Automated Fault Tree Learning from Continuous-valued Sensor Data: A Case Study on Domestic Heaters

Bart Verkuil[1], Carlos E. Budde[2], Doina Bucur[3]

[1,3] *Data Management & Biometrics, University of Twente, Enschede, 7522 NB, The Netherlands*
d.bucur@utwente.nl

[2] *Security, University of Trento, Trento, I-38122, Italy*
carlosesteban.budde@unitn.it

ABSTRACT

Many industrial sectors have been collecting big sensor data. With recent technologies for processing big data, companies can exploit this for automatic failure detection and prevention. We propose the first completely automated method for failure analysis, machine-learning fault trees from raw observational data with continuous variables. Our method scales well and is tested on a real-world, five-year dataset of domestic heater operations in The Netherlands, with 31 million unique heater-day readings, each containing 27 sensor and 11 failure variables. Our method builds on two previous procedures: the C4.5 decision-tree learning algorithm, and the LIFT fault tree learning algorithm from Boolean data. C4.5 pre-processes each continuous variable: it learns an optimal numerical threshold which distinguishes between faulty and normal operation of the top-level system. These thresholds discretise the variables, thus allowing LIFT to learn fault trees which model the root failure mechanisms of the system and are explainable. We obtain fault trees for the 11 failure variables, and evaluate them in two ways: quantitatively, with a significance score, and qualitatively, with domain specialists. Some of the fault trees learnt have almost maximum significance (above 0.95), while others have medium-to-low significance (around 0.30), reflecting the difficulty of learning from big, noisy, real-world sensor data. The domain specialists confirm that the fault trees model meaningful relationships among the variables.

1. INTRODUCTION

Fault tree analysis is a world-leading standard for safety and reliability assessment (Ruijters & Stoelinga, 2015; Vesely et al., 2002), with growing applications in emergent fields such as cybersecurity (Nagaraju, Fiondella, & Wandji, 2017).

https://doi.org/10.36001/IJPHM.2022.v13i2.3160

Fault trees are logical models for the propagation of basic functional failures. From an engineering perspective, a fault tree (FT) is a graphical representation of the possible failure modes of a system, i.e. the distinct observable failure processes of system functions, broken down into intermediate failures and their interactions (Ruijters et al., 2019).

FT models are ubiquitous for reliability, availability, maintainability, and safety (RAMS) analyses as they are interpretable by human experts, and extremely versatile: an FT can describe the failure behaviour of a system or process to almost any required level of detail. However, the traditional expert-driven FT building practice is resource intensive, subjective, and error-prone (Ruijters & Stoelinga, 2015). As a result, engineers and researchers are looking for ways to automate this modelling step (Nauta, Bucur, & Stoelinga, 2018; Lazarova-Molnar, Niloofar, & Barta, 2020; Ton et al., 2020).

1.1. Building FTs directly from data

The advent of the big data era has opened new possibilities for the automatic construction of FT models. At the same time, it exacerbates the combinatorial explosion in the number of models that can be derived. This is inherent to the structure of an FT, where each data variable is a potential leaf or gate.

Fault Tree Models. Technically, an FT is a single-rooted directed acyclic graph (DAG), whose leaves are called basic events (BEs) and represent indivisible failures in the system, such as "no power input" or "insufficient water supply". These BEs are connected to intermediate event nodes (IE): when the failure corresponding to a BE takes place, it propagates towards the connected IEs. In turn, each intermediate event has an output that can be connected as input to an upper IE, thus generating the DAG structure—see Fig. 1.

When the inputs of an intermediate event receive a fail signal, the IE can propagate this failure to its output. The propagation mechanism is defined by a logical gate that labels the IE:

OR gates propagate a failure if any input fails; AND gates only do it when all its inputs fail. There are further gates such as VOT: for an in-depth description of FTs we refer the interested reader to (Ruijters & Stoelinga, 2015).

The root of the tree is identified with the node at its top, called the top level event (TLE). The TLE represents the main event of interest that the FT models, e.g. a heater system failure. In turn, the BEs represent elemental failures that can lead to such an event, e.g. "insufficient water supply". The FT is said to fail, when the failures of certain BEs propagate and cause failures that eventually reach the TLE.

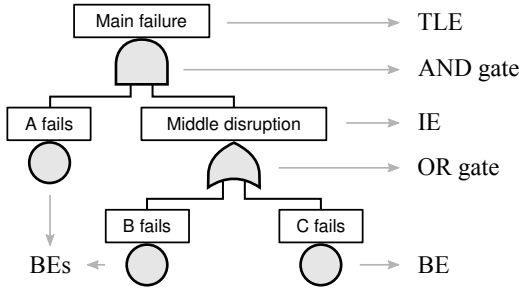

Figure 1. **Fault tree model** structure and terminology

FT construction. Traditionally, building an FT is a human-consensus process that includes experts from the areas related to the corresponding asset/system (Vesely et al., 2002; Berk, 2009). The process starts with the identification of the TLE, which defines the type of failure to study. This is a non-trivial initial decision: compare e.g. "heater does not start" vs. "heater stops after short operation". Both involve a heater system failure, but the causes leading to each type of failure can be completely different, and this impacts the resulting FTs (Ruijters & Stoelinga, 2015).

Once the TLE has been agreed upon, a functional analysis is carried out to determine the relevant causes that could lead to it. Then the causes of these causes are investigated and so on, until failures that are deemed elemental (the BEs) are reached. All of this involves expert knowledge, technical manuals, revisions, and discussions that span for weeks or months.

Big data. However, increasingly many assets are manufactured to record elemental failures (Booleans) and performance indicators (continuous sensor readings); and ever since the Internet of Things began, these readings are poured into the data streams of most companies (Evans, 2011).

This brings a new approach to risk analysis, where data comes first and (some) failures can be found based on it. More precisely, modern companies can take advantage of the data already available to them, determining its failure-predictive capabilities. However and unlike FTs, the learning models used to find correlations between failures and data are not necessarily human-intelligible. This hinders an adoption of the big data solution by many industrial sectors, specially those that traditionally use conservative risk management processes, e.g. aviation, nuclear power, housing, and railway operation (Chen, Ho, & Mao, 2007; Griffor, 2016).

Coincidentally, such industries are among the main users of FTs for safety analysis, given the versatility and high degree of explainability of FTs. These industries, and any companies with online readings of sensor and failure data, can greatly benefit by deriving FTs automatically from such data. That is the main contribution of this study: *a scalable learning algorithm to learn FTs from continuous and Boolean data, with a demonstration on the assets of a company that operates over 30 million domestic heaters across The Netherlands*.

Prior work. There is recent work on deriving FTs from failure data. This data can be obtained automatically—via online data streams—or semi-automatically—recorded in situ and then collected. In either case and to the best of our knowledge, all literature assumes a Boolean data input, i.e. true/false indicators of failures in the components of the main asset (Ruijters & Stoelinga, 2015; Nauta et al., 2018; Linard, Bucur, & Stoelinga, 2019; Lazarova-Molnar et al., 2020).

In contrast, our approach can process *continuous-valued data* coming from performance indicators, viz. sensors readings, which we relate to failure behaviour by learning a failure threshold per sensor. We use the C4.5 learning algorithm for this (Quinlan, 1993), originally designed for decision trees—another type of tree model which can visualise the hierarchical propagation of any decisions, including failure.

Decision trees are in some respects simpler than FTs: failure propagation in decision trees is not regulated by logical gates. However, in other respects the decision trees are richer in possibilities: they are learnt from continuous-valued data, and this can be done automatically, detecting thresholds for each variable such that a value below the threshold models a behaviour—e.g. failure—that differs from the behaviour when the value is above the threshold.

The connection between FTs and decision trees has been used before, for instance in (Lee, Alena, & Robinson, 2005) to derive the latter from existing FT models. Here we exploit the opposite direction, making use of the advantages that decision trees bring, to provide a realistic integrated method to learn FTs from real-world data. In that sense our work is closer in methodology to e.g. (Abdallah et al., 2018), where the main goal in our case is to use FT models to describe the governing fault mechanisms found.

Besides C4.5, our work builds on the LIFT algorithm (Nauta et al., 2018), which we modify to permit variable dataset sizes, and then use to learn the FT structure that fits best the failure under analysis. For this, LIFT employs the Mantel-Haenszel statistical test to correlate events. Other probabilistic causal models could be used, such as Bayesian Networks,

to learn relations between Boolean events and component failures. This has been studied in (Linard, Bucur, & Stoelinga, 2019; Linard, Bueno, Bucur, & Stoelinga, 2019), and applied to synthetic benchmarks, and failures in printer nozzles—all starting from Boolean input. However, using Boolean data (only) when continuous data is also be available, fails to capture failure mechanisms that surface in e.g. sensor readings (Ton et al., 2020). In view of this, our work presents a step towards more general, automatic learning algorithms.

1.2. Goal and approach

The main goal of this work is to use continuous sensor data collected by existent company assets, to build FTs for the failure mechanism of these assets—inasmuch as these failures are correlated to the data—in an automatic and scalable manner. To orient our investigation, we formulate this goal in term of the following research questions:

- **RQ1**: Which automatic method(s) can generate a structure of failure mechanisms, conforming to the FT standard, from a continuous-valued data input of sensor readings?
- **RQ2**: How to measure the quality of the resulting FTs, in terms of the correlation between the sensors data and the failures that they are expected to predict?
- **RQ3**: Is the learning method directly applicable to big industrial input data?

In RQ1, "automatic methods" stands for algorithms that do not require human input, and can learn the FTs solely from the sensor data fed into the learning procedure.

In RQ2, we are mostly interested in quantitative quality metrics. Nevertheless, we also include the qualitative aspect of *explainability*: these are enquiries to experts (in the application domain of the case study) on whether the resulting FT structures match their intuition, or if they are instead difficult to comprehend from their expert perspective.

For RQ3, we ask whether the runtime of this learning method scales with large data sizes, particularly when there are many failure notifications in the dataset to learn from.

Outline and main results. We implemented our results in an industrial case study, described next in Sec. 2. The details of our FT learning procedure are given in Sec. 3, followed in Sec. 4 by the results of its application to the case study.

Regarding RQ1, we found that *feeding continuous data to the C4.5 learning algorithm*—to interpret Boolean failure behaviour out of the sensor readings—*and then passing the results to LIFT*—to learn FT structures from the Boolean-interpreted data—*can generate FTs that relate continuous values to explainable failure mechanisms*.

As for RQ2, we used the concept of gate significance introduced in (Nauta et al., 2018), but replacing the PAMH score by the phi coefficient to allow for changes in the data size and dimensionality (Cramér, 1946). *The quality of an FT is then measured as the significance of its top gate, interpreted as a lower bound on the correlation between gate inputs and output*, across all the gates of the FT.

The application of this approach to our case study of domestic heaters resulted in 44 FTs. Of these, the FTs of failures related to the supply- or return-water temperature achieved significance as high as 0.96 (higher is better, the range of values is $[-1.0, 1.0]$). For almost every other failure, an FT was produced with significance around 0.3. Moreover, in cases where expert technicians had good understanding of the underlying failure mechanisms, they confirmed that the relevant sensor variables were included in the corresponding FTs.

The above also answers RQ3 positively: *we could successfully work with sensor data, recorded daily between 2015 and 2020 by Dutch heaters, totalling 31 million unique heater-day readings*. Per fault tree, the runtime of our learning algorithm in a standard desktop computer is in the order of minutes.

2. THE CASE STUDY

We analyse a large dataset containing time series of continuous-valued sensor data, collected automatically by Intergas Heating. The data describes the operation of Intergas domestic heaters—the Combi Compact HRE models—installed at consumers' houses throughout The Netherlands (Intergas, 2018). Between 2015 and 2020, data sensed locally by each connected heater was collected every $24\,\mathrm{h}$. The resulting dataset consists of multiple terabytes of raw data, with 31 million unique heater-day data collections, each with data from 27 built-in sensors and 11 self-diagnosed system failures.

Sensor variables. The 27 built-in sensors measure internal variables, such as water usage or the temperature at various zones, and an external variable, namely outside temperature. All variables are listed in alphabetical order in Table 1, with a description in terms of components of the Combi Compact HRE service instructions (Intergas, 2018). For each variable, four basic statistics are recorded or computed automatically every day: the minimum, maximum, average, and range.

Failure variables. The values of all sensors are monitored by a control loop running in the heater. Some sensors perform safety-critical measurements. For example, sensors s1_temp and s2_temp from Table 1 are located in the heat exchanger, and measure whether the heat from the gas is transferred to the water at the expected rate. If a critical sensor reports values surpassing a threshold, the heater diagnoses this as a *system failure* (a Boolean), and marks the failure in the data collected for the day. We study 11 such Boolean failures, listed in Table 2, which are present in the input data. The failure variables "Warning low t1" and "Warning low t2" are triggered by sensors s1_temp and s2_temp, so they are expected to be highly correlated. Similarly, the failure variable

Table 1. **Heater sensor variables** (27 in total)

Sensor variable	Description
bc_tapflow	Water usage (L/min)
boilertemp	Machine temperature (°C)
burnerstarts_24h	Number of starts per 24 h
ch_pressure	Water pressure (bar)
flue_sided_resistance	Torque for fan rotation in flue duct (N m)
gasmeter_ch_24h	Gas used per 24 h for central heating (m^3)
gasmeter_dhw_24h	Gas used per 24 h for domestic hot water (m^3)
heaterload_ch_24h	Delivered power % of full power over last 24 h, per hour, in central heating mode (%)
heaterload_ch_total	Delivered power % of full power over last 24 h, total, in central heating mode (%)
heaterload_dhw_24h	Delivered power % of full power over last 24 h, per hour, in domestic hot water mode (%)
heaterload_dhw_total	Delivered power % of full power over last 24 h, total, in domestic hot water mode (%)
heatertemp	Boiler temperature (°C)
io_curr_high	Ionization current on high output power (µA)
io_curr_low	Ionization current on low output power (µA)
outside_temp	Outside temperature (°C)
override_outside_temp	Alternative temperature measure (°C)
pump_pwm	Pulse-width modulation control signal for modulating pump (%)
room_override_zone1	Override temperature for zone 1 (°C)
room_override_zone2	Override temperature for zone 2 (°C)
room_set_zone1	Set temperature for zone 1 (°C)
room_set_zone2	Set temperature for zone 2 (°C)
room_temp_zone1	Measured temperature for zone 1 (°C)
room_temp_zone2	Measured temperature for zone 2 (°C)
s1_temp	Supply water temperature (°C)
s2_temp	Return water temperature (°C)
s3_temp	Warm water temperature (°C)
waterflow_ch	Water used for central heating (L)

"Warning low ch_pressure" reports low water pressure. Depending on the type of heater, a normal pressure is between 1 and 1.5 bar; by the company's business model, a value of 0.49 bar triggers a failure diagnosis. No other failure variable is directly associated to a sensor variable from Table 1.

Table 2. **Boolean failure modes** (11 in total)

Failure	Description
Lockout code 0	Sensor fault at self check
Lockout code 4	No flame signal
Lockout code 5	Poor flame signal
Lockout code 8	Incorrect fan speed
Lockout code 11	Fault in S1 flow (vent)
Lockout code 13	Fault in S1 flow (switch)
Warning flame lost	Flame lost
Warning ignition failed	Ignition failed 4 times
Warning low ch_pressure	Channel pressure below 0.49 bar
Warning low t1	Temperature s1_temp at 0 °C
Warning low t2	Temperature s2_temp at 0 °C

In summary, the table of available data consists of 27 real-valued sensor columns—each with four daily statistics computed over high-frequency readings—and 11 Boolean failure columns. Each row in this data table corresponds to the readings of a heater during a day of operation. We analyse each failure mode independently, using all sensor variables and data collected from all the heaters which exhibited such a failure, to gain a general understanding of how raw sensor readings are associated to each system failure.

Data quality. We remove duplicates from the raw dataset, as well as data from heaters with clearly corrupt readings, e.g. with values out of feasible physical ranges. Despite this preprocessing, a large amount of noise remains in the data due to individual sensors with temporal malfunction or disconnections: these can report values near the minimum or maximum range, or the value zero. For example, some temperature sensors occasionally report the value 327.67 °C, which is the maximum recordable temperature[1]. Since they occur in real data, we choose to preserve these extreme readings in the dataset, and learn from them. This data-cleaning process is, as usual, heavily dependent on the nature of the data; as such, we do not include it as part of our proposed solution to the research questions.

There are also missing readings, caused either by a failed heater or by failed network communication during data collection[2]. Fig. 2 shows the number of (unique) heater readings recorded per day, from the beginning of data recordings at the company in 2015. The positive slope of the curve reflects the installation of new heaters by Intergas across The Netherlands. The drops in the line indicate missing data: while roughly 90% of the heaters on record miss at least one reading during this five-year period, data overall remains abundant.

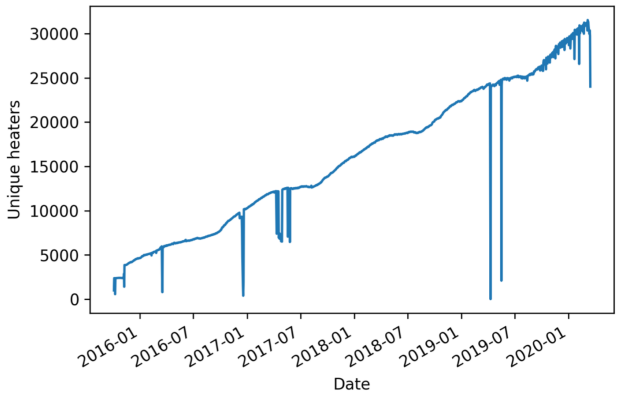

Figure 2. **Number of unique heaters recorded per day**. Drops in the curve indicate missing values, due to malfunctioning heaters or network communication issues.

3. METHOD

The data from the previous section is automatically collected by the Combi Compact HRE heaters of Intergas. To address RQ1 we design an algorithm that uses it to learn a fault tree for each system failure, without human input in the process.

Since the variables from Table 2 were designed as Boolean

[1]This is due to the use of 16-bit registers to hold signed integers, which results in a maximum value of $2^{15} - 1 = 32767$.

[2]The latter is explained by the use of the UDP transport protocol, which does not guarantee delivery nor duplicate protection.

indicators of heater failure, we use them as TLEs of our fault trees. Below these TLEs, the intermediate and basic events will be formed with automatically-selected sensor variables, which are found to be correlated to the top event via the methods detailed in Secs. 3.1 and 3.2. As a result, the gates of the FT link the behaviour of the sensors to the TLE, modelling how failures (are learnt to) propagate across system subcomponents. However, fault trees encode Boolean relations, while our sensor readings are real-valued variables. We bridge this gap by discretising each sensor variable.

3.1. Learning failure thresholds for sensor variables

The first step of our approach is to learn real-valued *thresholds* for the sensor variables, such as $27.21\,^{\circ}\mathrm{C}$ for the minimum daily value of the temperature variable s2_temp from Table 1. Each threshold applies only in the context of a particular failure mode: for instance the previous example is related to the failure variable "Lockout code 4". In contrast, the threshold $38.81\,^{\circ}\mathrm{C}$ was found for the range of daily values of s2_temp in relation to the failure variable "Lockout code 11".

To evaluate how good a threshold $\theta \in \mathbb{R}$ is, we use a metric based on information gain for Boolean variables. Essentially, the gain measures the correlation between the failure variable, and the sensor variable split by the threshold θ. This follows the internal logic of the C4.5 algorithm (Quinlan, 1993). We now explain in detail how we use it for our purposes.

An optimal threshold is learnt for each combination of sensor variable, statistic, and failure variable. Let s denote the statistic of a sensor variable (e.g. minimum daily value of s2_temp), and f a failure variable (e.g. "Warning low t1"). These are two columns in our data table: s is a real-valued column, and f a Boolean-valued column. In the corresponding dataset $D = s \mid f$, let p_0 denote the proportion of class 0 in column f: this represents normal operation, viz. absence of failure recordings. In turn, let p_1 denote class 1 (failure). The *information entropy* E is then defined as a weighted sum of these class probabilities, which reaches value 1 if the classes are equal in size, and value 0 if one class is empty:

$$E(D) = -\big(p_0 \cdot \log(p_0) + p_1 \cdot \log(p_1)\big).$$

Then, the dataset $D = s \mid f$ is split by a proposed threshold $\theta \in \mathbb{R}$ into a left and a right split. The left split, denoted $D^{\theta}_{\leqslant} \subseteq D$, are the rows where the sensor values s are lower than θ. The right split $D^{\theta}_{>} \subseteq D$ is analogously defined.

The gain of a threshold. The *gain* $G^{\theta}(D)$ of a proposed threshold θ for the dataset D is defined as the difference in entropy between the unsplit dataset, and the weighted sum of the entropy values after the split:

$$G^{\theta}(D) = E(D) - \frac{|D^{\theta}_{\leqslant}|}{|D|} E\big(D^{\theta}_{\leqslant}\big) - \frac{|D^{\theta}_{>}|}{|D|} E\big(D^{\theta}_{>}\big),$$

where $|D|$ is the size in number of rows of dataset D, so the weights are the fractions of readings on each side of the split.

Intuitively, G^{θ} measures the amount of entropy removed by the split at θ. In a perfect split, all sensor values s below θ would be part of one class of f (either normal or failure), and the remaining sensor values would be part of the other class. More generally, the better the threshold, the higher the gain.

To find the optimal threshold for a dataset $D = s \mid f$, our algorithm compares the gains of the possible thresholds in the range of s. Fig. 3 illustrates the selection of a threshold for the daily minimum of the sensor variable s2_temp w.r.t. the failure variable "Warning low t1". These variables are correlated only to a certain degree, as shown by the overlapping histograms: the blue histogram counts the number of min s2_temp values on days with normal heater operation (w.r.t. the failure "Warning low t1"); the orange histogram are values on days when that failure was recorded. Clearly, normal values are related to higher temperatures, while the temperature measured on days with failures are comparatively lower. The gain of each possible threshold is plotted in Fig. 3 as a red line at the bottom. The sensor value that maximises the gain is the optimal threshold that our method learns, in this case $21.0\,^{\circ}\mathrm{C}$, shown as a red dashed vertical line.

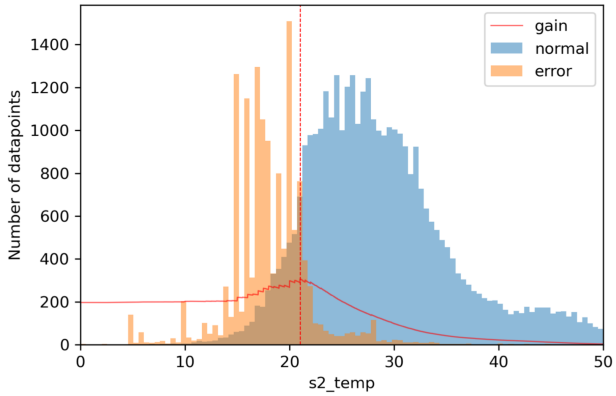

Figure 3. **Threshold selection** for the daily minimum of the sensor variable s2_temp, w.r.t. the failure variable "Warning low t1". The optimal threshold is $\theta = 21$, corresponding to gain $G^{\theta} = 0.60$ (gain values are plotted with a 500 factor). The threshold applies to the values of s2_temp: a good threshold splits the normal and error histograms.

Besides the optimal threshold, the algorithm returns the side of the split with the highest correlation to the failure class. This discretises the sensor variable s into a Boolean, with values 0 on the split (left or right) with a majority of normal readings, and 1 on the split with a majority of failure readings.

In Fig. 3, a fault tree for the failure variable "Warning low t1" might use as basic or intermediate event the Boolean condition $\min(\text{s2\_temp}) \leqslant 21$, associated to that system failure.

3.2. Learning the fault tree

In the second step of our approach, we use the Boolean data obtained during threshold learning, to generate and evaluate an FT for each failure mode. Each input dataset contains all sensor variables (in one of their daily statistics) and one failure variable.

As an example of data availability for the failure variable, Fig. 4 shows the number of notifications per day for "Warning low ch_pressure". The rising trend is due to the increasing number of heaters installed—c.f. Fig. 2. Seasonality is clear, as significantly more notifications are sent in months with lower outside temperatures in The Netherlands. For this failure mode, the dataset contains 52489 daily data points with failure notifications.

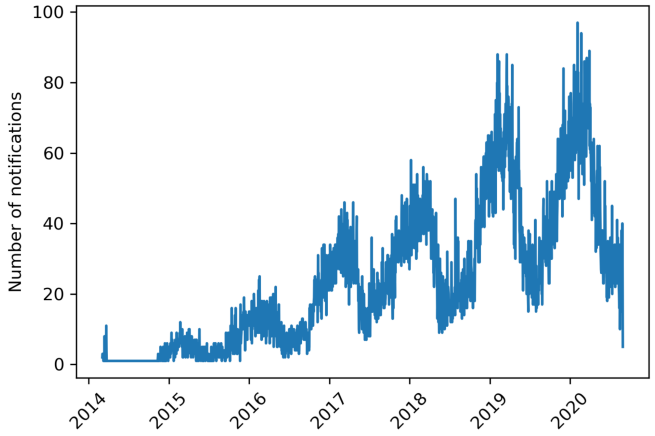

Figure 4. **Daily notifications** for "Warning low ch_pressure"

To each such data point showing failure, we add as counterpart a data point that describes a *normal* operation regime. More than one choice may be valid for this. We select the most recent data point showing normal operation for the same heater. The nearest future datapoint after the heater is fixed would be another valid option, but options that take into account a longer history of operation for the heater are less valid, because the heaters have different life spans (or ages) in the dataset. Adding this counterpart leads to a combined, balanced dataset of roughly twice the size. This is the dataset used to generate the fault tree.

To learn the FT we use an improved version of our previous algorithm LIFT (Nauta et al., 2018). Given a Boolean failure variable and the set of all real-valued sensor variables, thresholds are first learnt for the discretisation of the sensor variables into Boolean variables—henceforth referred to as *thresholded variables*—as described in Sec. 3.1.

We then initialise a fault tree T with only the failure variable as TLE, and proceed iteratively as shown in the flowchart in Fig. 5. At each iteration, the algorithm chooses a thresholded variable that is not yet the output of a gate in T, and greedily searches for the best possible gate that can be constructed. A logical gate is defined by its type (e.g. AND) and input variables. LIFT explores all combinations of gate types and subsets of thresholded variables, up to a configured maximum number, set to 3 for our experiments.

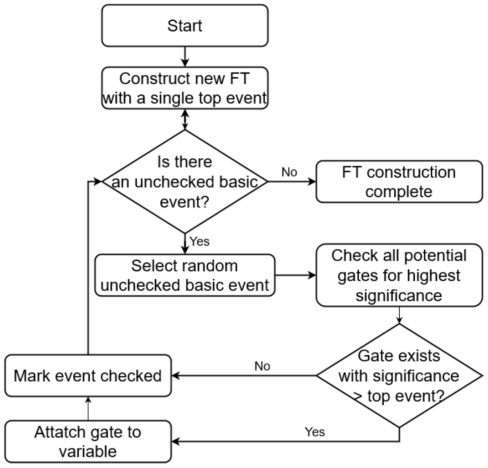

Figure 5. **The learning algorithm**

The significance of a gate. During search, the gate with maximum *significance* is chosen. Intuitively, the significance of a gate measures the positive correlation between the output and the inputs of a gate. The original implementation of LIFT used the Mantel-Haenszel statistical test for this (Nauta et al., 2018). Instead, here we use the *phi coefficient* as significance score (Yule, 1912; Cramér, 1946). It is a correlation coefficient for binary variables, whose value is 1 if the two variables are in complete agreement about their values, 0 indicates no relationship, and -1 complete disagreement. In particular, the phi coefficient remains stable when the size of the dataset changes, unlike the Mantel-Haenszel statistical test.

For an illustration consider *(i)* the failure variable "Lockout code 11", *(ii)* the statistic that measures the range of values of a sensor variable in a day, and *(iii)* the proposed gate:

$$\text{AND}(\text{s1\_temp} > 88.18\,\textbf{,}\,\text{pump\_pwm} > 0.0).$$

The significance measures the positive association of two variables: **1.** the output of the gate, i.e. the value of "Lockout code 11"; and **2.** the Boolean expression, i.e. the AND gate over the thresholded sensor ranges "s1_temp" and "pump_pwn".

Additionally, we implement the following rules:

- A new gate is added to the FT only if its significance is higher than or equal to that of the top gate.
- To limit the search space, each sensor variable is used at most in one location in the fault tree.

The purpose of these rules is to reduce the search space, thus lowering the runtime of LIFT. This is also the reason why the

maximum number of inputs chosen for the gates was set to 3 for our experiments. Therefore, removing these rules can only increase the quality of the resulting FTs—at the cost of increasing the computation time.

The construction of an FT terminates when no gate can be added that increases its significance, or when all sensor variables have already been used. The resulting fault tree T is then reported together with the significance of its TLE.

Note that, since the algorithm only adds gates to T that have higher significance than its TLE, the TLE significance is *a lower bound of correlation* between gate inputs and output across all the gates in T. Also, we annotate the BEs of T with the probability p of them occurring in the dataset—but, as indicated above, this probability is relative to a dataset which has been balanced between failure and normal operations. As such, p should only be interpreted in this context.

3.3. Computational complexity and execution runtime

To prepare the data for the learning procedure, the Intergas database (terabytes of raw data) was processed in its Apache Spark framework for big data, executed in a large computing cluster. This resulted in datasets prepared for FT learning, where the size of the resulting dataset was on the order of 10^5 records per failure mode and daily statistic. For these datasets, our learning procedure could be executed in a single computer, constructing each FT in a matter of minutes.

We highlight however that the procedure has high complexity: Namely, to find the optimal threshold during the discretisation of each sensor variable, all its unique values present in the data are considered. This has linear complexity in the dataset size, but must be repeated for each of the 44 combinations of failure variables and daily statistics for which different thresholds are necessary. Therefore, the aforementioned 10^5 records still lead to relatively long execution runtimes.

Furthermore, LIFT considers all feasible gate combinations: a combinatorial problem in the number of the sensor and failure variables. We cap this combinatorial explosion by limiting the maximum combination size to 3. Still, the resulting polynomial of degree 3 in the total number of variables causes the minutes (rather than seconds) runtimes mentioned above.

4. RESULTS

We obtain 44 fault trees with various significance levels. We first present the fault trees with the highest significance, then provide a summary of results. We note that, for this case study, from among the four basic statistics of the sensor data, the daily minimum, maximum, and range tended to be the most useful, leading to fault trees with higher significance than the daily average.

Fault tree for "Warning low t1". This failure mode makes for a good test of the fault-tree learning method, because it has a clear semantic link to the s1_temp sensor variable for the supply water temperature (which is thus expected to appear in the fault tree). This is indeed what we obtain: Fig. 6 shows the fault tree (with nearly maximum significance, 0.96) for this failure mode. When the daily statistic min(s1_temp) falls at or below $\theta = 0$, this occurrence strongly associates with the failure "Warning low t1", with gain 0.86. This is the main reason of the failure notification, but the tree provides more information. An interruption in the water supply, min(bc_tapflow) $\leqslant 0.0$, is also linked to the failure. Then, low minimum daily readings for s1_temp associate with a temporary lack of water pressure, min(ch_pressure) $\leqslant 0.0$. The final sensor variable, room_set_zone2, whose value is always below the threshold ($p = 1\%$), is due to the gate syntax requiring more than one input, and can be omitted.

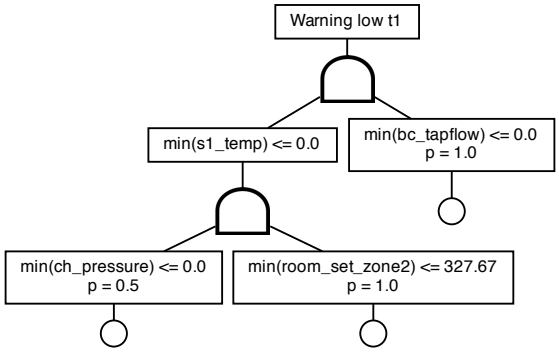

Figure 6. **Fault tree** for "Warning low t1". Significance 0.96.

An alternative fault tree (also with nearly maximum significance, 0.95) for this failure mode is in Fig. 7. This uses maximum daily readings and is less intuitive, since the maximum daily value of s1_temp are not relevant for the failure, the way the minimum daily value is. Instead, two other daily maximums are significantly associated with "Warning low t1": the sensor variables boilertemp (whose maximum daily values above $\theta = -34.03$ are strongly associated with the failure "Warning low t1", with gain 0.81) and outside_temp (similarly, above $\theta = -11.48$ with gain 0.84). These extremely low temperature thresholds are explained by the behaviour of the temperature sensor: it records in the data a very broad range [-51, MAX] $^\circ$C of maximum temperature values, where MAX is the maximum possible stored value on the sensor register (327.67 $^\circ$C). For the two sensors present in this fault tree (boilertemp and outside_temp), the MAX temperature occurs in the data much more often than realistic temperatures, and thus must signal a local sensor failure. When both of these sensors record extreme maximum daily values, this associates with the failure mode "Warning low t1", and could thus be used as an early warning sign.

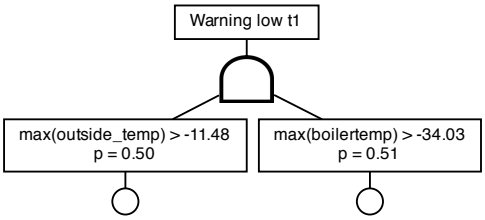

Figure 7. **Fault tree** for "Warning low t1". Significance 0.95.

Fault tree for "Warning low t2". This failure variable is similar to "Warning low t1" in that it has a semantic link to a temperature sensor variable, here s2_temp. In the resulting fault tree shown in Fig. 8, s2_temp appears as expected, with minimum daily values at or below $\theta = 0$ strongly associated with "Warning low t2", gain 0.87. In this case though, the fault tree also shows a different intuitive association: a temporary lack of water pressure (ch_pressure) `or` a low supply water temperature (s1_temp) associate with a low return water temperature (s2_temp). It is also likely that this is a causal, not only correlational, relationship between these sensor variables, since the water supply system precedes (and must affect) the return water system.

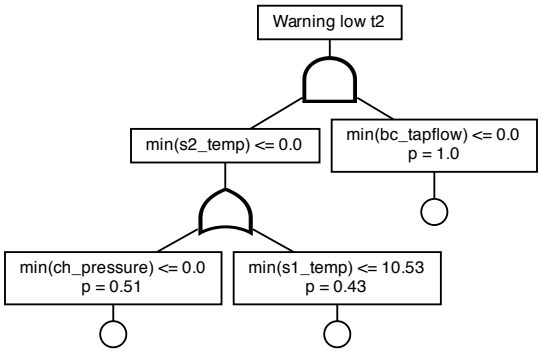

Figure 8. **Fault tree** for "Warning low t2". Significance 0.96.

Also here, an alternative fault tree (significance 0.95) for this failure mode is in Fig. 9, now using the daily range statistic. "Warning low t2" associates with days with large variation in outside temperature, range(outside_temp) $>$ 13.97 (gain 0.85), and the sensed machine temperature range(boilertemp) (gain 0.84). For the sensor outside_temp, the range readings are above 0 and mostly below 80 °C. This range of outside_temp itself associates with the range of ch_pressure (gain 0.75) and that of s2_temp (gain 0.73), sensor variables which were also found predictive in the FT from Fig. 8 using the minimum daily readings. We note that the daily minimum of s2_temp is more predictive of "Warning low t2" than the range of s2_temp: this variable appears higher in the FT using daily minimums. However, overall, both minimum and range daily readings are similarly predictive for this failure, with small and significant FTs obtained.

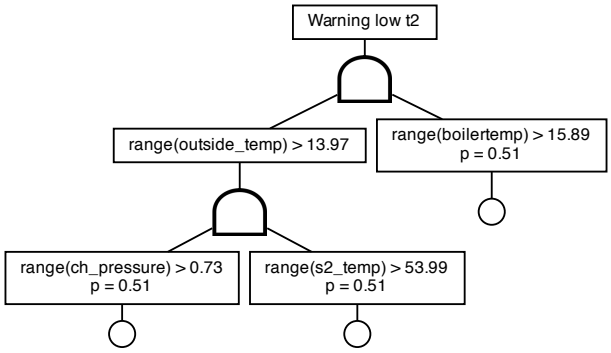

Figure 9. **Fault tree** for "Warning low t2". Significance 0.95.

Fault tree for "Lockout code 11". The fault trees obtained so far for temperature-related failure modes had nearly maximum significance levels, suggesting the presence of strong associations among the variables. This is not always the case: other failure modes are more complex, and much harder to model out of noisy data, resulting in lower significance levels and deeper fault trees.

For the failure mode "Lockout code 11" (a sensor fault also related to s1_temp), the fault tree with the highest significance is shown in Fig. 10. This tree uses the daily range statistic of the sensor variables. It has a significance of 0.35, meaning that, at every gate, the input-to-output correlation is at least 0.35. In other words, only part of the reasons for failure are learnt out of the data available, and either more exist but are not sensed in the data, or the failure is partly random. We describe below the associations which were learnt.

The left-hand side of the tree suggests that the failure can be predicted by the behaviour of the water supply temperature s1_temp, although only partly (gain 0.09). The failure correlates with a very large daily range for s1_temp: this range varies between 0 and 180.2 °C in the data (a large range, due to the presence of extreme temperature readings, as described for "Warning low t1"). The higher the daily range value, the more likely the failure is. In turn, range(s1_temp) is correlated with the ranges of the other two temperature sensors s2_temp and s3_temp, whose values vary mostly between 0 and 100 °C in the data. The range of the return water temperature s2_temp is then correlated with (and possibly caused by) the range of the current on high output power io_curr_high and the resistance of the flue duct flue_res, with higher values always signalling failure. The right-hand side of the tree states that above-zero control signal for the modulating pump (range(pump_pwm), with values in [0, 100] %) is also a predicting factor for this failure mode, although also weakly (gain 0.03). A combination of other technical variables are further predictive for range(pump_pwm).

An alternative FT for "Lockout code 11", obtained using minimum daily sensor data, is shown in Fig. 11. It is much

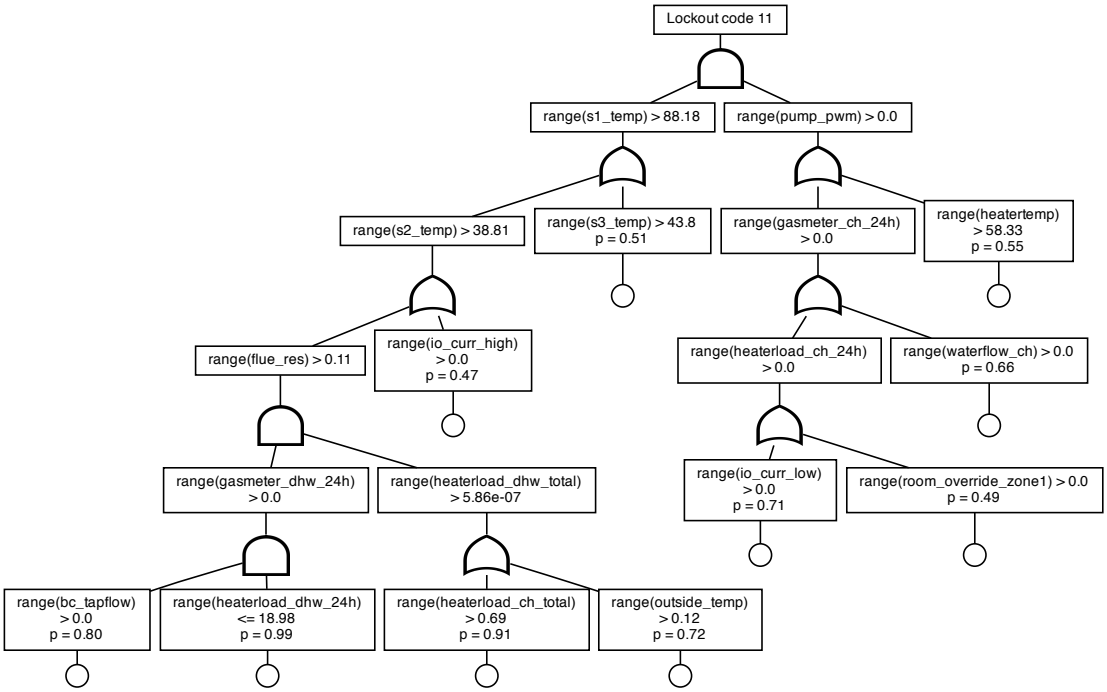

Figure 10. **Fault tree** for "Lockout code 11". Significance 0.35.

smaller and slightly less significant (significance 0.27). The minimum daily readings for four temperature sensors (s1_temp to s3_temp and heatertemp) are slightly predictive of this failure: in all cases, if the minimum daily temperature falls below a threshold of 24-27 °C, the failure is more likely to occur.

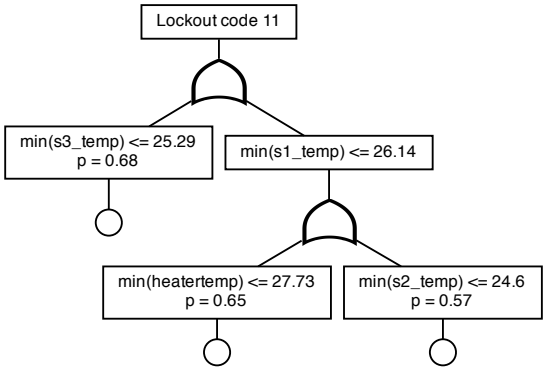

Figure 11. **Fault tree** for "Lockout code 11". Significance 0.27.

Such alternative fault trees learnt a partial pattern for the failure from different sets of daily statistics. They can then be used together, as a predictive ensemble for the failure mode.

Fault tree for "Lockout code 4". This failure mode triggers when there is no flame signal, so it is of a different nature than the previous examples. However, the set of predictive sensor readings and their thresholds are similar to those which also

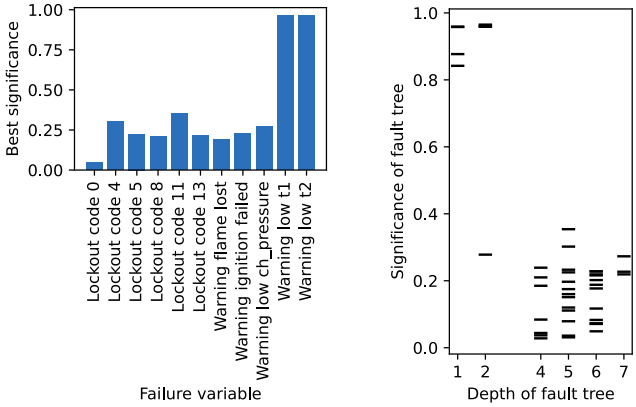

Figure 12. **Summary of results**: significance scores (left) and significance vs. depth for all fault trees obtained (right).

predict other failures, so the failures themselves are likely dependant. The best fault tree obtained for "Lockout code 4" is shown in Fig. 13 (significance 0.30) and uses minimum daily readings. Some temperature sensors (s1_temp to s3_temp) appear with similar thresholds as for "Lockout code 11" (the fault tree in Fig. 11), and they are the sensor variables with the highest gain, so strongest individual association to the failure. In total, 20 out of 27 sensor variables available are thresholded and included, leading to a complex fault tree.

Summary of results and evaluation. We obtained 44 FTs of various depths, internal structure, and significance scores: Fig. 12 provides a quantitative overview. Trees with low depths

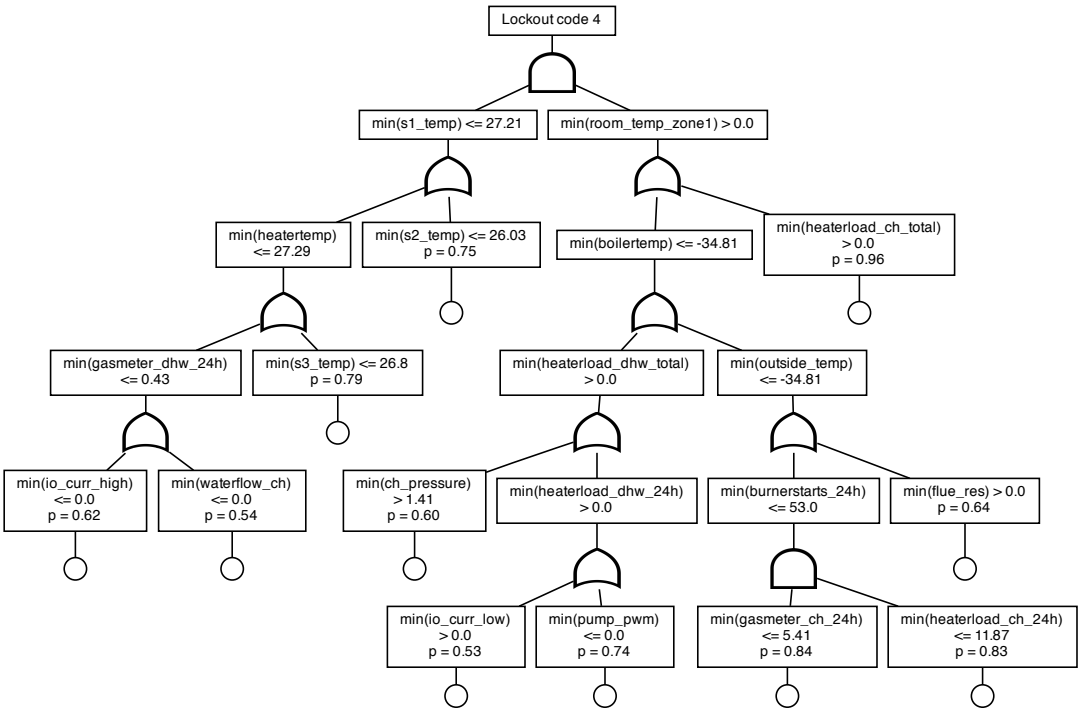

Figure 13. **Fault tree** for "Lockout code 4". Significance 0.30.

tend to have high significance. With regards to the best significance score per failure variable, we observe three classes: (a) failure variables for which the significance score was nearly maximum ("Warning low t1" and "... t2"), (b) those for which the significance is medium-to-low, between 0.21 and 0.35 (the majority of the cases), and (c) those for which no significant FT could be learnt (one case: Lockout code 0). The depths of the trees, defined as the number of gates on the longest path from the TLE to a BE, varies between 1 and 7. For our case study, we find that the FTs are deep only when they also have low significance.

The runtime of the method to construct one fault tree, for this dataset and algorithm configuration, is a few minutes (for threshold computation) and a few tens of seconds (for the tree generation) on an average consumer desktop computer.

5. DISCUSSION AND CONCLUSIONS

Evaluation. To evaluate the FTs learnt, we consulted with Intergas domain specialists, who confirmed that the fault trees do contain meaningful relationships between variables. In cases where the domain experts already had good understanding of how a failure occurs, they also confirmed that the important variables were in the tree.

Limitations. Many design decisions may affect the accuracy of the resulting fault trees. In particular we note:

High runtimes The method has high computational complexity, although it can be configured so that it remains feasible. Our runtime per fault tree with a dataset on the order of 10^5 records and 27 sensor variables per record is on the order of minutes in a standard desktop computer.

Lack of optimality This learning procedure, as many classical machine-learning algorithms such as those based on decision trees (Breiman, Friedman, Olshen, & Stone, 1984) is a greedy heuristic: while the gate with the highest significance available is constructed at every step, the average significance score over the entire fault tree may not be maximum. As a consequence, there may be other good alternatives for the tree structure.

Single threshold: bias is likely Model bias can stem from making oversimplified assumptions about the data. We compute only one threshold per sensor variable (a potential cause of bias), and then use the result at most once in an FT. However, in some datasets, multiple thresholds for the same variable may be of interest, in different parts of the tree. We made the choice to disregard this option and precompute this single threshold per sensor variable and failure variable, to keep the runtimes feasible.

Real-world data is faulty and complex We observed unordinary sensor values (very low, very high, and zero), particularly from temperature sensors; these are documented in the sensor data sheets (Banner, 2021), and are unavoidable during the lifetime of a sensor. Both noise and unusual or missing (zero) values are pervasive in the data: we have only cleaned the dataset minimally. The

resulting FTs have learnt from these unusual values.

Overfitting to noise is possible In datasets where noise is frequent, it is possible that some of the low-significance trees overfit (learn from) noise.

No proven causality The fault trees learn correlations rather than causal relationships, which are difficult to extract from passive (non-interventional) datasets like ours.

No cover for dynamic failures We use the LIFT algorithm, which can learn static fault trees (AND and OR gates). However, some failures may be dynamic in nature, e.g. when failure order matters. This cannot be captured by static gates, and poses a much harder learning problem.

We note, nonetheless, that some of these limitations could be resolved in other case studies. We designed and configured the method to keep the runtime low, but some of the restrictions can be lifted if the runtime remains feasible.

Discussion of results. We were able to address our research questions to different degrees of success.

Regarding RQ1, the combination (and modification) of the C4.5 and LIFT algorithms allowed us to build FTs automatically from sensor data. The basic and intermediate events of these trees were chosen based on the correlations of failure behaviour and sensor values, which was our main objective.

Access to failure variables was instrumental to achieve automation, since these guide both steps of our learning algorithm. More precisely, for each FT: (1) we discretise all sensor values, by learning failure behaviour based on the available failure variables, and (2) we evaluate all possible combinations of FT structures, selecting a failure variable as TLE.

As mentioned above, our decision to discretise each sensor variable with a single threshold may be imposing unneeded limitations. Still, some resulting FTs achieved high quality, both according to our metrics and by expert assessment.

In that respect and regarding RQ2, we use gate significance—lifted to FTs by taking the significance of its TLE—as metric for the the quality of our trees. The original implementation of LIFT used a similar concept, which we modified as indicated in Sec. 3.2 to cater for variable dataset sizes. An extra advantage of our choice, for this case study, is that larger FTs had generally lower significance. Smaller FTs are easier to interpret by humans, and thus preferable for risk management.

Finally, RQ3 was answered in a positive manner, as our implementation could handle the input of a company with several millions of data readings. Although the initial data processing and filtering was performed in the cluster of Intergas, this is not considered a limitation since data protection regulations make it a usual case. More relevant are our rules to keep the learning problem within manageable size for a desktop computer. We achieved our goal, but loosing restrictions to favour the FT quality—e.g. allowing several thresholds per sensor variable—will have a direct impact in the execution runtime of the core learning algorithm.

Perspectives and future work. A first point of improvement would be to allow repeated uses of a sensor variable in an FT, with different thresholds, and a higher number of children per gate. The challenge lies in keeping the runtime reasonable, which may call for rules such as scoring and penalising sensor variables already present in an FT. Semi-automatic FT construction is promising for this, to mitigate the combinatorial explosion faced by the FT learning step. We intend to contribute to public benchmarks such as (Ruijters et al., 2019), by submitting the fault trees of our work.

ACKNOWLEDGEMENTS

Funded by the European Union under GA number 101067199 ProSVED. Views and opinions expressed are those of the author(s) only and do not necessarily reflect those of the European Union or The European Research Executive Agency. Neither the European Union nor the granting authority can be held responsible for them.

REFERENCES

Abdallah, I. N., Dertimanis, V. K., Mylonas, H., Tatsis, K. E., Chatzi, E. N., Dervili, N., ... Maguire, E. (2018). Fault diagnosis of wind turbine structures using decision tree learning algorithms with big data. *Safety and Reliability – Safe Societies in a Changing World*, 3053–3061. doi: 10.1201/9781351174664-382

Banner. (2021). *Sure Cross QM30VT2 vibration and temperature sensor.* (Datasheet)

Berk, J. (2009). *System failure analysis*. ASM International.

Breiman, L., Friedman, J. H., Olshen, R. A., & Stone, C. J. (1984). *Classification and regression trees*. Routledge.

Chen, S., Ho, T., & Mao, B. (2007). Reliability evaluations of railway power supplies by fault-tree analysis. *IET Electric Power Applications*, *1*(2), 161–172. doi: 10.1049/iet-epa:20060244

Cramér, H. (1946). *Mathematical methods of statistics (PMS-9)*. Princeton University Press. doi: 10.1515/9781400883868

Evans, D. (2011). *The internet of things: How the next evolution of the internet is changing everything* (Tech. Rep.). CISCO; San Jose, CA, U.S.: Cisco Internet Business Solutions Group (IBSG).

Griffor, E. (2016). *Handbook of system safety and security*. doi: 10.1016/C2014-0-05033-2

Intergas. (2018). *Combi Compact HRE. Installation, service and user instructions.* (Installation manual 88287806)

Lazarova-Molnar, S., Niloofar, P., & Barta, G. K. (2020). Data-driven fault tree modeling for reliability assessment of cyber-physical systems. In *WSC* (pp. 2719–

2730). IEEE. doi: 10.1109/WSC48552.2020.9383882

Lee, C., Alena, R., & Robinson, P. (2005). Migrating fault trees to decision trees for real time fault detection on international space station. In *2005 IEEE aerospace conference* (pp. 1–6). doi: 10.1109/AERO.2005.1559584

Linard, A., Bucur, D., & Stoelinga, M. (2019). Fault trees from data: Efficient learning with an evolutionary algorithm. In *SETTA* (Vol. 11951, pp. 19–37). Springer. doi: 10.1007/978-3-030-35540-1_2

Linard, A., Bueno, M. L., Bucur, D., & Stoelinga, M. (2019). Induction of fault trees through bayesian networks. In *ESREL*. doi: 10.3850/978-981-11-2724-3_0596-cd

Nagaraju, V., Fiondella, L., & Wandji, T. (2017). A survey of fault and attack tree modeling and analysis for cyber risk management. In *HST* (pp. 1–6). IEEE. doi: 10.1109/THS.2017.7943455

Nauta, M., Bucur, D., & Stoelinga, M. (2018). LIFT: learning fault trees from observational data. In *QEST* (Vol. 11024, pp. 306–322). Springer. doi: 10.1007/978-3-319-99154-2_19

Quinlan, J. R. (1993). *C4.5: Programs for Machine Learning*. Morgan Kaufmann.

Ruijters, E., Budde, C. E., Nakhaee, M. C., Stoelinga, M., Bucur, D., Hiemstra, D., & Schivo, S. (2019). FFORT: a benchmark suite for fault tree analysis. In *ESREL*. doi: 10.3850/978-981-11-2724-3_0641-cd

Ruijters, E., & Stoelinga, M. (2015). Fault Tree Analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, *15–16*, 29–62. doi: 10.1016/j.cosrev.2015.03.001

Ton, B., Basten, R., Bolte, J., Braaksma, J., Di Bucchianico, A., van de Calseyde, P., ... Stoelinga, M. (2020). PrimaVera: Synergising predictive maintenance. *Applied Sciences*, *10*(23). doi: 10.3390/app10238348

Vesely, W., Stamatelatos, M., Dugan, J., Fragola, J., Minarick, J., & Railsback, J. (2002). Fault tree handbook with aerospace applications. *NASA Office of Safety and Mission Assurance*. (version 1.1)

Yule, G. U. (1912). On the methods of measuring association between two attributes. *Journal of the Royal Statistical Society*, *75*(6), 579–652.