on synthetic data: given a “ground truth” FT, we synthesise a random dataset, apply the learning algorithm, and then compare the machine-learnt FT to the ground truth.

An example dataset is shown in Fig. 2a, in compact form: each row is a *count* (e.g., 20) of identical *records*, where each record is an untimed list of Boolean observations of events (denoted  $A, B, C$  and  $T$ , with  $T$  the global system failure, or outcome). The order of the records in a dataset is not significant.

A tree formalism commonly machine-learnt from such observational data is the Binary Decision Tree (BDT), a practical tool for the description, classification and generalisation of data [2]. The BDT learning algorithm appears to be a natural starting point for the design of an FT learning algorithm; however, we argue below why the BDT learning logic is unsatisfactory.

**Detecting causality from data.** The construction of a BDT is a greedy algorithm: a series of local optimum decisions determines subsequent branching nodes. Each decision uses a variable test condition (e.g., a classification error) to find the best split in the data records [3]. For example, when creating a BDT

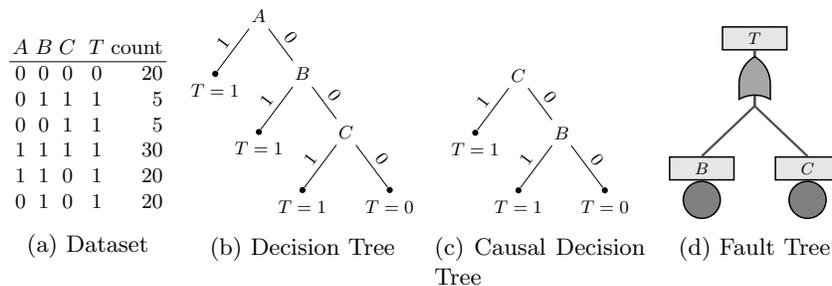


Fig. 2: An example showing that a BDT does not encode causal relationships.

for the dataset in Fig. 2a to classify variable  $T$ , a naive approach is to first split the dataset on variable  $A$ , and obtain the BDT in Fig. 2b. However, decision trees model *correlations* (which are symmetric) and not the *causal* relationships (which are asymmetric) required for an FT. As a correlation between variables does not imply a causation, the knowledge represented in a decision tree does not provide root causes for faults, and thus cannot support decision making.

To overcome this problem, the *Causal* Decision Tree (CDT) [4] was recently introduced. A CDT differs from a BDT in that each of its non-leaf nodes has a causal interpretation with respect to the outcome  $T$ . CDTs find causal relationships automatically by using the Mantel-Haenszel statistical test [5]. A causal relationship between a variable and the outcome exists if the causal effect is statistically significant (i.e., is above random chance). A CDT for the dataset in Fig. 2a is in Fig. 2c; it shows that  $C$  has a causal relationship with  $T$  and that  $B$  has a causal relationship with  $T$  under the “context” (i.e., fixed variable assignment)  $C = 0$ .  $A$  is not included in this CDT. The path  $(A = 1) \rightarrow (T = 1)$  in the BDT with probability  $P(T = 1|A = 1) = 1$  correctly classifies half of the records in the dataset. However, the path does not code a causal relationship between  $A$  and  $T$  since, for example, given  $C = 1$ ,  $P(T = 1|A = 1) - P(T = 1|A = 0) = 0$ . When fixing the value of  $C$ , a change in  $A$  does not result in a change in  $T$ . In fact,  $C$  causes  $T$ , and  $B$  causes  $T$  under the context  $C = 0$ .

CDTs on average achieve similar classification accuracy as decision trees, even though this is not a CDT objective; also, the size of CDTs is on average half that of decision trees [4], simplifying their analysis. Some aspects of CDT learning are useful in the automatic construction of an FT. However, while a CDT can only model the causal relationship between a variable and the outcome, the strength of an FT is the additional modelling of (a) multiple independent variables that may cause a failure, and (b) if-then Boolean logic. As shown in Fig. 2d, the CDT of Fig. 2c can be redrawn as an FT with a single OR gate.

In the following, Sect. 2 gives the related work on the automated synthesis of fault trees. Sect. 3 formally introduces FTs. Sect. 4 presents the LIFT algorithm and examples. Sect. 5 evaluates LIFT on datasets with noise or superfluous variables. Sect. 6 discusses possible LIFT variants. The conclusions, including future work, are presented in Sect. 7.

## 2 Related Work

System dependability evaluation via fault tree analysis is largely a manual process, performed over informal fault-tree models which will not accurately describe an evolving system [6]. Due to this, the *automatic synthesis of fault trees* has been of recent interest. However, we stress the fact that most of the existing contributions generate the necessary dependability information from existing, formal system models, and are thus Model-Based Dependability Analysis (MBDA) techniques [6–8]. In contrast, there is little research aiming to synthesise causal dependability information for *black-box systems*, for which formal models do not exist, or for which the quantity and quality of the available sensed data surpasses the quality and completeness of existing system models.

**Learning fault trees from data.** Observational data was used for machine-learning fault trees in the Induction of Fault Trees (IFT) algorithm [9], based on decision-tree learning. As in our method, all that is needed are observations of measurable quantities taking certain values. However, IFT completely disregards the matter of causality between events, and essentially learns a syntactically correct FT which encodes exactly the same information as a decision tree – so the FT is essentially a classifier, rather than a means of modelling causal effect.

**Generating fault trees from formal system models.** A diverse body of techniques is available for this; we refer to recent reviews on MBDA for a complete picture [6–8] and give here a brief overview of the most relevant generation methods. While these approaches cannot directly synthesise FTs from observational data (as in our work), other techniques able to learn the required system models from observational data could (indirectly) bridge this gap.

In the Hierarchically Performed Hazard Origin & Propagation Studies (HiP-HOPS) framework [10], any system model formalising the transactions among the system components, annotated with failure information for components (as Boolean expressions), may be used to synthesise an FT. Using these annotations, the synthesis is straightforward: it proceeds top-down from the top event and creates local FTs based on the component failure annotations; these are then merged into a global FT showing all combinations leading to system failure. If formal models in the AltaRica high-level system description language are available, they include explicit transitions modelling causal relations between state variables and events, which can similarly be used to synthesise classic FTs [11]. The Formal Safety Analysis Platform (FSAP/NuSMV-SA) generates, from NuSMV system models, FTs which show only the relation between top events and basic events, and not how faults propagate among the system components [12]. The Architecture Analysis and Design Language (AADL) includes an Error Model for the specification of fault information, and a number of techniques exist to translate an AADL model into static or dynamic FTs (recently, in [13]). AADL models have also been translated into models compatible with the HiP-HOPS and AltaRica frameworks, enabling cross-framework FT synthesis [6].

A process of FT generation with explicit reasoning about causality is described in [14]; however, this approach still requires a formal system model to exist. Given such a probabilistic system model, a set of probabilistic counterex-

amples (i.e., system execution paths of temporally ordered, interleaved events leading to a system fault) is obtained from the process of model-checking. As the system is concurrent, the counterexamples potentially, but not necessarily, model causality. Logical combinations of events are determined as causes of other events using a set of test conditions; the time complexity is cubic in the size of the set of counterexamples.

**Other approaches.** Causal Bayesian Networks (CBNs) [15] can also be learnt from observational data, as well as Boolean formulas (BFs) [16]; both models may be translated into FTs, and both learning problems are NP-hard or require exponential time [17, 16]. As our algorithm will also be shown to have a worst-case exponential complexity, both CBNs and BFs remain feasible alternatives to FT learning.

### 3 Background: Fault Trees

We define the basic components of an FT formally in Definitions 1–4.

**Definition 1.** A *gate*  $G$  is a tuple  $\langle t, \mathbf{I}, O \rangle$ , where:

- $t$  is the type of  $G$ , with  $t \in \{And, Or\}$ .
- $\mathbf{I}$  is a set of  $n \geq 2$  intermediate events  $\{i_1, \dots, i_n\}$  that are inputs to  $G$ .
- $O$  is the intermediate event that is output for  $G$ .

We denote by  $I(G)$  the set of intermediate events in the input of  $G$ , and by  $O(G)$  the intermediate event in the output of  $G$ .

**Definition 2.** An **AND gate** is a gate  $\langle And, \mathbf{I}, O \rangle$  where output  $O$  occurs (i.e.  $O$  is True) if and only if every  $i \in \mathbf{I}$  occurs.

**Definition 3.** An **OR gate** is a gate  $\langle Or, \mathbf{I}, O \rangle$  where output  $O$  occurs (i.e.  $O$  is True) if and only if at least one  $i \in \mathbf{I}$  occurs.

**Definition 4.** A **basic event**  $B$  is an event with no input and one intermediate event as output. We denote by  $O(B)$  the intermediate event in the output of  $B$ .

Intuitively, a basic event  $B$  models an elementary system fault in the real world; its output  $O(B)$  is True when this elementary system fault occurs. Then, all system components modelled by the events in the input of an AND gate must fail in order for the system modelled by the event in the output to fail.

We then formalise the fault tree in Definition 5.

**Definition 5.** A **fault tree**  $\mathbf{F}$  is a tuple  $\langle \mathbf{BE}, \mathbf{IE}, T, \mathbf{G} \rangle$ , where:

- $\mathbf{BE}$  is the set of basic events;  $\forall B \in \mathbf{BE}, O(B) \in \mathbf{IE}$ . A basic event may be annotated with a probability of occurrence  $p$ .
- $\mathbf{IE}$  is the set of intermediate events, where  $\mathbf{IE} \cap \mathbf{BE} = \emptyset$ .
- $T$  is the top event,  $T \in \mathbf{IE}$ .
- $\mathbf{G}$  is the set of gates;  $\forall G \in \mathbf{G}, I(G) \subset \mathbf{IE}, O(G) \in \mathbf{IE}$ .

- The graph formed by  $\mathbf{G}$  should be connected and acyclic, with the top event  $T$  as unique root.

Given fault tree  $\mathbf{F}$ , we denote by  $IE(\mathbf{F})$  the set of intermediate events in  $\mathbf{F}$ .

The basic LIFT algorithm (Sect. 4) will learn trees rather than directed acyclic graphs (DAGs), i.e. an intermediate event can be the input of only one gate. Sect. 6 will then discuss a DAG variant of the LIFT algorithm.

**Comparison FT-CDT.** Unlike FTs, CDTs can be learnt from data, and also encode causal relationships between variables; an example CDT was given in Fig. 2c. However, there are major *syntactic* differences between the two formalisms. An FT can be n-ary, while a CDT can only be binary: every branching decision is based on a Boolean variable. Also, an FT is more concise: it models only the positive (failure) outcome, while the CDT must model both outcomes of any variable. Finally, the position of the outcome differs: while in FTs the top event models the system outcome, in a CDT this is modelled by leaf nodes.

## 4 Machine Learning Fault Trees

The dataset from which an FT can be learnt contains untimed, Boolean observations of system events; an FT *event* corresponds to a column *variable* in the dataset. A record and a dataset are formally defined in Definitions 6–7.

**Definition 6.** A record  $R$  over the set of variables  $\mathbf{V}$  is a list of length  $|\mathbf{V}|$  containing tuples  $[(V_i, v_i)]$ ,  $1 \leq i \leq |\mathbf{V}|$ , where:

- $V_i$  is a variable name,  $V_i \in \mathbf{V}$ .
- $v_i$  is a Boolean value of  $V_i$ .

**Definition 7.** A dataset  $\mathbf{D}$  is a set of  $r$  records, all over the same set of variables  $\mathbf{V}$ . Each variable name in  $\mathbf{V}$  forms a column in  $\mathbf{D}$  and each record forms a row. When  $k$  identical records are present in  $\mathbf{D}$ , a single such record is shown, with a new count column for the value  $k$ .

A synthetic dataset (of 185 records in total, but only 11 unique records) is shown in Table 1. We assume the *sufficiency* of any dataset (i.e., all shared causes are measured [18]) and also its *faithfulness* (i.e., the data accurately represents the real-world dependencies [18]). However, because of either sensor glitches or human error, there may be some noise in the dataset (i.e., flipped bits).

From a dataset, causal relationships between (groups of) variables can be discovered to form an FT. For this, one can use the standard Mantel-Haenszel Partial Association test (PAMH) [5], a test used for the analysis of data that is *stratified*. When stratifying the dataset, the effect of other variables on the outcome variable  $T$  is eliminated, and hence the difference reflects the causal effect of one variable (say,  $E$ ) on the outcome  $T$ . By this test, a causal relationship between two given variables is statistically significant if and only if the PAMH-score  $\geq \tilde{\chi}_{\alpha,1}^2$ , where  $\tilde{\chi}_{\alpha,1}^2$  is the standard critical value of the chi-square

Table 1: Example dataset

<i>A</i>	<i>B</i>	<i>C</i>	<i>D</i>	<i>E</i>	<i>F</i>	<i>G</i>	<i>T</i>	count
0	0	0	0	0	0	0	0	30
1	0	1	1	0	1	0	0	20
0	1	0	1	1	0	0	0	20
0	1	0	1	1	1	0	0	20
1	0	0	1	0	0	0	0	15
0	0	1	0	0	1	0	0	15
0	0	0	0	1	0	0	0	15
0	0	1	0	1	1	0	0	15
1	1	1	1	1	0	1	0	20
0	1	1	1	1	1	1	1	10
1	0	1	1	1	1	1	1	5

distribution with 1 degree of freedom [19]. A significance level of  $\alpha = 0.05$  or  $\alpha = 0.01$  is often used in practice.

A stratum is formally defined in Definition 8 (and is a classic concept, as per [19]). A concrete example is given later in this section, in Example 1.

**Definition 8.** *Given a dataset  $\mathbf{D}$  over the set of variables  $\mathbf{V}$ , a **stratum**  $\mathbf{S}_{E,T}$  (where  $E$  and  $T$  are variables from  $\mathbf{V}$ ) is a contingency table which shows the distribution of the values of variables  $E$  and  $T$  in the dataset, as counts, in the following format:*

	$T=1$	$T=0$	Total	
$E = 1$	$n_{11}$	$n_{12}$	$n_{1.}$	where $n$ denotes the number of records that satisfy a given valuation of $E$ and $T$ .
$E = 0$	$n_{21}$	$n_{22}$	$n_{2.}$	
Total	$n_{.1}$	$n_{.2}$	$n_{..}$	

The PAMH-score can be calculated over multiple strata (as done in CDTs [4] and first formalised in [19]); here we have a single stratum, as follows:

$$\text{PAMH}(E, T) = \left( \left| \frac{n_{11}n_{22} - n_{21}n_{12}}{n_{..}} \right| - \frac{1}{2} \right)^2 \bigg/ \frac{n_{1.}n_{2.}n_{.1}n_{.2}}{n_{..}^2(n_{..} - 1)}$$

Using these concepts of strata and the PAMH-score, the LIFT algorithm<sup>1</sup>, shown in Alg. 1, synthesises an FT from a dataset  $\mathbf{D}$ . A variable in  $\mathbf{D}$  corresponds to an intermediate event. All intermediate events to be included in the FT must be present in the dataset, but not all of those events in the dataset may be needed in the FT. We do not know what the basic events will be, nor which gates form the FT, nor which intermediate events are attached to the gates.

<sup>1</sup> Code can be found at <https://github.com/M-Nauta/LIFT>

---

**Algorithm 1: LIFT: Learning a Fault Tree from a dataset**


---

**Input:**  $\mathbf{D}$ , a data set containing  $r$  records over  $\mathbf{V}$ ;  
 $T$ , the intended top event with  $T \in \mathbf{V}$ ;  
 $\alpha$ , the significance level for the Mantel-Haenszel test

**Result:** Fault Tree  $\mathbf{F}$

```

1 Function CheckANDGate( $E, \mathbf{I}$ ):
2    $result = \text{False}$ ,  $pamh = 0.0$ 
3    $\mathbf{v} = [v_1, \dots, v_r]$  in which  $v_j$  is 1 if every  $i \in \mathbf{I}$  in record  $j$  of  $\mathbf{D}$  is 1
4   if  $n_{12}$  of  $\mathbf{S}_{\mathbf{v}, E} < \alpha n..$  &&  $n_{21}$  of  $\mathbf{S}_{\mathbf{v}, E} < \alpha n..$  then
5      $pamh = PAMH(\mathbf{v}, T)$ 
6     if  $pamh \geq \tilde{\chi}_{\alpha, 1}^2$  then
7        $result = \text{True}$ 
8     return  $result, pamh$ 

9 Function CheckORGate( $E, \mathbf{I}$ ):
10  | Similar to lines 2-8

11 Function CreateLevel( $\mathbf{F}, \text{Leaves}$ ):
12  | for  $l \in \text{Leaves}$  do
13  |    $k = 2$ ,  $gate = \text{False}$ 
14  |   while not  $gate$  and  $k \leq |\mathbf{V} \setminus IE(\mathbf{F})|$  do
15  |     for  $\mathbf{a}$  in generator of combinations of size  $k$  from  $\mathbf{V} \setminus IE(\mathbf{F})$  do
16  |       | compute  $isGate, pamh = \text{CheckANDGate}(l, \mathbf{a})$ 
17  |       | compute  $isGate, pamh = \text{CheckORGate}(l, \mathbf{a})$ 
18  |       if at least one  $\mathbf{a}$  exists where  $isGate$  was  $\text{True}$  then
19  |         | select that  $\mathbf{a}$  and gate type  $t$  where  $pamh$  was maximum
20  |         | add gate  $\langle t, \mathbf{a}, l \rangle$  to  $\mathbf{F}$ 
21  |         |  $gate = \text{True}$ 
22  |       else
23  |         |  $k++$ 
24  |       if not  $gate$  then
25  |         |  $p =$  ratio of records in  $\mathbf{D}$  where  $l = 1$ 
26  |         | create basic event  $B$  as input for  $l$  in  $\mathbf{F}$ , and annotate  $B$  with  $p$ 
27  |       return  $\mathbf{F}$ 

28 let  $\mathbf{F} =$  Fault Tree  $\langle \emptyset, \{T\}, T, \emptyset \rangle$ 
29 while at least one new event is added to  $\mathbf{F}$  do
30   | let  $\text{Leaves} =$  set of all intermediate events at the lowest level of  $\mathbf{F}$ 
31   |  $\mathbf{F} = \text{CreateLevel}(\mathbf{F}, \text{Leaves})$ 

```

---

*Checking a proposed gate.* To create a fault tree  $\mathbf{F}$ , the LIFT algorithm iteratively adds a level to  $\mathbf{F}$ , starting with just the top event (line 28-31 in Alg. 1). Each time `CreateLevel` is called, the depth of the FT increases with one level. For each intermediate event  $E$  at the lowest level of  $\mathbf{F}$ , sets (of size  $\geq 2$ ) containing intermediate events not yet in  $\mathbf{F}$  are proposed as input of a new (AND or OR) gate whose output is  $E$ . This is done by checking the gate for correctness according to the properties of Definitions 2–3. If both gates are correct, any design choice can be made, as discussed later in Sect. 6.

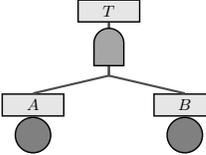
*Example 1.* Using the data set shown in Table 2a, a significance level  $\alpha$ , the outcome variable  $T$  and the set of variables  $\mathbf{I} = \{A, B\}$ , one can check if gate  $G$  of the form  $\langle \text{And}, \mathbf{I}, T \rangle$  meets the property specified in Definition 2 and is statistically significant using function `CheckANDgate`.

Table 2: Dataset for Example 1, over variables  $A$ ,  $B$  and outcome variable  $T$ ; stratum and fault tree learnt.

$A$	$B$	$T$	count
0	0	0	30
1	0	0	25
0	1	0	20
0	1	1	1
1	1	1	15

$A$	$B$	$\mathbf{v}$	$T$	count
0	0	0	0	30
1	0	0	0	25
0	1	0	0	20
0	1	0	1	1
1	1	1	1	15

	$T=1$	$T=0$	Total
$\mathbf{v} = 1$	15	0	15
$\mathbf{v} = 0$	1	75	76
Total	16	75	91



(a) Dataset

(b) Dataset incl.  
var.  $\mathbf{v}$  (AND gate)

(c) Stratum  $\mathbf{S}_{\mathbf{v},T}$

(d) FT learnt

A temporary new variable  $\mathbf{v}$  is added to the dataset (Table 2b).  $\mathbf{v}$  encodes an AND relation between  $A$  and  $B$ ;  $\mathbf{v}$  occurs (is 1) only when both  $A$  and  $B$  occur. This variable  $\mathbf{v}$  can then be compared with top event  $T$  to measure if there is a causal relationship between  $T$  and  $A$  AND  $B$ . The stratum  $\mathbf{S}_{\mathbf{v},T}$  is computed by counting the corresponding records in Table 2b, as shown in Table 2c.

The user can specify the ratio of noise allowed by LIFT per stratum. (If the user can assume that the flipped bits are uniformly distributed in the dataset, the expected per-stratum noise ratio is equal to the global noise ratio.) For simplicity, we set this noise “allowance” equal to the significance level  $\alpha$ ; the algorithm is easily modified for any other level. In the dataset shown in Table 2a, one can see that one record may be noise. In this example, we will set  $\alpha = 0.05$ , so we allow 5% noise in a stratum. It then follows that the proposed AND gate  $G$  of the form  $\langle \text{And}, \{A, B\}, T \rangle$  meets the property of Definition 2 because in stratum  $\mathbf{S}_{\mathbf{v},T}$  we have  $n_{12} = 0$ ,  $n_{21} = 1$ , meaning one record where the values of  $\mathbf{v}$  and  $T$  differ. We do allow a ratio  $\alpha$  out of  $n_{..} = 91$  to differ, but  $1 < 0.05 \cdot 91$  holds. However, if we would have selected a significance level  $\alpha = 0.01$ , since  $1 < 0.01 \cdot 91$  does not hold, the FT couldn’t include this gate.

If the proposed gate has less noise than allowed (which is in this case true for  $\alpha = 0.05$ ), we can determine if the causal relation between  $T$  and  $\mathbf{v}$  is significant, by calculating the PAMH-score:

$$\text{PAMH}(\mathbf{v}, T) = \left( \frac{15 \cdot 75 - 1 \cdot 0}{91} - \frac{1}{2} \right)^2 \bigg/ \frac{15 \cdot 76 \cdot 16 \cdot 75}{91^2(91 - 1)} = 76.66 .$$

For  $\alpha = 0.05$ , the critical value  $\tilde{\chi}_{\alpha,1}^2 = 3.84$ . Since the PAMH-score is higher than  $\tilde{\chi}_{\alpha,1}^2$ ,  $\mathbf{v}$  and  $T$  can be concluded to have a significant causal relationship.

Similarly, a proposed OR gate is checked. By creating a temporary new variable  $\mathbf{v}$  for the OR gate, one can create a stratum to calculate the noise and the PAMH-score in a similar way. This OR gate will have too much noise and is

therefore not correct. So,  $\langle \text{And}, \mathbf{I}, T \rangle$  is added to  $\mathbf{F}$ . Table 2d shows the final FT learnt from the original dataset in Table 2 for  $\alpha = 0.05$ .

An FT may have a path containing two subsequent gates of the same type; in this case, the FT solution is not unique, and one may optimise for either *minimal gate sizes*, or *minimal tree depth*. We choose here the former, i.e., select the smallest input sets for all gates. LIFT is easily modified for another aim. Example 2 below clarifies this situation.

*Example 2.* Take the dataset in Table 1 at the beginning of this section. The LIFT algorithm starts with an FT containing only the top event  $T$  (line 28 in Alg. 1). For this top event, the algorithm generates all combinations (sets) of intermediate events, in order of increasing size (line 15). For each set  $\mathbf{a}$  containing intermediate events, LIFT tests whether a gate  $\langle \text{Or/And}, \mathbf{a}, T \rangle$  (either AND or OR) meets the property in Definitions 2–3 and does not exceed the noise allowance (line 4). If true, LIFT checks if the PAMH score is higher than the threshold for the Mantel-Haenszel test.

For this dataset, there is no correct OR gate  $\langle \text{Or}, \mathbf{a}, T \rangle$ . However, there are 9 sets of intermediate events that can act as input for a correct and significant AND gate  $\langle \text{And}, \mathbf{a}, T \rangle$ :  $\mathbf{a} = \{F, G\}$ ,  $\mathbf{a} = \{E, F, G\}$ ,  $\mathbf{a} = \{D, F, G\}$ ,  $\mathbf{a} = \{C, F, G\}$ ,  $\mathbf{a} = \{D, E, F, G\}$ ,  $\mathbf{a} = \{C, E, F, G\}$ ,  $\mathbf{a} = \{C, D, F, G\}$ ,  $\mathbf{a} = \{C, D, E, F\}$  and  $\mathbf{a} = \{C, D, E, F, G\}$ . In other words, there are multiple structural solutions for the FT, when the FT has a path with two subsequent gates of the same type. In such cases, LIFT learns the solution with minimum-sized gates. The input sets are generated in increasing size, and LIFT will stop proposing input sets when the minimum correct input set is found. In this example, the smallest set has size two, namely  $\{F, G\}$ . Therefore, gate  $\langle \text{And}, \{F, G\}, T \rangle$  is added to  $\mathbf{F}$  (as shown in Fig. 3). In case of multiple correct sets of the same size (which can arise when there are more variables in the dataset than needed in the FT) the set with the highest PAMH-score is selected (line 19 in Alg. 1). The algorithm can be easily modified for another design decision, as argued later in Sect. 6.

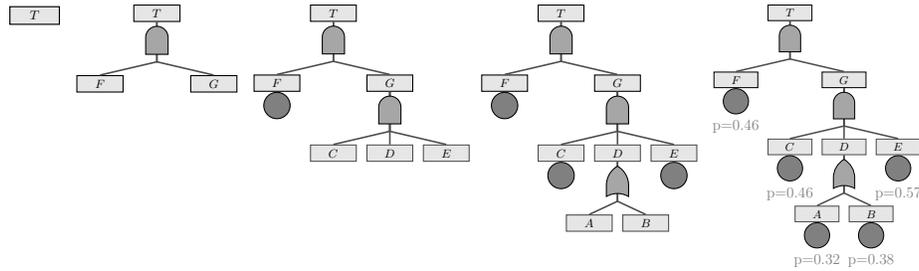


Fig. 3: Applying LIFT in Example 2 on the dataset shown in Table 1.

In the next iteration, for both  $F$  and  $G$  again sets of intermediate events are tried. The search space is now  $2^5 - 5 - 1 = 26$ , because  $|\mathbf{V} \setminus IE(\mathbf{F})| = 5$ . There is no correct gate  $\langle \text{Or/And}, \mathbf{a}, F \rangle$  for intermediate event  $F$ . Therefore, a basic

event is added as input for  $F$  (line 26). For intermediate event  $G$ , one correct and significant AND gate  $\langle \text{And}, \{C, D, E\}, G \rangle$  is found and added to  $\mathbf{F}$ . Similar iterations are done for  $C$ ,  $D$  and  $E$  followed by  $A$  and  $B$ , as shown in Fig. 3.

When the dataset contains information on system states which are always measured in a fixed time horizon (i.e. *discrete* time), one can easily derive stochastic measures such as failure probabilities using standard probability laws. The statistical probability that an event  $E \in \mathbf{D}$  occurs is simply  $P(E = 1) = \frac{\# \text{ records where } E=1}{\text{total } \# \text{ records}}$ ; all basic events are annotated with these probabilities.

## 5 Evaluation

The algorithm is evaluated following the approach shown in Fig. 4. A number of fault trees  $\mathbf{F}_{gt}$  are generated as ground truth; from each of these FTs, a dataset is synthesised randomly, including adding noise and superfluous variables (both of these processes of synthesis are described below). LIFT takes this dataset and a given significance value  $\alpha$  as input, and learns another FT  $\mathbf{F}$ , which can then be compared to the ground truth. We say that a learnt FT is “correct” if it is *structurally equivalent* to the ground-truth FT, i.e., syntactically (and not only semantically) equivalent, where only the order of the inputs to any gate may differ. We require that the learnt FT recovers the *exact gates* as in the ground truth, since these gates may model concrete system components, for which the correct causes of failure should be learnt. Our evaluation is thus stronger than an isomorphism check for the FTs.

Furthermore, we assess how noise and superfluous variables in the dataset influence the ratio of correctly learnt FTs.

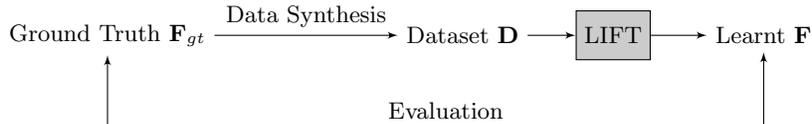


Fig. 4: Evaluation approach: Randomly generate a dataset from an FT, apply LIFT to that dataset and compare the learnt FT with the ground truth.

*Generating all FTs of a certain size* As ground truth, we generate *all possible* FTs over a fixed number (here, 8) of intermediate events, with no probabilities annotated on basic events. We only generate trees and leave DAGs (with shared variables) as future work. To mimic a manually constructed FT where readability is important, we set a minimum of 2 intermediate events and a maximum of 5 intermediate events as input to a gate, and thus obtain 76 different FTs.

*Generating a synthetic dataset from an FT* Based on a generated FT, we mimic a real-life situation by randomly synthesising 1000-record datasets where basic events happen with a certain probability (and are not rare events). The generation process starts with valuating all basic events to either 0 or 1, with a

randomly chosen probability between 20% and 50% that each basic event is 1. These values are propagated through the gates in the FT up to the top event; dependent on the type of each gate, the gate’s output event is assigned 0 or 1. Each iteration of this procedure results in one data record.

To assure that gates are correctly recognised and that every gate is at least once true, every combination of inputs for a gate occurs in at least  $c\%$  of the rows in the dataset (i.e. in the case of 1000 rows and  $c=2$ , every combination occurs at least 20 times). We created datasets for both  $c = 0.5\%$  and  $c = 2\%$ . We leave the task of discriminating between rare events and noise for future work.

*Adding noise to the dataset* In a real-life situation, having perfectly clean data is rare because of wrong measurements, sensor glitches or manual errors for example. To mimic noise, a number up to 5% of the rows in the dataset are added, each with 1-2 wrong (flipped) values.

*Adding superfluous variables to the dataset* Our algorithm should also create a correct fault tree when there are variables in the dataset which have *no causal effect* and should not be included in the learnt FT. We thus experiment with adding up to 4 non-causal system variables. The case of a causal superfluous variable is discussed in Sect. 6.

## 5.1 Results

An analysis is done on the influence of noise or superfluous variables in the dataset on the number of correct fault trees obtained by LIFT.

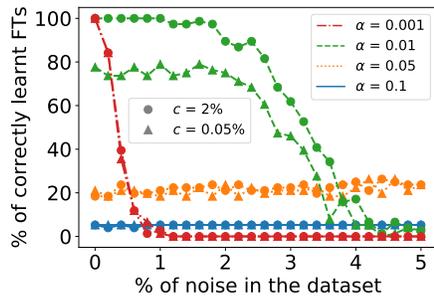


Fig. 5: Percentage of correctly learnt fault trees relative to the percentage of rows with noise in the dataset. All 76 different FTs with 8 intermediate events are generated. The dataset for each FT contains no non-causal variables, and 1000 records plus an extra percentage of noisy records.

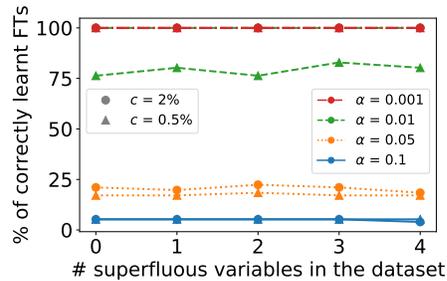


Fig. 6: Percentage of correctly learnt FTs relative to the number of non-causal variables in the dataset. All 76 different FTs with 8 intermediate events are generated. The dataset for each FT contains 1000 records, no noisy records, but up to 4 non-causal variables.

As we generated all ground-truth FTs with exactly 8 intermediate events, and 2-5 inputs for each gate (76 FTs in total), the basic datasets contain 8 columns (one for each intermediate event), and 1000 rows. Noisy rows are then added to the dataset, depending on the level of noise desired.

Figure 5 shows the percentage of correctly learnt fault trees relative to the percentage of rows with noise in the dataset. All learnt FTs are correct in the absence of noise and with the significance level  $\alpha = 0.001$ . However, this  $\alpha$  is by nature incapable of correctly dealing with noise, since LIFT may not find a significant gate due to the noise. A higher  $\alpha$  is less sensitive for noise, but does result in a lower number of correct FTs. A learnt FT may be incorrect when LIFT finds a significant gate with a smaller number of inputs than what should actually be the case. Therefore, the significance level should be chosen based on the amount of noise in the dataset. Furthermore, one can see that  $c$  naturally influences the number of correctly learnt FTs: the less rare the events are in the dataset, the more likely is LIFT to learn the correct FT.

Figure 6 shows the percentage of correctly learnt FTs relative to the number of non-causal random variables present in the dataset; one can see that these variables have little effect on the accuracy, showing that LIFT indeed finds only causal relationships.

## 5.2 Complexity

**Time complexity** LIFT exhaustively checks all input event combinations in order of their size, so in worst case there is one gate with all variables (except the top event) as input. This means that for all input sets, a stratum is created that loops over all  $r$  records and over  $k$  variables that are in that set. The number of different combinations of size  $k$  is  $\binom{n}{k}$  where  $n = |\mathbf{V}| - 1$ . Therefore, the time complexity of these operations is  $r \cdot \sum_{k=2}^n k \cdot \binom{n}{k}$ . The PAMH-score of each stratum is calculated and compared with the significance level, which has a constant time complexity. This results in a time complexity of  $O(nr2^n)$ .

Learning boolean formulae, closely related to learning static fault trees, from examples obtained by querying an oracle is exponential in the size of the vocabulary in the general case as well as for many restrictions [16]. More precisely, a static fault tree with only AND and OR gates can be seen as a monotone boolean function for which the Vapnik-Chervonenkis (VC) dimension is exponential in  $n$  [20]. So, a general *exact* FT learning algorithm cannot be more efficient than the VC dimension. Reaching better complexity, which could be useful for large datasets, is then only possible when an approximated FT is learnt, instead of an exact solution. Such a variant of Alg. 1 may apply a *greedy search-and-score* approach rather than our constraint-based approach with exhaustive search, as inspired by structure-learning algorithms for Bayesian networks. However, those algorithms may suffer from getting stuck in a local maximum, resulting in a lower reconstruction accuracy. Furthermore, the highest-scoring network structure is not necessarily the only viable hypothesis [21].

**Space complexity** The input for Algorithm 1 consists of dataset  $\mathbf{D}$  with  $r$  records and  $n$  columns, top event  $T$  and significance level  $\alpha$ . Therefore, the input space complexity is  $\Theta(rn)$ . If the generator of combinations is on-the-fly, its auxiliary memory complexity is  $O(n^2)$ .

## 6 Discussion

**Interpretation of causality** Currently, all intermediate events that should be in the fault tree have to be included in the dataset. However, obtaining a dataset containing all relevant variables may be impractical. One problem is the presence of hidden variables that influence measured variables but are not in the dataset themselves [18]. The other one is the selection bias: values of unmeasured variables may influence whether a unit is included in the dataset [21]. This can result in a learnt causal relationship between observed variables that does not correspond to the real causal relations. Drawing valid causal inferences using observational data is therefore not just a mechanistic procedure, but always depends on assumptions and justification that require domain knowledge [22]. We are aware of the critical assessment of causal claims based on observational data, but we think the learnt fault tree will still be valuable to give insights which possibly were unknown beforehand and facilitates further causal inference.

**Algorithm variants** We made certain design decisions for the basic LIFT algorithm in Alg. 1. Below, we present some of the many possible variants.

*Multiple gate types* In the case of multiple significant correct gates with the same number of inputs, the LIFT algorithm chooses the one with the highest PAMH-score. However, there may be cases where both an OR gate and an AND gate are correct. For example, in case of the dataset as shown in Table 3, an OR gate will be created when a very high significance level is chosen. However, two of these records may be noise, so with a lower significance level an AND gate will result in a correct gate as well. Selecting the gate type is then a matter of choice: one can argue to choose the OR gate as this matches exactly the dataset, or choose the AND gate since the interpretation of this gate is stricter than the OR gate. One can also argue that the algorithm need not make a decision at all and that it outputs multiple FTs. LIFT is easily modified for any design decision.

Table 3: Dataset where both an OR gate and an AND gate may be correct.

$A$	$B$	$T$	count
0	0	0	1
1	0	1	1
0	1	1	1
1	1	1	10,000

*Multiple significant FTs* When there are *causal* superfluous variables in the dataset, there may be cases of multiple correct sets of intermediate events of the same size, that can all serve as input to a statistically significant gate. While the basic LIFT algorithm chooses the input set with the highest PAMH-score, it is easily modified for a different design choice, such as returning all correct FTs.

*The FT as a Directed Acyclic Graph (DAG)* The basic LIFT algorithm learns trees, so the examples and evaluation presented in this paper all learn tree structures. However, in general FTs may share subtrees, meaning that an intermediate event can be the input of multiple gates, and therefore have a directed acyclic structure [1]. The LIFT algorithm can be modified to create DAGs by generating broader combinations  $\mathbf{a}$  of intermediate events, outside the while loop at line 14 of Alg. 1: instead of a generator of all combinations of size  $\geq 2$  from  $\mathbf{V} \setminus IE(\mathbf{F})$ , one can instead have a generator of all combinations from  $\mathbf{V} \setminus T$ , with an extra check that the created graph  $\mathbf{F}$  remains acyclic.

*More efficient exploration of variable combinations* Other features of the dataset (e.g., the graph of dependencies between variables), or even domain knowledge, may be used to reduce the number of combinations of variables to be tried by LIFT as inputs to gates.

## 7 Conclusion

In this paper, we presented an algorithm to automatically learn a statistically significant fault tree from Boolean observational data, inspired by the construction algorithm for Causal Decision Trees. In absence of noise, all learnt FTs were found to be structurally equivalent to the ground truth when the significance level is 0.001. With up to 3% noise in the data, a significance level of 0.01 results in around 65% correct FTs. As a downside, the basic LIFT algorithm does an exhaustive search, and thus has exponential time complexity. It also cannot deal with hidden variables.

In future work, the algorithm can be extended to learn other elements of a fault tree, such as the XOR gate (true if and only if exactly one of its input events is true). Note that elements that need sequence information (such as the Priority-AND gate or the SPARE gate) cannot be implemented, since the required dataset format doesn't contain timing information. Learning fault trees from *timed observational data* is also a direction for future work. For this, learning Bayesian networks, closely related to FTs, may also be a competitive direction to take. Moreover, one may allow continuous data instead of only binary values, similar to the C4.5 algorithm for decision trees [23] that creates a binary expression for continuous values. This expression encodes the conditions under which a measurement results in a failure.

**Acknowledgements** This research was supported by the Dutch STW project SEQUOIA (grant 15474). The authors would like to thank Joost-Pieter Katoen and Djoerd Hiemstra for valuable feedback.

## References

1. Ruijters, E., Stoelinga, M.: Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review* **15** (2015) 29–62
2. Murthy, S.K.: Automatic construction of decision trees from data: A multi-disciplinary survey. *Data Mining and Knowledge Discovery* **2**(4) (1998) 345–389
3. Tan, P., Steinbach, M., Kumar, V.: *Introduction To Data Mining*. Pearson Education (2006)
4. Li, J., Ma, S., Le, T., Liu, L., Liu, J.: Causal decision trees. *IEEE Transactions on Knowledge and Data Engineering* **29**(2) (2017) 257–271
5. Mantel, N., Haenszel, W.: Statistical aspects of the analysis of data from retrospective studies of disease. *J. of the national cancer institute* **22**(4) (1959) 719–748
6. Kabir, S.: An overview of fault tree analysis and its application in model based dependability analysis. *Expert Systems with Applications* **77** (2017) 114–135
7. Aizpurua, J.I., Muxika, E.: Model-based design of dependable systems: Limitations and evolution of analysis and verification approaches. *International Journal on Advances in Security Volume 6, Number 1 & 2, 2013* (2013)
8. Sharvia, S., Kabir, S., Walker, M., Papadopoulos, Y.: Model-based dependability analysis: State-of-the-art, challenges, and future outlook. In: *Software Quality Assurance*. Elsevier (2016) 251–278
9. Madden, M.G., Nolan, P.J.: Generation of fault trees from simulated incipient fault case data. *WIT Trans. Information and Communication Technologies* **6** (1994)
10. Papadopoulos, Y., McDermid, J.: *Safety-directed system monitoring using safety cases*. PhD thesis, University of York (2000)
11. Li, S., Li, X.: Study on generation of fault trees from Altarica models. *Procedia Engineering* **80** (2014) 140–152
12. Bozzano, M., Villaflorita, A.: The FSAP/NuSMV-SA safety analysis platform. *International Journal on Software Tools for Technology Transfer* **9**(1) (2007) 5
13. Li, Y., Zhu, Y.a., Ma, C.y., Xu, M.: A method for constructing fault trees from AADL models. In: *Int. Conf. on Autonomic and Trusted Computing*, Springer (2011) 243–258
14. Leitner-Fischer, F., Leue, S.: Probabilistic fault tree synthesis using causality computation. *Int. J. Critical Computer-Based Systems* **30** **4**(2) (2013) 119–143
15. Li, J., Shi, J.: Knowledge discovery from observational data for process control using causal Bayesian networks. *IIE transactions* **39**(6) (2007) 681–690
16. Jha, S., Raman, V., Pinto, A., Sahai, T., Francis, M.: On learning sparse boolean formulae for explaining AI decisions. In: *NASA Formal Methods Symposium*, Springer (2017) 99–114
17. Chickering, D.M., Heckerman, D., Meek, C.: Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research* **5** (2004) 1287–1330
18. Kleinberg, S.: *Why: A Guide to Finding and Using Causes*. O’Reilly (2015)
19. Birch, M.: The detection of partial association, i: the  $2 \times 2$  case. *Journal of the Royal Statistical Society. Series B (Methodological)* (1964) 313–324
20. Kearns, M., Li, M., Valiant, L.: Learning boolean formulas. *Journal of the ACM (JACM)* **41**(6) (1994) 1298–1328
21. Koller, D., Friedman, N.: *Probabilistic graphical models: principles and techniques*. MIT press (2009)
22. Rohrer, J.M.: *Thinking clearly about correlations and causation: Graphical causal models for observational data*. (2017)
23. Quinlan, J.R.: *C4. 5: programs for machine learning*. Elsevier (2014)