# Induction of Fault Trees through Bayesian Networks

Alexis Linard[1], Marcos L. P. Bueno[1,3], Doina Bucur[2] and Marielle Stoelinga[1,2]

[1]*Institute for Computing and Information Science. Radboud University, Nijmegen, The Netherlands.*
*E-mail: {a.linard,m.bueno}@cs.ru.nl*
[2]*University of Twente, Enschede, The Netherlands.*
*E-mail: {d.bucur,m.i.a.stoelinga}@utwente.nl*
[3]*Department of Computer Science, Federal University of Uberlândia, Brazil*

Cyber-physical systems have increasingly intricate architectures and failure modes, which is due to an explosion of their complexity, size, and failure criticality. While expert knowledge of individual components exists, their interaction is complex. For these reasons, obtaining accurate system reliability models is a hard task. At the same time, systems tend to be continuously monitored via advanced sensor systems. This data describes the components' failure behavior and can be exploited for failure diagnosis and learning of reliability models. This paper presents an effective algorithm for the learning of Fault Trees from data. Fault trees (FTs) are a widespread formalism in reliability engineering. They capture the failure behavior of components and their propagation through an entire system. To that end, we first use machine learning to compute a Bayesian Network (BN) highlighting probabilistic relationships between the failures of components and root causes. Then, we apply a set of rules to translate a BN into an FT, based on the Conditional Probability Tables to decide, amongst others, the nature of gates in the FT. We evaluate our method on synthetic data and a benchmark set of FTs.

*Keywords*: fault tree induction, safety-critical systems, cyber-physical systems, machine learning, Bayesian network inference, risk analysis, failure diagnosis.

## 1. Introduction

Reliability engineering provides methods, tools and techniques to evaluate and mitigate the risks related to complex systems. Fault tree analysis is one of the most prominent technique in this field and is widely deployed in the automotive, aerospace and nuclear industry.

Fault trees (FTs) Vesely et al. (1981) are deterministic graphical models that represent how component failures arise and propagate through the system, leading to system-level failures. Component failures are modelled in the leaves of the tree as *basic events*. FT *gates* model how combinations of basic events lead to a system failure, represented by the top event in the FT. The analysis of such FTs (Ruijters and Stoelinga, 2015) is multifold: they can be used to compute dependability metrics such as system reliability and availability; understand how systems can fail; identify the best ways to reduce the risk of system failure, etc.

A key bottleneck in FT analysis is, however, the effort needed to construct a faithful FT model. FTs are usually built manually by domain experts. Given the complexity of today's systems, industrial FTs often contain thousands of gates. Hence, their construction is a very difficult task, and also error-prone, since their soundness and completeness largely depends on domain expertise.

With the emergence of the industrial Internet-of-Things, Cyber-physical systems are more and more equipped with smart sensor systems, monitoring whether or not a system component is in a failed state or not. Their data can, therefore, be very fruitfully exploited to learn reliability models. Such data can be crucial for the engineers to build an FT (Joshi et al., 2007). Recent work focused on learning FTs from observational data, identifying causalities from data (Nauta et al., 2018). In this paper, we focus on FT generation from data, using the prior identification of Bayesian Networks (BNs).

Bayesian Networks (Pearl, 1988) are probabilistic graphical models widely used in industry and healthcare as a diagnosis tool. BNs are also used in dependability analysis of systems, being an interesting formalism for the generation of qualitative and diagnosis measurements. In this case, given an FT representing a system's failure behavior, a set of rules are applied to transform this FT into a BN, to later generate these metrics (Bobbio et al., 2001).

This methodology caught our attention: considering a case where data is available, but the FT structure is unavailable mostly due to lack of expert knowledge, the reverse operation seems needed. We considered the case where observational data is available, and where the basic and

*Proceedings of the 29th European Safety and Reliability Conference.*
*Edited by* Michael Beer and Enrico Zio

*Published by* Research Publishing, Singapore.
ISBN: 978-981-11-2724-3; doi:10.3850/978-981-11-2724-3_0596-cd

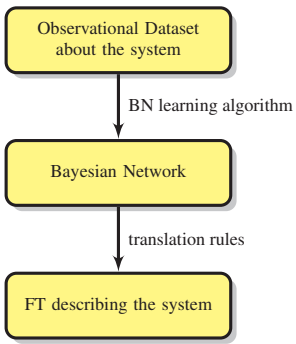



Fig. 1.: FT learning framework.

intermediate events, i.e. observed variables, are also known. Learning the structure as well as the parameters of a BN can be optimally performed from data (Cooper, 1990). Given the learned BN, we apply a set of translation rules to translate any BN node (a random variable) into an FT gate (a deterministic OR or AND gate), using the probabilistic information from the local conditional probability tables inferred from data. Our framework is depicted in Fig. 1.

We applied our algorithm, both on synthetic data and a benchmark of FTs previously studied in the literature. Our experiments show that our method is efficient and accurate ($> 99\%$). Being a first step, our algorithm focuses on *static* FTs, featuring only Boolean gates. An important topic for future work is the extension to dynamic FTs (Dugan et al., 1990). Therefore, we will discuss an extension to learning of dynamic BNs, than can be in turn translated into dynamic FTs.

This paper is organized as follows. Section 2 reviews related work on learning FTs from data. In Section 3 we recall preliminary definitions on FTs and BNs. We present then in Section 4 our technique to infer an FT using BN translation. In Section 5 we show the results we achieved. Finally, we discuss and conclude about future work.

## 2. Related Work

Related work on learning FTs spans several areas of research: recent work on the generation of FTs from observational data describing the system; since FTs are in essence Boolean functions, literature on learning Boolean functions from observational data; and, finally, relations between BNs and FTs.

**Learning Boolean formulas and classifiers from data.** Observational data was used to generate FTs with the IFT algorithm (Madden and Nolan, 1994) based on standard decision-tree statistical learning. The advantage of learning a graphical decision tree out of data is the inherent interpretability of decision-tree models and their ease of translation into other graphical models.

Boolean formulas or networks are learnt using a similar tree-based method (Kearns et al., 1994). The classic C4.5 learning algorithm yields a Boolean decision tree that is easily translatable into a Boolean formula, by constructing the conjunction of all paths leading to a leaf modelling a True value (i.e., system failure), and then simplifying the Boolean function (by either computing its conjunctive or disjunctive normal form). The resulting models encode the same information as a decision tree (i.e., a classifier for the observational data), so they lack the validation of causal relations, but are expected to preserve their predictive power about the system. Furthermore, Boolean formulas were also machine-learnt using black-box classifiers (classifiers not easily interpretable as a graphical model). Such methods include Support-Vector Machines, Logistic Regression and Naive Bayes.

**Learning causal models from data.** LIFT (Nauta et al., 2018) is a recent approach for learning static FTs from data, with Boolean event variables, n-ary *And*/*Or* gates, annotated with event failure probabilities. All intermediate events to be included in the FT must be present in the dataset, but not all may be needed in the FT. LIFT also includes a causal validation step to filter for the most likely causal relationships among system events, but the worst-case complexity is exponential in the number of system events in the data. Its main advantage is that of being one of the few automated FT-learning methods which specifically validate causality. However, LIFT is sensitive to noise, which is a drawback when dealing with real-world data.

Bayesian Networks (Li and Shi, 2007) are standard graphical models which can be learnt from data. They are widely used in industry, as well as in the health domain as a diagnosis tool. These models have straightforward translations into FTs (Bobbio et al., 1999, 2001), which are used in FT analysis in order to compute metrics such as reliability or expected number of failures. It has been shown that BNs are hard to synthesize accurately (Chickering et al., 2004). However, assumptions provided by domain experts can reduce the search space for uncovering these models from data.

While previous study (Khakzad et al., 2011) favour BNs due to their more general formalism, both in terms of uncertainty as in nominality of data variables, we stand up for FTs, due to their intuitive formalism preferred in industry. We propose a novel method to learn an FT from a tabular dataset composed of observational tuples, in which values (failures) for each Boolean basic event, intermediate event and top event in the system are known. We compare it with a subset of these existing learning algorithm, in terms of performance when fitting data.

## 3. Background

We first define the structure of FTs, consisting of logic gates and Boolean fault events (an event has value True if that fault occurs in the system). *Intermediate events* (IEs) in the FT are logical combinations of other events (i.e. the OR or AND of a set of events), with only *basic events* (BEs) as the leaves of the tree, and one special intermediate event called the *top event* as root. We also formally define an observational dataset from which we learn an FT. The *top event* of the FT must be a variable in the dataset and can be seen as the outcome to predict by the FT. We finally describe Bayesian Networks, which are graphical models representing probabilistic dependencies between variables. The formulations below follow definitions from Scutari (2010) and Nauta et al. (2018).

### 3.1. *Fault Trees*

Fault trees model how component failures propagate into system failures. Subtrees can be shared, so FTs can be directed acyclic graphs (DAGs). The leaves of the tree model component failures and are called *basic events*. Two types of gates (*And* and *Or*) model how BE failures lead to system failures. The graphical notation for gates and basic events is given in Fig. 2.

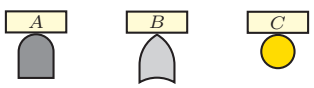



Fig. 2.: Graphical notation for *And* ($A$) and *Or* ($B$) gates, and for a basic event ($C$).

**Definition 3.1.** A **gate** $G$ is a tuple $(t, \mathbf{I}, O)$ s.t.:

- $t$ is the type of $G$ with $t \in \{And,\ Or\}$.
- $\mathbf{I}$ is a set of $n \geq 2$ input events of $G$.
- $O$ is the event that is the output of $G$.

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

**Definition 3.2.** An *And* gate is a gate $(And, \mathbf{I}, O)$ where output $O$ occurs if and only if every $i \in \mathbf{I}$ occurs. An *Or* gate is a gate $(Or, \mathbf{I}, O)$ where output $O$ occurs if and only if at least one $i \in \mathbf{I}$ occurs.

**Definition 3.3.** A **basic event** $B$ is an event with no input and with a failure rate denoted $p_B$.

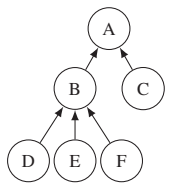
Other gates are also used: the XOR (exclusive OR), voting, and NOT gates (Vesely et al., 1981; Lee et al., 1985). The root of the tree is the top event $T$, which represents the failure condition of interest, such as the stranding of a train, or the unplanned unavailability of a satellite. An FT fails if its top event fails.

**Definition 3.4.** A **fault tree F** is a tuple $(\mathbf{BE}, \mathbf{IE}, T, \mathbf{G})$ where:

- **BE** is the set of basic events.
- **IE** is the set of intermediate events.
- $T$ is the top event, $T \in \mathbf{IE}$.
- **G** is the set of gates; $\forall G \in \mathbf{G}, I(G) \subset \mathbf{IE} \cup \mathbf{BE}, O(G) \in \mathbf{IE}$.
- The graph formed by **G** should be connected and acyclic, with the top event $T$ as unique root.

### 3.2. *Bayesian Networks*

Bayesian networks (BNs) are graphical models where nodes represent random variables and arrows represent probabilistic dependencies between them. In the remaining of this paper, we consider the case of binary *discrete* BNs where variables are Boolean.




(a) Example BN.

| | | $P(A \mid B, C)$ | |
|---|---|---|---|
| | | $A = 0$ | $A = 1$ |
| $B = 0$ | $C = 0$ | 1 | 0 |
| $B = 0$ | $C = 1$ | 0 | 1 |
| $B = 1$ | $C = 0$ | 0 | 1 |
| $B = 1$ | $C = 1$ | 0 | 1 |

(b) Example CPT for variable A.

| | | | $P(B \mid D, E, F)$ | |
|---|---|---|---|---|
| | | | $B = 0$ | $B = 1$ |
| $D = 0$ | $E = 0$ | $F = 0$ | 1 | 0 |
| $D = 0$ | $E = 0$ | $F = 1$ | 1 | 0 |
| $D = 0$ | $E = 1$ | $F = 0$ | 1 | 0 |
| $D = 0$ | $E = 1$ | $F = 1$ | 1 | 0 |
| $D = 1$ | $E = 0$ | $F = 0$ | 1 | 0 |
| $D = 1$ | $E = 0$ | $F = 1$ | 1 | 0 |
| $D = 1$ | $E = 1$ | $F = 0$ | 1 | 0 |
| $D = 1$ | $E = 1$ | $F = 1$ | 0 | 1 |

(c) Example CPT for variable B.

Fig. 3.: Example BN and related CPTs.

The graphical structure $G = (\mathbf{V}, A)$ of a Bayesian network is a *directed acyclic graph* (DAG), where $\mathbf{V} = \mathbf{BE} \cup \mathbf{IE}$ is the set of *nodes* and $A$ is the set of *edges*. We say that if a node has a *parent* node, then the node is marginally dependent (or unconditionally dependent) of the parent node. A node is said to be a *root* node when it has no parent node. Each node in the BN structure is associated with a Conditional Probability Table (CPT). The joint distribution of the BN can be factorized according to its graphical structure, which leads to:

$$P(V_1, \ldots, V_n) = \prod_{i=1}^{n} P(V_i \mid \Pi_{V_i}) \qquad (1)$$

where $\Pi_{V_i}$ denotes the set of parents of the node $V_i$ in the graph $G$. In such a case, both the joint and the local distributions (i.e. the CPTs) are multinomial.

An example BN and related CPTs is shown in Fig. 3. Note here that these CPTs are a case of *certainty*, where the conditional probabilities are

set to either 0 or 1 (unlike CPTs with uncertainty, where probabilities range from 0 to 1).

### 3.3. *Learning Dataset*

We define a dataset as a collection of records. Each record is a valuation for the set of BEs and IEs, indicating whether a *failure* was observed for that BE/IE. Further, we assume that our dataset is *labeled*, i.e., also indicates whether the top event $T$ has failed, yielding the outcome of the FT.

**Definition 3.5.** A record $R$ over the set of variables **BE** $\cup$ **IE** is a list containing tuples $[(V_i, v_i)], \forall V_i \in$ **BE** $\cup$ **IE**, where $v_i$ is a Boolean value of $V_i$.

We call a *noisy* record one where the value of at least one variable has been measured incorrectly.

**Definition 3.6.** A dataset **D** is a set of $r$ records, all over the same set of variables **BE** $\cup$ **IE**. Each variable name in **BE** $\cup$ **IE** forms a column in **D**, and each record forms a row.

Table 1 shows an example dataset for the BN from Fig. 3a.

| A | B | C | D | E | F |
|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 1 |
| 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 |

Table 1.: Example dataset for BN in Fig. 3a.

## 4. Learning Fault Trees from Diagnosis Models

Learning of BNs is two-fold: the first task is to perform *structure learning*, where the goal is to learn the *skeleton* of the BN together with the directions of the dependencies. The second task is *parameter learning*, to approximate CPTs from data (e.g. via maximum likelihood estimation). Structure learning is typically done by heuristic search (using, e.g. the BIC scoring function and a TABU search). It provides approximate Bayesian network structures in a reasonable time, as finding optimal structure is intractable. It is important to note here that involving domain knowledge can reduce the complexity of structure learning: if one knows what dependencies have to be present (or not) in the structure, this given information reduces the search space of the structure.

In this section, we show how we perform efficient structure and parameter learning of BNs from data. We also show how we achieve the translation of a BN into an FT thanks to a set of translations rules.

### 4.1. *Learning of Bayesian Networks*

**Structure Learning** BN structure learning algorithms can be grouped in two categories: The first one, *constraint-based algorithms* (Bonissone et al., 1991), which learn structure causal models. They can be summarized in two steps: (1) learn the *skeleton* (i.e. the undirected graph underlying the network structure) of the network; (2) set the direction of all the edges. The second category, *score-based algorithms*, assign a score to each candidate BN and try to maximize it with some heuristic search algorithm. Greedy search algorithms (such as *hill-climbing* or *tabu search*) are common.

**Blacklisting** Prior assumptions on the data or the problem to solve can be integrated into the learning algorithm through *blacklists* and *whitelists*. They define arcs which are respectively missing or present in the BN. In our setting, we defined three families of dependencies which are not desired in the BN structure. These undesired arcs represent conditional dependencies which are assumed not to be relevant in the BN. Our blacklisted arcs are the followings:

- $T \rightarrow V_i, \forall V_i \in$ **BE** $\cup$ **IE**. However, it is allowed that $V_i \rightarrow T$ because this captures the natural cause-effect direction of the system being modeled.
- $B_i \rightarrow B_j, \forall B_i, B_j \in$ **BE**. Two any BEs are not involved in direct probabilistic interaction among themselves (however, this constraint can be relaxed, see Section 6).
- $IE \rightarrow B, \forall IE \in$ **IE**, $\forall B \in$ **BE**. However, it is allowed that $IE \rightarrow B$ because this captures the natural cause-effect direction of the system being modeled. This constraint is particularly necessary for a good identification of which variables will play the role of gates once the BN is translated into an FT.

The blacklisted arcs of BN in Fig. 3a are then as follows:

- $A \rightarrow B$
- $A \rightarrow C$
- $A \rightarrow D$
- $A \rightarrow E$
- $A \rightarrow F$
- $C \rightarrow D$
- $C \rightarrow E$
- $C \rightarrow F$
- $D \rightarrow C$
- $D \rightarrow E$
- $D \rightarrow F$
- $E \rightarrow C$
- $E \rightarrow D$
- $E \rightarrow F$
- $F \rightarrow C$
- $F \rightarrow D$
- $F \rightarrow E$
- $B \rightarrow C$
- $B \rightarrow D$
- $B \rightarrow E$
- $B \rightarrow F$

**Parameter Learning** The goal of parameter learning is to approximate, given data on random variables, the CPTs of a given BN structure. After

learning the parameters via maximum likelihood estimation, the BIC (Bayesian Information Criterion) score of the BN is computed. If $M$ is a candidate BN, then its BIC is given by:

$$\mathrm{BIC}(M) = -2\ln(L) + K\ln(N) \qquad (2)$$

where $L$ is the maximized log-likelihood of $M$, $K$ is the number of parameters of $M$, and $N$ is the sample size (i.e. the size of the dataset). Models with smaller BIC are preferred. It measures not only the fitness of the model to the data, but also has the advantage of controlling for the model complexity.

### 4.2. *Translation Rules*

In Bobbio et al. (1999, 2001), a two-fold method to translate an FT into a BN was proposed. It consists of a one-to-one transformation of FT gates into event nodes in the BN, as well as the related CPT generation:

(1) Translate $T$ and all $V_i \in$ **BE** $\cup$ **IE** into BN nodes.
(2) Add probabilistic dependencies between related events (BEs or IEs).
(3) Set failure probabilities of BEs to follow definitions of $p_B$ for all $B \in$ **BE**.
(4) Set failure probabilities of IEs to follow the truth tables of related events.

In practice, the latter can be replaced based on expert knowledge. This is in order to relax conditions with certainty to CPTs with uncertainty.

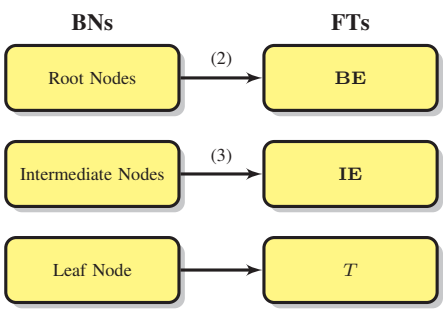



Fig. 4.: Translation of a BN into an FT.

In our case, a BN and the associated CPTs are already learned from data. As a consequence, we have to perform the reverse translation compared to Bobbio et al. (1999, 2001). Our translation is below, and also shown in Fig. 4:

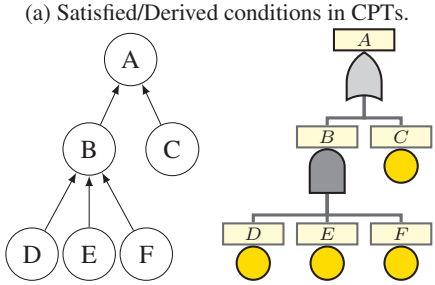
(1) Remove all nodes which are not connected to the system level failure node in the BN.
(2) For each *root* node of the BN, create a BE in the FT (each root node in the BN is necessarily a basic event, following blacklisting constraints). For each created BE $B$, set $p_B$ to $P(B = 1)$.
(3) For each non-root node of the BN, create an IE in the FT.
(4) Connect events in the FT the way their related nodes in the BN are connected.
(5) For each gate in the FT, identify its nature (*Or* or *And*) by inspecting the CPTs.

The identification of the nature of a gate follows the CPT of its related node. For each node $V_i \in$ **IE**, we apply the following set of translation rules:

(1) if $P(V_i = 1 \mid \bigwedge_{c \in C} c = 1) = 1$ where $C$ is the set of parents of $V_i$, then $V_i$ is an *And* gate in the FT.
(2) $P(V_i = 1 \mid \bigvee_{c \in C} c = 1) = 1$ where $C$ is the set of parents of $V_i$, then $V_i$ is an *Or* gate in the FT.

$$P(A = 1 \mid B = 1 \vee C = 1) = 1$$
$$P(B = 1 \mid D = 1 \wedge E = 1 \wedge F = 1) = 1$$

(a) Satisfied/Derived conditions in CPTs.




(b) Example BN. (c) Induced Fault Tree.

Fig. 5.: Translation of BN of Fig. 3 into an FT.

**Uncertainty** One can see here that probabilistic non-root nodes of a BN are translated into deterministic gates in the FT. The certainty feature of the CPTs may be unrealistic in the real-world. One may prefer to assume that the logical gates are *noisy* (i.e. probabilistic) and return an FT gate $G$ with an associated probability $p_G$, as being the probability of failure of the IE when the logical condition is met. Therefore, we defined an alternative uncertain case for translating an FT:

(1) if $P(V_i = 1 \mid \bigwedge_{c \in C} c = 1) > 1 - \epsilon$ where $C$ is the set of parents of $V_i$, then $V_i$ is an *And* gate in the FT with $p_G = P(V_i = 1 \mid \bigwedge_{c \in C} c = 1)$
(2) $P(V_i = 1 \mid \bigvee_{c \in C} c = 1) > 1 - \epsilon$ where $C$ is the set of parents of $V_i$, then $V_i$ is an *Or* gate with $p_G = P(V_i = 1 \mid \bigvee_{c \in C} c = 1)$

where $\epsilon$ is a tolerance factor for deciding whether a gate is an *Or* gate or an *And* gate.

In case none of the conditions above hold, we would recommend to leave out the variable from the dataset, since it may not be relevant to the problem. Note here that, since subtrees can be shared in an FT, it may be the case that a BE or an IE is an input to more than one gate.

An example of translation of BN and CPTs in Fig. 3 into an FT is shown in Fig. 5.

## 5. Experimental Evaluation

We have evaluated the efficiency and effectiveness of our method for a set of synthetic cases, as well as for industrial benchmarks. We compared our methods with other learning techniques, among others, five approaches from the literature. The accuracy of an FT is the number of records in the dataset for which the value of the top event, given the values of the BEs, is correctly computed.

### 5.1. *Experimental set up*

The first three methods in our evaluation are: (1) Support Vector Machine (abbreviated svm in the figures), (2) Logistic Regression (log) and (3) Naive Bayes Classifier (nba). These methods are Boolean classifiers that, given the values of the BEs, predict the value of the top event $T$. During our experiments, we used state-of-the-art Python implementations for these techniques. Being classifiers, methods (1) - (3) do not yield fault tree models, only a prediction for the value of $T$. Then, we have used three methods that do learn FT models: (4) We have compared our results to the well-known C4.5 algorithm for learning decision trees (abbreviated c45). Decision trees can be transformed to FTs, by first computing in the decision tree the conjunction of all paths leading to failure leaves, and then simplifying the conjunction to CNF. (5) We have also compared to the earlier LIFT approach, which returns an FT.

To compare these methods, the observational data was divided into 2 sets: one training set, used as input to the learning algorithms (with an average of 2/3 of all possible observations and possibly duplicate records), and a test set containing all observational variables (complete truth table), used to evaluate the solution returned by our algorithms. We used the library *bnlearn* to learn BNs (Scutari, 2010). The parameters of our algorithm were a *tabu search* for learning the structure of the BN, and the maximization of the BIC score for learning the parameters. The blacklisting of edges was set as detailed in Section 4.1. We considered the uncertain case, with $\epsilon = 0.1$.

A further experiment was carried out (bn-o) where edges $V_i \rightarrow T$ for all parent node of $T$ in the BN are known and set as *whitelist*, i.e. the top event, and its direct input events in the FT are known. This is a real case in practice: indeed, experts may partially know the structure of the FT (hence of the BN) and like to include this information before the learning of the BN.

### 5.2. *Synthetic Dataset*

We first considered 100 randomly generated FTs with 6 to 15 BEs, and for each FT a randomly generated data set, with 200 to 230k records. The FTs considered contain on average 5 gates, and BEs probabilities range from 1e-4 to 1e-2. Figures 6 and 7 present respectively the average accuracy and the average runtime, both as functions of the number of BEs.

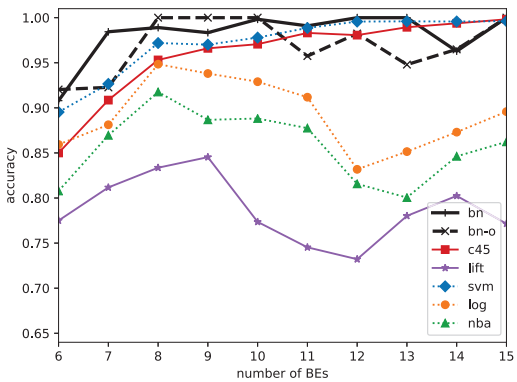



Fig. 6.: Accuracy as a function of BEs.

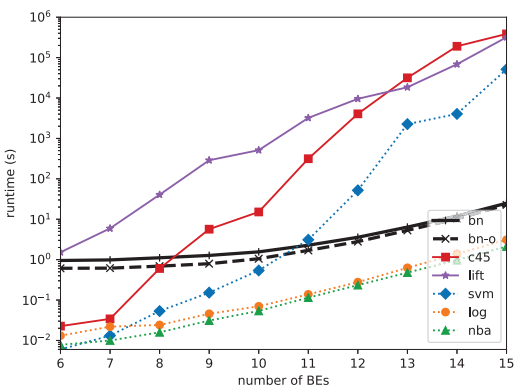



Fig. 7.: Runtime of different algorithms.

Fig. 6 shows that our methods bn and bn-o are optimal as well in terms of accuracy as runtime. When the number of BEs in the FT is high, svm and c45 are equally optimal. However, the svm method only provides a classifier, not an FT. Further, the c45 method does not perform well in terms of runtime, see Fig. 7. Methods log and nba are fast, but provide low accuracy. Finally, we see that LIFT obtains less good results. This can be explained by an exponential complexity, as well as the fact that LIFT return FTs with a low depth in most of the cases or even missing many BEs.

We also carried out an experiment where noise is added in the dataset, in order to test the robustness of the different algorithms. The noise varies

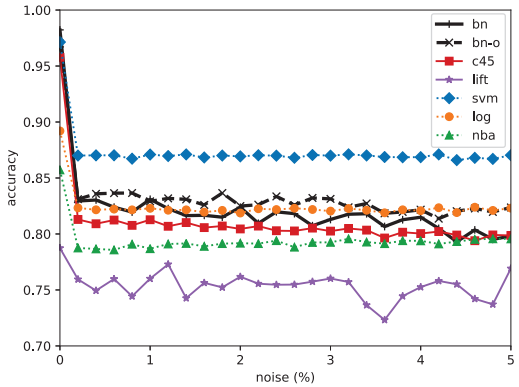



Fig. 8.: Effect of noise on the learned FTs.

from 0 to 5% of noisy records. In the results shown in Fig. 8, we see that our methods are relatively robust against noise compared to other methods. However, we see that the accuracy of the learned FTs drops whenever noise is present in the dataset.

### 5.3. *Fault Tree Benchmark*

We present here the results we obtained for a set of publicly available benchmark suite[a,b], consisting of industrial FTs from the literature (Boudali and Dugan, 2005; Junges et al., 2017; Montani et al., 2006; Stamatelatos et al., 2002). The FTs used are: Cardiac Assist System (CAS), Container Seal Design Example (CSD), Multiprocessor Computing System (MCS), Monopropellant Propulsion System (MPS), Pressure Tank (PT), Sensor Filter Network (SF14) and Spread Mooring System (SMS_A1).

| | # gates | # BEs | T prob | BEs probs |
|---|---|---|---|---|
| CAS | 8 | 9 | 3e-6 | [1e-6, 5e-6] |
| CSD | 4 | 6 | 1e-6 | [1e-5, 1e-1] |
| MCS | 10 | 11 | 6e-6 | [2e-6, 8e-2] |
| MPS | 9 | 9 | 2e-5 | [5e-5, 2e-2] |
| PT | 5 | 6 | 4e-5 | [5e-6, 1e-4] |
| SF14 | 8 | 9 | 3e-3 | [2e-3, 3e-3] |
| SMS_A1 | 7 | 10 | 3e-2 | [6e-8, 9e-3] |

Table 2.: Statistics on datasets.

The structural composition of these FTs (stated in terms of number of gates, number of BEs, probability of the top event and probabilities of BEs) is shown in Table 2.

Whereas the FT models were given in the literature, no data sets were available. Therefore, we have randomly generated these data sets, containing 10M records per case (in order to handle low failure rates). Since the benchmark does provide failure probabilities per BEs, we have used those

[a] https://dftbenchmarks.utwente.nl/

[b] https://gitlab.science.ru.nl/alinard/learning-ft-bn

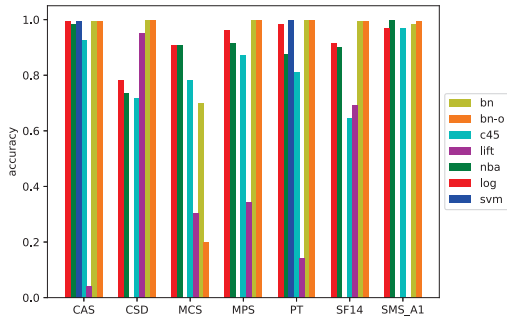



Fig. 9.: Results of FT Benchmark (accuracy).

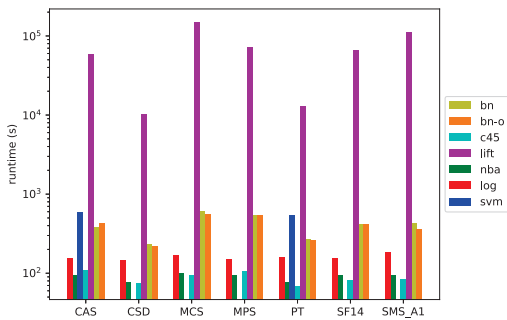



Fig. 10.: Results of FT Benchmark (runtime).

probabilities: If $p_B$ is the failure probability of BE $B$ in the benchmark, then we set, in each data record, $R[B] = 1$ with probability $p_B$.

## 6. Discussion

One of the limitations of our work is the need for data exhaustiveness: indeed, we require knowledge about all BEs, as well as all intermediate events. One can imagine a realistic case where some of the intermediate events are missing. However, uncovering these hidden variables is known to be hard in BN synthesis.

Another limitation is the focus on static FTs. However, one can see straightforward extensions that could use the formalism of dynamic FTs (DFTs) Dugan et al. (1990). These come with additional gates, catering for common dependability patterns like spare management and functional dependencies. Some additional translation rules lead to semantics missing in static FTs, but present in DFTs:

(1) *Functional Dependency* (FDEP): models how the occurrence of an event triggers the failure of other components. So if $P(V_i = 1 \mid \underset{c \in C}{\Longrightarrow} c = 1) = 1$ where $C$ is the set of parents of $V_i$, then $c \rightarrow V_i, \forall c \in C$ is a FDEP in the DFT.

Also, some constraints relaxation, e.g. in the blacklisting of edges, can lead to DFT gates:

(2) *Rate Dependency* (RDEP): models how a BE failure influences other BE failures. When the blacklisting of edges $B_i \rightarrow B_j, \forall B_i, B_j \in$ **BE** is relaxed, then edge $B_i \rightarrow B_j$ is a RDEP.

Another attractive formalism to take into account is the one of dynamic BNs: they relate variables to each other over time. It is known that dynamic BNs can model, among others, priority AND gates of dynamic FTs (Ruijters and Stoelinga, 2015). We leave the translation of dynamic BNs into dynamic FTs for future work.

## 7. Conclusion

We presented an efficient method for the automated generation of FTs from pre-computed BNs from data. To that extent, we defined a set of translation rules from one formalism to another. Our results show that our method is particularly robust to noise. We have also applied our technique to a benchmark of FTs to ensure its practical relevance. Our future research will focus on the learning of dynamic FTs. This will be done by the prior computation of a dynamic BN from time series, and the translation of such a dynamic BN to a dynamic FT with appropriate rules.

## Acknowledgement

This research is supported by the Dutch Technology Foundation (STW) under the Robust CPS program (project 12693), the EU project SUCCESS, as well as the Smart Industries program (project SEQUOIA 15474).

## References

Bobbio, A., L. Portinale, M. Minichino, and E. Ciancamerla (1999). Comparing fault trees and Bayesian networks for dependability analysis. In *International Conference on Computer Safety, Reliability, and Security*, pp. 310–322.

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

Bonissone, P., M. Henrion, L. Kanal, and J. Lemmer (1991). Equivalence and synthesis of causal models. *Uncertainty Artificial Intelligence*, 255–270.

Boudali, H. and J. B. Dugan (2005). A discrete-time Bayesian network reliability modeling and analysis framework. *Reliability Engineering & System Safety 87*(3), 337–349.

Chickering, D. M., D. Heckerman, and C. Meek (2004). Large-sample learning of Bayesian networks is NP-hard. *Journal of Machine Learning Research 5*, 1287–1330.

Cooper, G. F. (1990). The computational complexity of probabilistic inference using Bayesian belief networks. *Artificial intelligence 42*(2-3), 393–405.

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

Joshi, A., V. Gavriloiu, A. Barua, A. Garabedian, P. Sinha, and K. Khorasani (2007). Intelligent and learning-based approaches for health monitoring and fault diagnosis of radarsat-1 attitude control system. In *2007 IEEE International Conference on Systems, Man and Cybernetics*, pp. 3177–3183.

Junges, S., D. Guck, J.-P. Katoen, A. Rensink, and M. Stoelinga (2017). Fault trees on a diet: automated reduction by graph rewriting. *Formal Aspects of Computing 29*(4), 651–703.

Kearns, M., M. Li, and L. Valiant (1994). Learning boolean formulas. *ACM 41*(6), 1298–1328.

Khakzad, N., F. Khan, and P. Amyotte (2011). Safety analysis in process facilities: Comparison of fault tree and Bayesian network approaches. *Reliability Engineering & System Safety 96*(8), 925 – 932.

Lee, W.-S., D. L. Grosh, F. A. Tillman, and C. H. Lie (1985). Fault tree analysis, methods, and applications: A review. *IEEE transactions on reliability 34*(3), 194–203.

Li, J. and J. Shi (2007). Knowledge discovery from observational data for process control using causal Bayesian networks. *IIE transactions 39*(6), 681–690.

Madden, M. G. and P. J. Nolan (1994). Generation of fault trees from simulated incipient fault case data. *WIT Trans. Information and Communication Technologies 6*.

Montani, S., L. Portinale, A. Bobbio, M. Varesio, and D. Codetta-Raiteri (2006). A tool for automatically translating dynamic fault trees into dynamic Bayesian networks. In *RAMS'06. Annual Reliability and Maintainability Symposium, 2006.*, pp. 434–441. IEEE.

Nauta, M., D. Bucur, and M. Stoelinga (2018). LIFT: Learning fault trees from observational data. In *Quantitative Evaluation of Systems. QEST 2018.*, pp. 306–322.

Pearl, J. (1988). Probabilistic reasoning in intelligent systems: Networks of plausible inference.

Ruijters, E. and M. Stoelinga (2015). Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review 15-16*, 29 – 62.

Scutari, M. (2010). Learning Bayesian Networks with the bnlearn R Package. *Journal of Statistical Software 35*(i03).

Stamatelatos, M., W. Vesely, J. Dugan, J. Fragola, J. Minarick, and J. Railsback (2002). Fault tree handbook with aerospace applications.

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