Data-Based Generation of Reliability Models

Author:
GÁBOR KEVIN BARTA

Supervisor:
SANJA LAZAROVA-MOLNAR

Faculty of Engineering
UNIVERSITY OF SOUTHERN DENMARK

JUNE 2018

Abstract

Fault tree analysis is a prominent method in analyzing the reliability of a system. It has gained widespread use in many areas such as the automotive and aviation industry. The fault tree of a system is usually designed by domain experts based on their knowledge, experience, and information gathered on the topic. Since the fault tree is designed based on estimation, the fault tree might not accurately reflect the true behavior of a system after it has been put into operation. Instead, deriving the fault tree from data would be a better alternative. This thesis presents a data-driven technique to extract the fault tree from time series data of a system. The extracted fault tree of the system is then analyzed to determine the reliability, maintainability, and availability of the system and its components. The results can then be used to schedule preventative maintenance or to detect and replace weak components for the purpose of increasing the reliability of the overall system.

Acknowledgements

Here I would like to acknowledge and express my gratitude for the people that have supported me and helped me in the journey of writing my thesis. I would like to thank my supervisor for her support and for the opportunities she has given me. I am grateful for having a supervisor like her. I would like to thank my mother and father for raising me with support and love. I would like to thank my sister for being inspiring and for helping me with the artistic design of the poster for my thesis. I would like to thank my flat-mates who were supportive while I was writing my thesis.

Author's declaration

I hereby solemnly declare that I have personally and independently prepared this paper. Except where indicated by specific reference in the text, the work is the candidate's own work. The paper or considerable parts of it have not previously been subject to any examination or assessment.

SIGNED: [illegible] DATE: 01.06.2018

TABLE OF CONTENTS

List of Tables

List of Figures

List of source codes

Chapter 1

Introduction

As engineers, we dedicate countless hours of our lives to build cars, airplanes, software, computers, and any kind of systems which are safe, reliable, and convenient. Perhaps one of the most important attributes is to make a system reliable, since without reliability we cannot expect it to function properly. We want reliable systems so they can serve us humans for as long as we need in order to make our lives better and simpler.

However, reliability can be a difficult metric to measure and engineers have struggled with it throughout history. Most of the tragic accidents that have happened in the field of engineering was due to overestimating the reliability of a system.

The modern feats of engineering from skyscrapers as high as mountains and airplanes as fast as sound have left people in awe. But none of these accomplishments could have been done with out careful consideration of reliability and making sure these monuments stand the test of time.

1.1 Motivation and Purpose

Fault tree analysis is a prominent method in analyzing the safety and reliability of a system [1, p. 30]. It has been adopted as a standard reliability model by many industries such as automobile, aviation, and nuclear power. Fault tree analysis investigates which components influence the failure of a system.

Fault trees are usually constructed by domain experts based on their experience and the information they have gathered on the topic. [2] Therefore these fault trees are constructed based on estimation and not on data.

Is there a way to derive the reliability model of a system from data? Investigating this question is the purpose of this thesis, to design and implement a method of automated derivation of fault trees from data.

Deriving the reliability model from data is incredibly useful, when determining the true behavior of a system after it has been put into use. Extracting a model from data is known as data-driven modeling. Data-driven modeling is gaining traction in many areas for its ability to analyze data from a system in order to obtain connections between the system state variables [3].

The techniques used by data-driven modeling include artificial intelligence, machine learning, and computational intelligence. Data-driven modeling techniques are also making advances in fault detection and fault diagnosis for industrial systems and processes [4]. Producing a fault tree using data-driven modeling methods yields a model that is considered a white-box model, since down to the level of individual components the behavior of the system can be determined. White-box models allow us to investigate and understand the inner behavior of the system.

Such a method of extracting a fault tree from data would be beneficial because by understanding the behavior of a real-life system weak components could be identified and replaced for purpose of improving the reliability of the overall system.

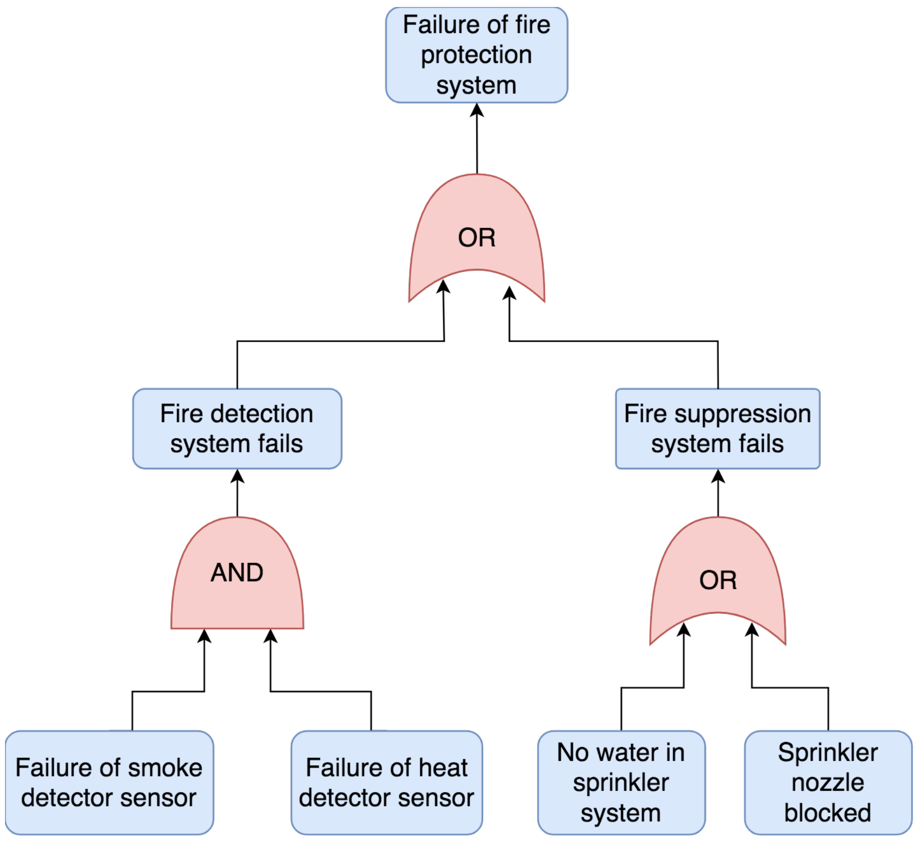

Failure of fire protection system
OR
Fire detection system fails
Fire suppression system fails
AND
OR
Failure of smoke detector sensor
Failure of heat detector sensor
No water in sprinkler system
Sprinkler nozzle blocked

Figure 1.1: Fault tree

1.2 Scope

The thesis is a proof of concept to evaluate whether it is feasible to generate the fault tree from data. To assess this assumption I had to abstract the fault tree to represent any system or process. I created a framework where I can construct arbitrary fault trees and simulate its behavior by generating the occurrences of the events into time series data. Then I developed a method of reconstructing the fault tree from the time series data. Afterwards I evaluated the reconstruction algorithm by comparing the original fault tree and reconstructed fault tree. The goal of the thesis is to research and develop a method of extracting fault trees from data.

1.3 Related Work

This section provides a summary of other studies and papers regarding fault tree analysis and data-driven modeling and also how it relates to my thesis and how their findings could be utilized.

Fault Tree Analysis Rao et al. use Monte Carlo simulation technique to assess the reliability and safety of systems represented by dynamic fault trees [5]. The Monte Carlo simulation is a method of generating times-of-failure and times-of repairs to investigate reliability, maintainability, and availability. Essentially, I employ the same time series generation method in my thesis. Lee et al. wrote a survey paper explaining different methods of fault tree analysis [6]. They mention techniques of automated fault tree construction such using Synthetic Tree Model (STM), Computer Automated Tree (CAT), which uses decision tables, and automatic fault-tree synthesis from the reliability graphs. S. Mukherjee and A. Chakraborty describe a technique to automatically generate fault trees using historical maintenance data in text form [7]. Their technique relies on domain knowledge and linguistic analysis. The above mentioned techniques also use data to construct a fault tree, however they do not use time series data, but rather other types of data such as text and reliability graphs.

Data-driven modeling Angelov et al. use a genetic algorithm to derive fuzzy rule-based models from data [8]. Their approach has advantage over black-box modeling since the internals of the system can be examined. The goal of my thesis is also to derive a model from data where the model can express information about the internal functions of the system under investigation. Tabesh et al. present techniques to predict failures and determine reliability in water pipe distribution networks [9]. They used artificial neural networks and neuro-fuzzy systems in their techniques and they used real-world water distribution networks for their data. The idea behind my thesis is also to be able to use the techniques on real-life systems and reconstruct a model representing them.

1.4 Outline of the Thesis

The outline of thesis is as follows. Chapter 2 provides necessary theoretical knowledge of probability theory and fault tree analysis to comprehend the methods used in the thesis. Chapter 3 explains in detail the implemented methods and techniques used to reach the goal of the thesis. Chapter 4 provides the results by running the algorithm on different scenarios. Chapter 5 evaluates the reconstruction algorithm, expresses it limitations, and provides a discussion about improvements that can be made. Finally, Chapter 6 concludes the thesis and mentions future work to consider.

Chapter 2

Background

In this chapter, I explain the prerequisites and theory needed to understand the thesis report. I explain concepts regarding reliability, maintainability, and availability and provide a basic overview of the mathematic and statistical fundamentals needed to understand these concepts. I explain in detail the aspects of fault tree analysis that is used in the solution. All the theory explained in this chapter was put into practice when developing the implementation. The structure and content of this chapter takes inspiration from *System Analysis Reference* [10] and *Life Data Analysis Reference* [11] to explain the above mentioned concepts.

2.1 System Fundamentals

System reliability is the study of systems and components in order to determine their reliability over the period of the system's life cycle. The reliability of a system is the probability of the system to function properly for an extended period of time [12].

We also have to discuss what we consider a system or component. Whoever is analyzing the system decides how much detail is needed to describe the system. Everything that is under the level of detail of the components can be omitted since we consider the components as a black box. From the point of view of the entire system we consider the system as a white-box. Therefore the reliability of the individual components can be measured and the component's relationship with the overall system can be determined.

Figure 2.1 shows an arbitrary system made out of three components. From the figure we know that these components somehow affect the overall system, but in this case we have no knowledge of their relationship.

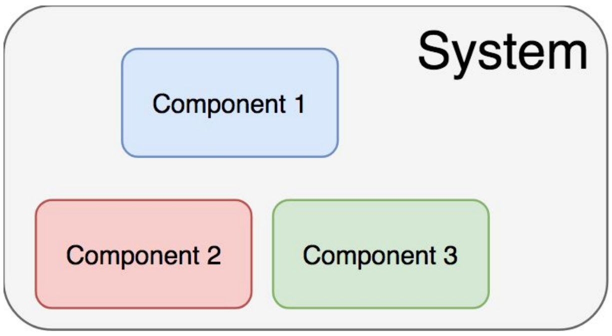


Figure 2.1: System and its components

Diagrams like reliability block diagrams (RBD) and fault trees show the relationship between the components and how they affect the overall system. The difference between the two is that a reliability block diagram is a success-oriented model while a fault tree is fault-oriented model [13]. The two most common relationship between components are *AND* and *OR*, in case we are talking about fault trees. In case of RBDs, it is called parallel and series, respectively. From here on, we will focus on fault trees therefore we use the terms *AND* and *OR* gates.

2.2 Probability

This section introduces basic concepts of probability which will be used later in the thesis. We only look at concepts needed for the understanding of the report, so this is no way considered a comprehensive section on probability. If further information is needed on the topic refer to other literature which cover it more in detail. This section explains how the probability concepts relate to reliability and fault trees.

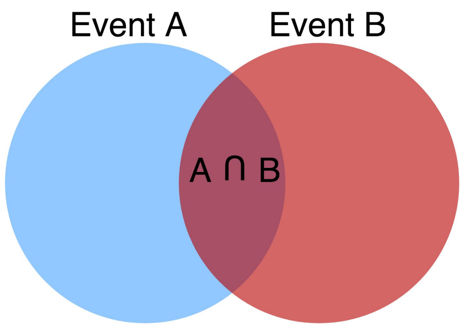


Figure 2.2: The intersection of event A and event B

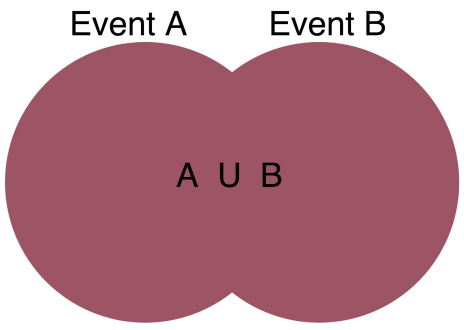


Figure 2.3: The union of event A and event B

Below are basic concepts and notations from probability and set theory.

- Experiment (E): An action which can have many outcomes.
- Outcome (O): A result from an experiment.
- Sample space (S): A set of all the outcomes from an experiment.
- Event: A collection of possible outcomes of an experiment, it is a subset of the sample space.
- $A \cup B$: The union of event A and event B is the set of outcomes which belong to event A, event B, or both event A and event B.
- $A \cap B$: The intersection of event A and event B is the set of outcomes which belong to both event A and event B.
- Compliment of event A ($\overline{A}$): All the outcomes that do not belong to event A.
- Empty set($\emptyset$): An empty set which does not have any outcomes.
- Probability is the measure of how likely is an event to happen.
- A: An event that can occur
- P(A): The probability that event A will occur
- $0 \leq P(A) \leq 1$: the probability that event A will occur has to be between 0 and 1, inclusive.
- $1 - P(A) = P(\overline{A})$ The probability of event A occurring subtracted from 1, has to equal the probability that event A will not occur.

In probability theory there are two main areas to cover, probability with discrete random variables and probability with continuous random variables.

2.2.1 Discrete Random Variables

Probability of discrete random variables is when the outcomes of an event can only be discrete values, values that we can count. The discrete random variable is a random variable that can such discrete values. For example, tossing a coin can result in two outcomes; the coin can land on tails or it can land on heads. The probability of both of those outcomes are 50-50%, for an unbiased coin. This also shows that the probability of each outcome in the sample space must some up to 1.

When we consider reliability, we use discrete probability to calculate the probability of failure for a component or system which does not depend on time. This means time does not have any affect on the probability of failure.

2.2.1.1 Reliability

Reliability is closely related to the probability of failure of a component. For example, a particular component has a probability of failure of 40%. From this we can conclude that the reliability of that component is 60%. The probability of failure is also called unreliability. Equation 2.1 shows the relationship between reliability and probability of failure.

$$Reliability = 1 - Probability\ of\ failure \tag{2.1}$$

2.2.1.2 Union of Events

Equation 2.2 shows the probability of the union of events A and B is equal to the probability of event A plus the probability of event B minus the intersection between the two events. The intersection is subtracted since the area should not be counted twice. This area can also be seen on Figure 2.2 as the burgundy colored area. The resulting area from the union of event A and B is shown on Figure 2.3.

$$P(A \cup B) = P(A) + P(B) - P(A \cap B) \tag{2.2}$$

2.2.1.3 Mutually Exclusive Events

Mutually exclusive events are events that cannot occur at the same time ($P(A \cap B) = \emptyset$). In this case, the equation simplifies to Equation 2.3.

$$P(A \cup B) = P(A) + P(B) \tag{2.3}$$

2.2.1.4 Conditional Probability

Conditional probability is the probability of one event occurring when another event has already happened. In this case, the outcome of one event is affected by another. Equation 2.4 shows how to calculate the probability of event A given that event B has occurred.

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \tag{2.4}$$

2.2.1.5 Independent Events

If we know that, for example, event B has no effect on the outcome of event A, they are said to be independent events. In that case, the equation can be simplified to Equation 2.5.

$$P(A \cap B) = P(A) \cdot P(B) \tag{2.5}$$

2.2.2 Continuous Random Variables

In probability, if the random variable can take up infinite number of values, we call it a continuous random variable. We think of these as values we can measure, such as time or distance. Particularly, in fault tree analysis we focus on how the probability changes over time.

We look at how to visualize the probability of continuous random variables using the probability density function and cumulative probability function. When discussing reliability, we call the probability distributions lifetime distributions to signify it represents the lifetime of a component. In the upcoming sections, we will cover the most commonly utilized lifetime distributions used to represent the reliability distribution for events.

2.2.2.1 Probability Density Function (PDF)

The probability density function, *f(x)*, of a continuous random variable, *X*, is one that satisfies Equation 2.6, where a and b are two numbers, and a is less than or equal to b. The equation states that if X takes a value between a and b then the probability of X is the area under the curve of the probability density function in the interval between a and b. In case of reliability, the PDF shows the relative frequency for times-to-failure with respect to time.

$$P(a \leq X \leq b) = \int_a^b f(x)dx \tag{2.6}$$

2.2.2.2 Cumulative Distribution Function (CDF)

The cumulative distribution function describes the probability of an event occurring up to a certain point in time. It is the cumulative values of the PDF. A value on the CDF curve at particular time is the area under the curve to the left of that point in time on the PDF. Equation 2.7 shows the mathematical relationship between the cumulative distribution function, *F(x)*, and the probability density function, *f(x)*. *F(x)* is the integral of *f(x)*.

$$F(x) = P(X \le x) = \int_0^x f(s)ds \tag{2.7}$$

The cumulative distribution function represent the probability of failure of a component. It is used to measure the probability that the failure will occur before time *t*. The CDF is also called the unreliability function and is denoted by *Q(t)*.

The way to calculate reliability from the probability of failure (or the unreliability function) is analogous with Equation 2.1. Below are mathematical equations explaining the relationship between the unreliability function, *Q(t)*, reliability function, *R(t)*, cumulative distribution function, *F(t)*, and the probability density function, *f(s)*.

$$Q(t) = F(t) = \int_0^t f(s)ds$$

$$R(t) = 1 - Q(t)$$

$$R(t) = 1 - \int_0^t f(s)ds$$

$$R(t) = \int_t^\infty f(s)ds$$

2.2.2.3 Failure Rate Function

The failure rate function describes the number of failures in one unit of time [12]. It can be calculated from the probability density function and the reliability function as seen in Equation 2.8. Using this equation the instantaneous failure rate can be determined at any point in time, it is also called the hazard function.

$$\lambda(t) = \frac{f(t)}{R(t)} \tag{2.8}$$

2.2.2.4 Mean Life

The mean life of a distribution is the average value of the continuous random variable. We learn later that for each life distribution it can be calculated differently using its parameters. Equation 2.9 is the general formula.

$$m = \int_0^\infty t \cdot f(t) dt \tag{2.9}$$

2.2.2.5 Parameter Types

Distributions have parameters that alter the way the base distribution looks. It is used to precisely fit the distribution for the application it used for. There are a maximum of three parameters we usually use to model lifetime distributions. [11, p. 18]

Scale The scale parameter is the most commonly used parameter. All the lifetime distributions we cover have the scale parameter. The scale parameter is responsible for stretching out the distribution along the horizontal axis.

Shape The shape parameter, as stated in the name, affects the shape of the distribution. Some lifetime distributions do not have this parameter since their shape cannot be changed, for example, the normal distribution has a predefined bell shape.

Location The location parameter is used to shift the distribution along the horizontal axis. In the case of lifetime distribution it represents a time shift.

2.2.2.6 Exponential Distribution

The exponential distribution is one of the most common distributions used for components when discussing reliability. The reason it is commonly used is because of its simplicity and also because it has a constant failure rate.

The exponential distribution takes one parameter, the scale parameter, which is indicated by a λ. The scale parameter, in the case of exponential distribution, is equal to the failure rate. It is the average number of failures per time unit. A characteristic of the exponential distribution is that it has a constant failure rate which does not depend on time. Figure 2.6 shows the failure rate of an exponential distribution with a scale parameter of $\lambda = 0.5$.

$$m = \frac{1}{\lambda}$$

The inverse of the the scale parameter is the the mean of the exponential distribution, the mean time to failure when we are discussing reliability. In this example, the mean time to failure is equal to *2*. Below are the formulas for the CDF, PDF, and reliability for the exponential distribution.

$$f(t) = \lambda e^{-\lambda t}$$

$$F(t) = 1 - e^{-\lambda t}$$

$$R(t) = e^{-\lambda t}$$

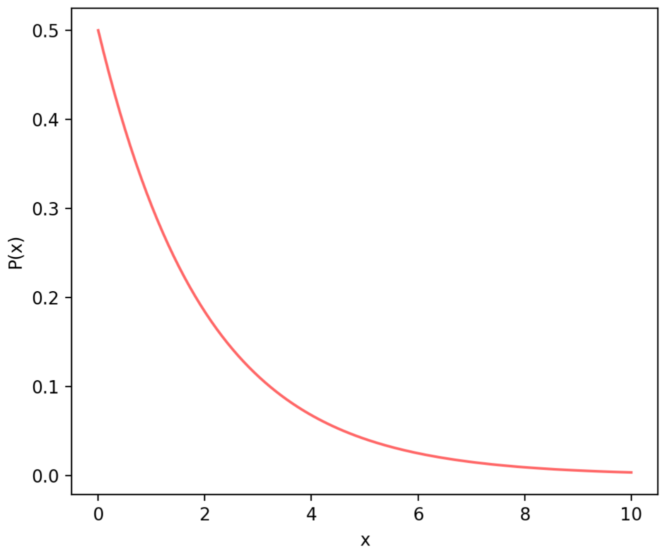


Figure 2.4: Probability density function for exponential distribution

Figure 2.4 shows the probability density function for an exponential distribution with a scale parameter of λ = *0.5*. Figure 2.5 shows the cumulative distribution function for the same distribution. As we can see the PDF value at t = *0* is the value of the scale parameter. The smaller the scale parameter the more the distribution is stretched out along the horizontal axis.

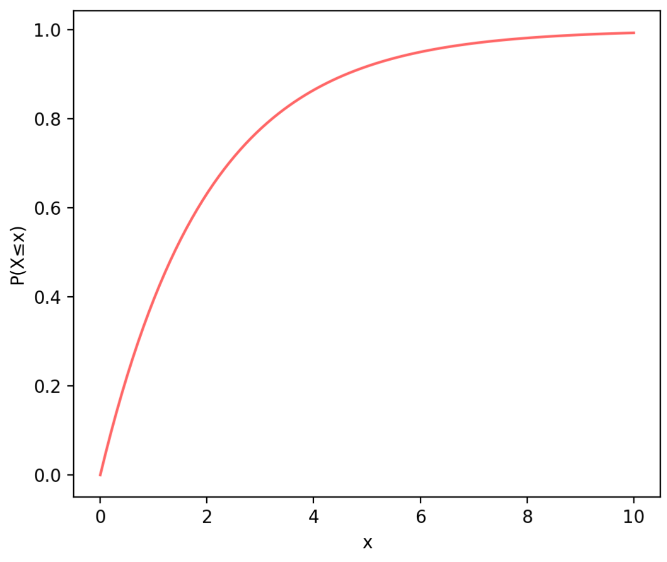


Figure 2.5: Cumulative distribution function for exponential distribution

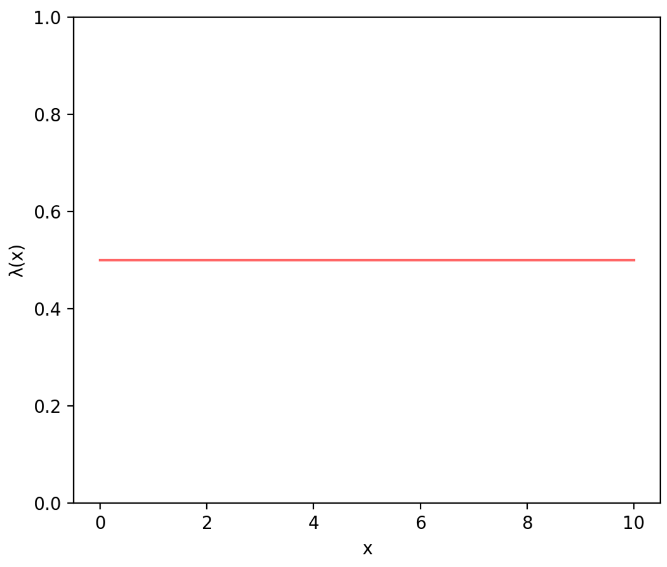


Figure 2.6: Failure rate function for exponential distribution

2.2.2.7 Normal Distribution

The normal distribution is commonly used for reliability analysis usually to model simple mechanical or electronic components. The shape of the normal distribution is already given as a bell shape so it does not have a shape parameter. However, it has a location and scale parameter. The location parameter is denoted by μ and is equal to the mean time to failure. The scale parameter for the normal distribution is denoted by σ which is the standard deviation of the times-to-failure.

The normal distribution is sometimes deemed inappropriate to model reliability since the left-hand limit of the distribution is negative infinity, which can result in negative times-to-failure. However, if the mean is high enough and the standard deviation is small enough the probability of having negative values is infinitesimally small [11, p. 155]. By the 68-95-99.7% Rule, 99.7% of the times-to-failure fall within three standard deviations of the mean. [14]

Figure 2.7 displays the the probability density function for a normal distribution with a $\mu = 5$ and a $\sigma = 1$. As we can see the domain of the distribution is from negative infinity to positive infinity, due to this, there is a slight probability of producing negative times-to-failure with the normal distribution.

Figure 2.8 shows the cumulative distribution for the normal distribution ($\mu = 5$, $\sigma = 1$). The equations for the PDF and CDF of the normal distribution is not shown due to its complexity and since it is not needed for the scope of the thesis to compute it analytically. Figure 2.9 shows the failure rate for the same distribution function. We can tell from the plot that the failure rate stays relatively constant, but then increases dramatically. This shows that the failure rate is dependent on time and the longer a component has been running more failures are likely to occur.

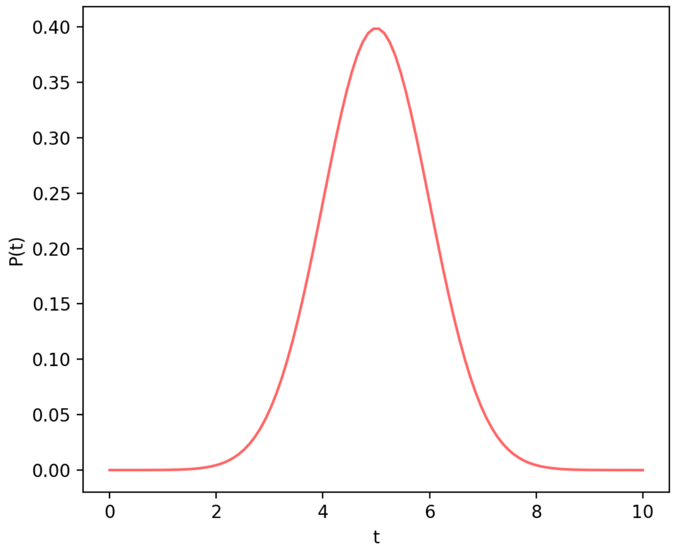


Figure 2.7: Probability density function of normal distribution

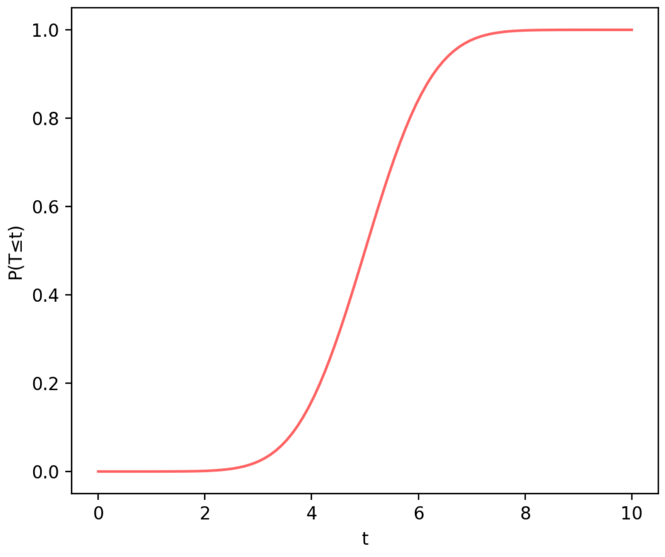


Figure 2.8: Cumulative distribution function of normal distribution

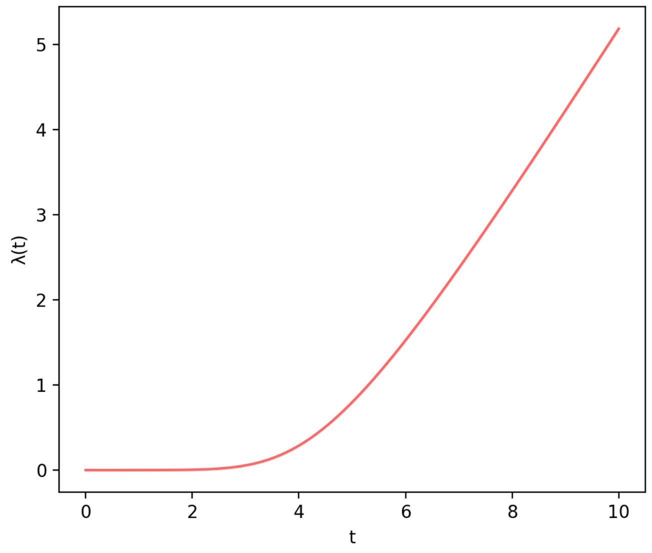


Figure 2.9: Failure rate of normal distribution

2.2.2.8 Lognormal Distribution

The lognormal distribution is also widely used in reliability, usually to model components of fatigue-stress nature such as the strength of materials. A distribution is considered a lognormal distribution when the natural logarithms of the times-to-failure follow a normal distribution.

Two parameters are used to characterize a lognormal distribution, the shape parameter and the scale parameter. The shape parameter is the denoted by σ' and the scale parameter is denoted by μ'. Note the differences between the parameters for the normal and lognormal distributions.

- σ': The standard deviation of the natural logarithm of the times-to-failure.
- μ': The mean of the natural logarithm of the times-to-failure.

The method of calculating the mean is a bit more complex for the lognormal distribution compared to the exponential and normal distributions. Equation 2.10 shows how to calculate the mean using the parameters of the lognormal distribution.

$$m = e^{\mu' + \frac{1}{2}\sigma'} \tag{2.10}$$

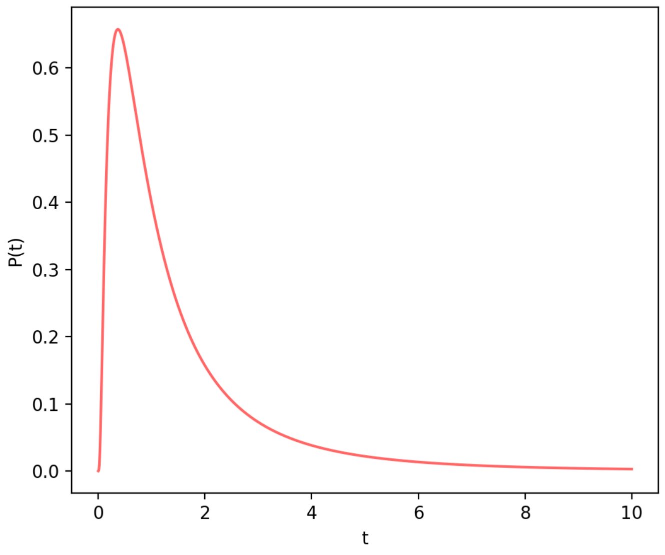


Figure 2.10: Probability density function of lognormal distribution

The probability density function of a lognormal distribution (μ' = 0, σ' = 1) is shown on Figure 2.10. The cumulative density function of the same lognormal distribution is shown on Figure 2.11. And Figure 2.12 shows the failure rate for the lognormal distribution. The failure rate increases rapidly at first and decreases gradually after reaching its peak.

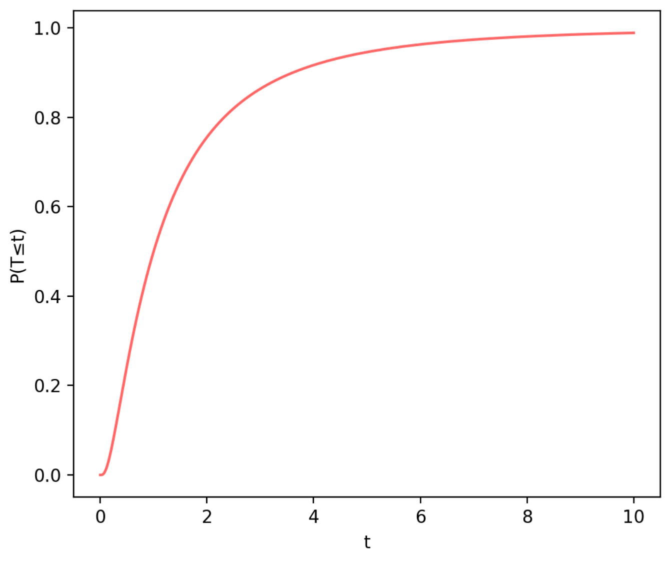


Figure 2.11: Cumulative distribution function of lognormal distribution

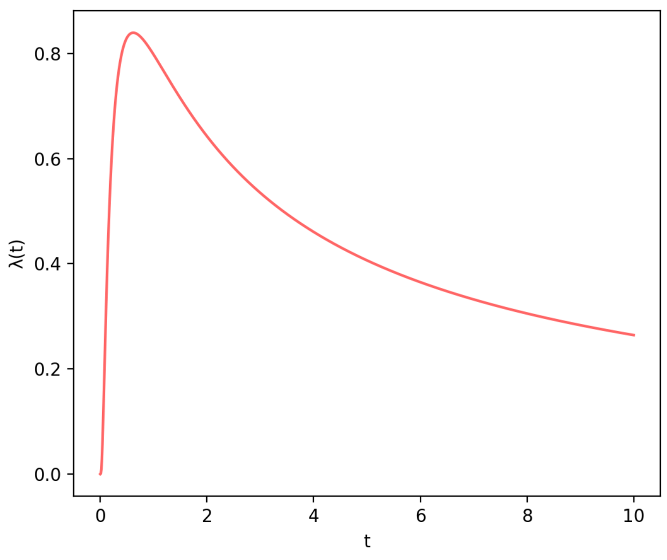


Figure 2.12: Failure rate of lognormal distribution

2.2.2.9 Weibull Distribution

The weibull distribution is extensively used in modeling reliability of components. The weibull distribution is a versatile lifetime distribution because of its shape parameter, since it can take up the shape of other distributions. Due to this property it is widely used to model both electrical and mechanical components and also the strength of materials.

In its most general form, the weibull distribution has all three parameters. However, we will only focus on its shape and scale parameters, since we keep the location parameter at zero. The scale parameter, denoted by α, is responsible for stretching the distribution along the horizontal axis. (Note: The scale parameter for weibull distribution can be indicated with other symbols beside α. Other commonly used symbols in other sources are λ and η.)

The more interesting parameter for the weibull distribution is the shape parameter, denoted by a β. Because of the shape parameter the weibull distribution can resemble other distributions. In fact, if $\beta = 1$, then the weibull distribution is reduced to an exponential distribution.

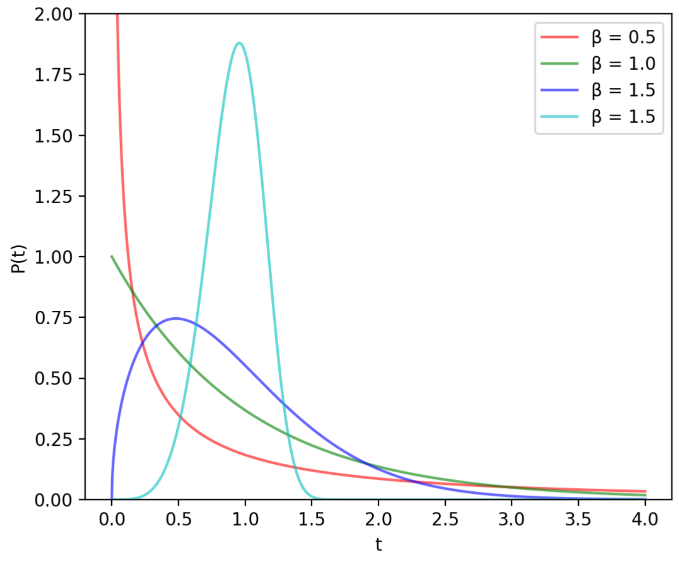


Figure 2.13: Probability density function of weibull distribution

Figure 2.13 shows the probability density function of four different weibull distributions each with a different shape parameter. If we look closely at the weibull distribution with a β = 1, it is exactly the shape of an exponential distribution. Figure 2.14 and Figure 2.15 show the cumulative distribution function and hazard function, respectively, of the four different weibull distributions.

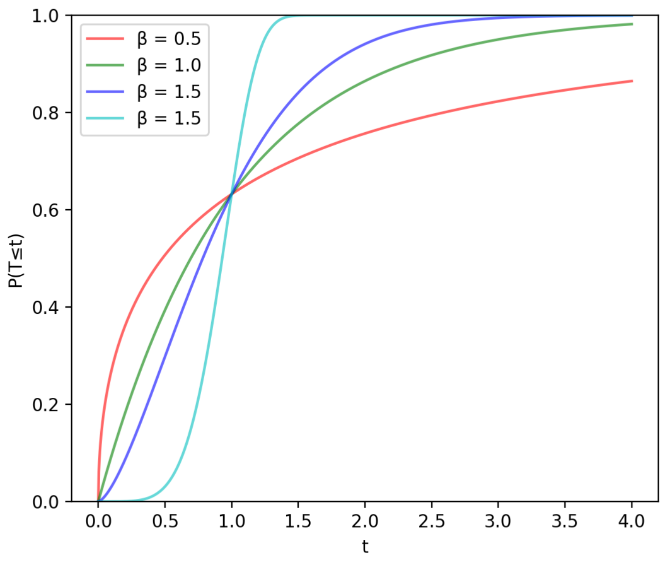


Figure 2.14: Cumulative distribution function of weibull distribution

Out of the lifetime distribution we cover, the weibull has to most complex method of calculating the mean. Before looking at the equation to calculate the mean of a weibull distribution, we must cover a function that is needed to calculate it, the gamma function. The gamma function (see Equation 2.12), denoted Γ*(n)*, is an extension of the factorial function (see Equation 2.11) to real and complex numbers. Since factorials can only be calculated for integers, the gamma function is needed when the argument is a decimal number.

$$\Gamma(n) = (n-1)! \tag{2.11}$$

$$\Gamma(n) = \int_0^\infty x^{n-1} e^{-x} dx \tag{2.12}$$

Equation 2.13 shows to how calculate the mean of the weibull function[11, p. 104]. Note that this equation is the simplified version for the two-parameter weibull distribution.

$$m = \alpha \cdot \Gamma(\frac{1}{\beta} + 1) \tag{2.13}$$

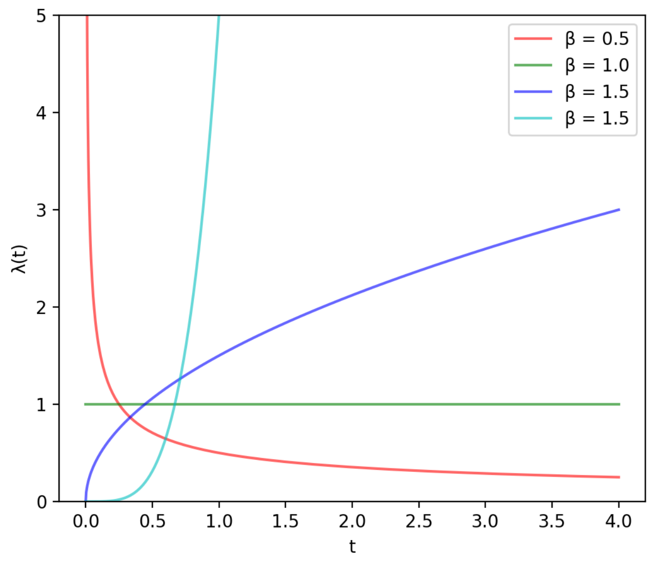


Figure 2.15: Failure rate of weibull distribution

2.3 Fault Tree Analysis

Fault tree analysis is a widely adopted method to analyze the risk and safety of systems in many different industries such as the automotive, aviation, and energy industry. Fault tree analysis is a graphical method of describing the behavior of a system and how the basic components affect the overall system [15]. It shows what combination of basic events can lead to a system failure. A fault tree is visualized as a tree data structure where the leaves are the basic events and the root represent the top event or the entire system. The gates in the fault tree represent the propagation of failure through the tree [1].

The behavior of a basic component is represented by a basic event. A basic event represents a failure which affects the state of the basic component. The events propagate through the fault tree and cause the top event to occur. The top event is the system hazard, which is the subject of analysis [16]. If the top event occurs it means there is a system failure. Therefore, the fault tree represents the cause-effect relationship of the events or faults in a system.

There are two different analysis techniques when discussing fault trees, qualitative analysis and quantitative analysis. Qualitative analysis considers the structure of the fault tree, while the quantitative analysis computes failure probabilities, reliability, etc. of the fault tree. We look at the basic elements in a fault tree and discuss the mentioned analysis in the sections below.

2.3.1 Structure

The elements in a fault tree can be divided into two categories: events and gates. Events represent the failure of components in a system. There are three types of events: basic events, intermediate event, and the top event. Basic events are the failures of the basic component in the system. W omit anything below the level of detail of basic events. The basic events are assumed to be independent events that effect the overall system.

Intermediate events are theoretical events that are caused by a combination of basic events. They make the fault tree easier to comprehend visually but do not affect the analysis of the fault tree [1].

The top event represents the failure of the entire system. There is only one top event in a fault tree and if this event occurs it means the system fails. The reason for fault tree analysis is to determine when the top event occurs and which basic events caused the occurrence.

Figure 2.16 shows an example of a fault tree. It contains the three different gates and the three different events. As the figure shows there is only one top event and the basic events are the nodes of the tree. The events are indicated by a blue rectangle while the gates are indicated by a red polygon.

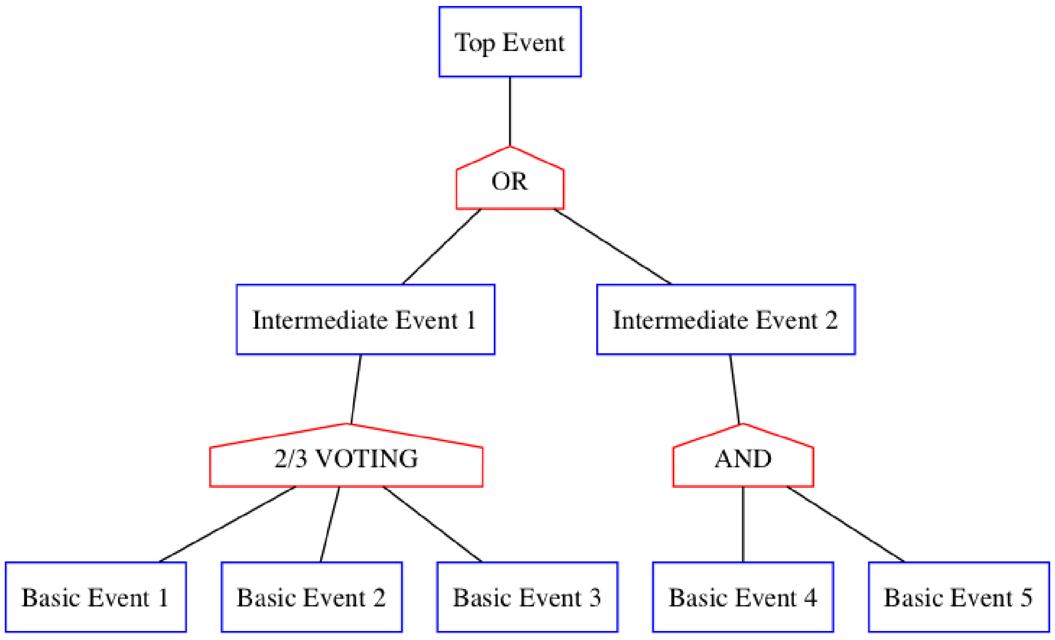


Figure 2.16: Example of a fault tree

2.3.2 Quantitative Analysis

Let us look at how we can apply the probability theory we covered and calculate the reliability of a system using fault tree analysis. In this section we look at the elements in the fault tree, more specifically the gates, and see how the quantitative analysis is conducted by propagating the failures through the gates. There are three main logic gates used in fault tree analysis, *AND*, *OR*, and k/N *VOTING*. The logic gates can have numerous inputs but only one output. The output is calculated using the probability properties we learned in section 2.2.1.

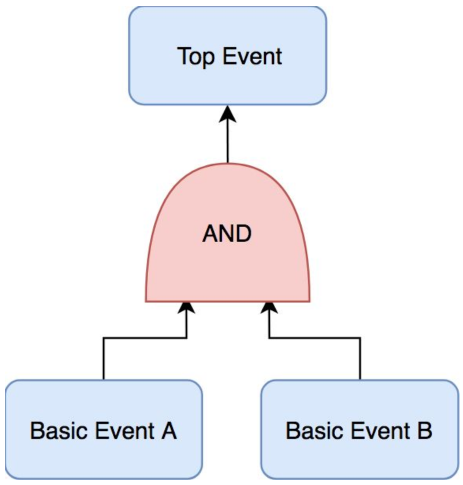


Figure 2.17: System with *AND* gate

2.3.2.1 *AND* Gate

In fault tree analysis, the output event of the *AND* gate occurs when all the inputs of the gate occur [10, p. 211]. This means for the output component to fail, all the input components must fail. We consider all basic events in fault trees to be independent events. In this case, we use Equation 2.5 to calculate the probability of the top event occurring [17]. Equation 2.14 shows how to calculate the probability of failure for the top event in a system with only one *AND* gate (see Figure 3.6).

$$P_{TE} = P_A \cdot P_B \tag{2.14}$$

We know from Equation 2.1 that the reliability is one minus the probability of failure for one component. We apply this knowledge to calculate the reliability of a system with two basic components [10, p. 212].

$$R_{TE} = 1-(1-R_A)(1-R_B) = R_A + R_B - R_A \cdot R_B \tag{2.15}$$

If there are more than two inputs to the *AND* gate then Equation 2.14 and 2.15 can be generalized and written as Equation 2.16 and 2.17, respectively. Where N is the number of components in the system.

$$P_{sys} = \prod_{i=1}^{N}(P_i) \tag{2.16}$$

$$R_{sys} = 1 - \prod_{i=1}^{N}(1 - R_i) \tag{2.17}$$

2.3.2.2 *OR* Gate

The output event of an *OR* gate occurs when at least one of the input events happen. So if any of the input components fail then the output component of the *OR* gate will also fail. Intuitively one feels the probability of failure of the output component must be greater than any of the probability of failures for the input components. Since a failure of any of the input components will cause the failure of the output component. In case of an *OR* gate the probability of failure is calculated as seen in Equation 2.18.

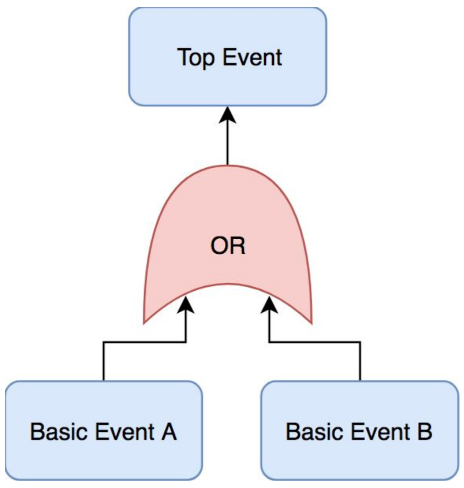


Figure 2.18: System with *OR* gate

One might notice the inverse relationship between calculating probability of failure and reliability for an output of an *AND* gate and an *OR* gate. Thus, calculating the reliability for the output component of an *OR* gate is similar to calculating the probability of failure of the output component of an *AND* gate. Equation 2.19 shows the formulate for calculating the reliability for the output component of an *OR* gate.

$$P_{TE} = 1 - (1 - P_A)(1 - P_B) \tag{2.18}$$

$$R_{TE} = R_A \cdot R_B \tag{2.19}$$

The equations for calculating reliability and probability of failure for the outputs of *OR* gates have general forms, where there can be several input components. Equation 2.20 is the general equation for calculating probability of failure with N components, while Equation 2.21 is the general equation for calculating the reliability.

$$P_{sys} = 1 - \prod_{i=1}^{N}(1 - P_i) \tag{2.20}$$

$$R_{sys} = \prod_{i=1}^{N}(R_i) \tag{2.21}$$

2.3.2.3 *k/N VOTING* Gate

The k/N *VOTING* gate has N number of inputs and for the output event to occur, at least k out of N input events must occur. Figure 3.10 shows an example of a *2/3 VOTING* gate. Here, at least *2* out of the *3* basic components must fail for the top event to occur.

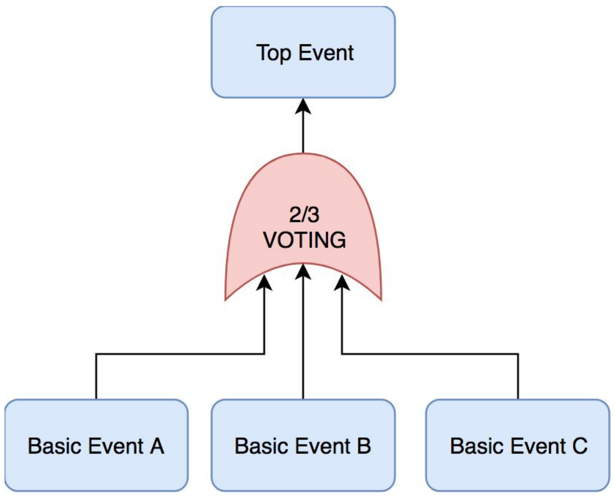


Figure 2.19: System with k/N *VOTING* gate

The k/N *VOTING* gate has a special property, since with certain values of k it is reduced to the other gates. If $k = 1$, then the k/N *VOTING* behaves as an *OR* gate. If $k = N$, then the k/N *VOTING* behaves as an *AND* gate.

The method of calculating the reliability and probability of failure for the output component of a k/N *VOTING* gate is quite complex, especially for arbitrary values of k and N. In case of k = *2* and N = *3* Equation 2.22 is used to calculate the reliability [10, p. 216].

$$R_{sys} = -2 \cdot R_A \cdot R_B \cdot R_C + \cdot R_A \cdot R_B + \cdot R_A \cdot R_C + \cdot R_B \cdot R_C \tag{2.22}$$

We look at a code snippet (see Listing 1) which calculates the probability of the output event in an arbitrary k/N *VOTING* gate. The algorithm is from a pseudocode found in the survey paper written by Ruijters et al. [1, p. 38].

The *k_N_voting* method takes three arguments, k, N, and *probabilities*. Arguments k and N are self-explanatory and *probabilities* is a list of the input probabilities. The algorithm uses recursion to calculate the output probability. First it checks if k = *1*, if it is it returns *1*. If k = N, it returns the output probability of an *AND* gate with *probabilities* as the input probabilities.

Line 5 is essentially Equation 2.16. If k is neither equal to 0 or equal to N, then the algorithm enters recursion process to calculate the output probability.

```
1  def k_N_voting(k, N, probabilities):
2      if k == 0:
3          return 1
4      if k == N:
5          return reduce(lambda x, y: x*y, probabilities)
6
7      return probabilities[0] * k_N_voting(k - 1, N - 1, probabilities[1:])
8      + (1 - probabilities[0]) * k_N_voting(k, N - 1, probabilities[1:])
```

Listing 1: Algorithm for calculating the output probability of a k/N VOTING gate

2.3.2.4 Reliability Propagation through Fault Trees

We have discussed how to calculate the probability of failure and reliability when we have one of the three gates in a system. However what if there is more complex system with many gates, like the one seen on Figure 2.16? This is when we have to propagate the reliability or probability of failure starting from the leaves of the fault tree all the way to the root.

The idea is to start from the basic events and calculate the output event probability for the gates which have only basic events as input. Then the output events of those gates will be the input of the next gates on the next level. We do these calculations until we reach the root of the fault tree, the top event. The reliability of the entire system is calculated in this manner.

We demonstrate this process through an example where we calculate the reliability of the system represented by the fault tree in Figure 2.16. Below is a list indicating the reliability of each basic component.

- R_{BE1} = 0.6
- R_{BE2} = 0.5
- R_{BE3} = 0.8
- R_{BE4} = 0.7
- R_{BE5} = 0.9

Basic Event 1, 2, and 3 are connected by a *2/3 VOTING gate*. Using Equation 2.22 we calculate the reliability of *Intermediate Event 1* to be *0.7*. *Basic Event 4 and 5* are connect by a *AND gate*. Using Equation 2.15 we calculate the reliability of *Intermediate Event 2* to be *0.97*. Now we know the reliability of all the intermediate events, so we can calculate the reliability of the top event. *Intermediate Event 1 and 2* are connected with an *OR gate*. Using Equation 2.19, we calculate the reliability of the *Top Event* to equal *0.679*.

If we also want to know the probability of failure of the system we just use Equation 2.1 and calculate that the probability of failure in this case is equal to *0.321*. Therefore we only have to calculate either reliability or probability of failure using the fault tree, since the other can easily be calculated at the end.

2.3.3 Qualitative Analysis

Qualitative analysis is preformed on fault trees to analyze the structure of the fault tree and check for system vulnerabilities. Some qualitative techniques are cut sets, minimal cut sets, path sets, minimal path sets, and common cause failures [1, p. 34]. We focus on cut sets and minimal cut sets since that is technique used later in the thesis.

2.3.3.1 Minimal Cut Sets

Cut sets and minimal cut sets (MCS) help us determine what events cause system failures. It shows the vulnerabilities of the system [18]. A cut set is a set of events that cause top event to occur. We take a look at a simple fault tree and determine the cut sets for it.

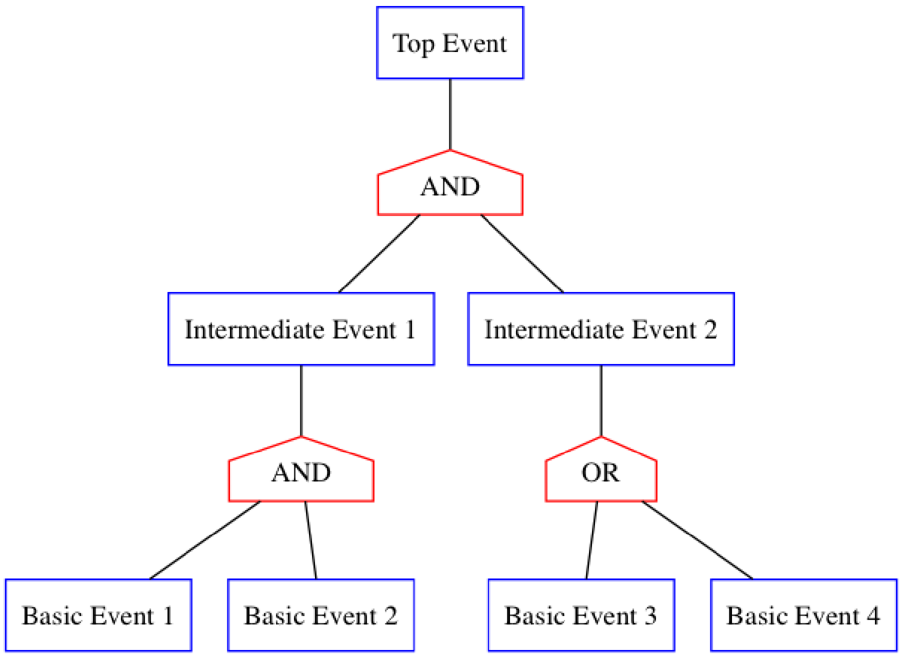


Figure 2.20: Fault tree example for calculating minimal cut sets

Figure 2.20 shows a simple fault tree. By analyzing the fault tree we can tell that for the *Top Event* to occur both intermediate events have to occur. For *Intermediate Event 1* to occur, both *Basic Event 1* and *Basic Event 2* have to occur. For *Intermediate Event 2* to occur, either *Basic Event 3* or *Basic Event 3* has to occur. From these statements, we can conclude that for the *Top Event* to occur *Basic Event 1, 2, and 3* has to fail, this is one cut set. Another cut set is when *Basic Event 1, 2, and 4* cause the system to fail. The system will also fail if *Basic Event 1, 2, 3, and 4* occur, so this is yet another cut set of the system.

We calculated the cut sets for this fault tree now we determine the minimal cut sets. A minimal cut set is a cut set which has no subset that is a cut set. This essentially means the minimal cut set cannot be reduced to a smaller set which will cause a system failure. A minimal cut set has a minimal number of events that will cause the system to fail.

In the example we previously looked at, we have the following cut sets: *(1, 2, 3) (1, 2, 4), (1, 2, 3, 4)*. Out of these cut sets, one cut set can be reduced to another cut set. Cut set: *(1, 2, 3, 4)* can be reduced to either *(1, 2, 3)* or *(1, 2, 4)*. Therefore, the minimal cut sets are: *(1, 2, 3), (1, 2, 4)*.

- Cut Sets: *(1, 2, 3) (1, 2, 4) (1, 2, 3, 4)*
- Minimal Cut Sets: *(1, 2, 3) (1, 2, 4)*

Usually in a simple fault tree, such as this one, the cut sets can be determined visually, but this is not the case for complex fault trees. For more complex fault trees we have to use methods such as boolean manipulation or binary decision trees [1, p. 35].

2.3.4 System Reliability Analysis

We looked out how fault trees are used to calculate the reliability of a system and we covered the basics of discrete random variables and continuous random variables. Now we see how this knowledge is applied to analyzing the reliability of the system depending on different circumstances.

We look out the ways we classify reliability, a system, and the technique used to calculate the reliability of a system. There are two ways to look at reliability, as a static value and as a function of time. A system can be classified into two categories a non-repairable system and a repairable system. And there are two ways to calculate the reliability of a system, analytically and through simulation.

2.3.4.1 Static Reliability

So far we have learned how to use fault trees to calculate the reliability of a system. We have viewed the reliability of components and systems as a static value. The static value can be thought of as the reliability of a component at a given time or the reliability of a component which does not depend on time [10, p. 8].

2.3.4.2 Time-Dependent Reliability

In real life, the reliability of component and systems are not static, but dependent on time. Therefore we must use the continuous probability distribution we learned about in Section 2.2.2 to model reliability distributions of the components in a system. The distributions we model with continuous probability distributions are actually the failure distribution, which indicates the distribution of times-to-failure. The reliability distribution is one minus the CDF of the failure distribution. But in the future, when we discuss the reliability distribution it means the distributions which represent the times-to-failure of a component.

The time units are arbitrary time units and can be seconds, minutes, days, etc. However in future examples in the thesis, we use a general time unit that can represent any time unit. It is important to note that when using a time unit such as hours, you must stay consistent throughout the analysis and use only hours in order to have correct results [10, p. 9].

We can use fault tree analysis to calculate the reliability of time-dependent components in the same manner as we calculate the reliability of static components. But instead of propagating static reliability values through the fault tree we propagate time-dependent reliability functions, so the resulting reliability of the system is a function of time.

2.3.4.3 Non-Repairable Systems

Non-repairable systems are systems with components that are not repaired or replaced after failure. Since we have only looked at reliability so far, the systems we have investigated we assumed to be non-repairable systems. Non-repairable systems only have one distribution, the reliability distribution indicating the times-to-failure. Equation 2.23 is the reliability for a component with a exponential distribution of times-to-failure.

$$R(t) = e^{-\lambda t} \tag{2.23}$$

Mean Time to Failure The mean time to failure (MTTF) is the average time it takes for a component or system failure to happen. In case of a reliability distribution with a constant failure rate the MTTF can be calculated using Equation 2.24. For the other distributions we have covered in Section 2.2.2, we can calculate the MTTF by using their respective equation for the mean value of the distribution.

$$MTTF = \frac{1}{\lambda} \tag{2.24}$$

2.3.4.4 Repairable Systems

Repairable systems are systems where the component are repaired or replaced after failure. Repairing components also take time, so we introduce time-to-repair, the time it takes for a component to be repaired and brought back into an operating state. The distribution of the times-to-repair is called the maintainability distribution. In a similar manner as the distribution of the times-to-failure is called the reliability distribution. Equation 2.25 is the maintainability for a component which has a exponential distribution of times-to-repair.

$$M(t) = 1 - e^{-\mu t} \tag{2.25}$$

Note the differences between Equation 2.23 and 2.25. The differences come from the fact that the reliability represents the probability that an event will not occur (the component will not fail) while the maintainability represents the probability that an event will occur (the component will be repaired) [10, p. 114]. Because of this the maintainability is analogous with the unreliability (probability of failure) of a component.

The μ in Equation 2.25 is the repair rate. This represent the number of repairs in one unit of time, the same way the failure rate represents the number of failures in one unit of time.

Mean Time to Repair The mean time to repair (MTTR) is the average time it takes for a component or system to be repaired. In case of a maintainability distribution with a constant repair rate the MTTR can be calculated using Equation 2.26. Since the MTTR is analogous to the

MTTF, the MTTR can be calculated for other distributions the same way it is possible to calculate the mean value of the distribution.

$$MTTR = \frac{1}{\mu} \tag{2.26}$$

Availability Since in repairable systems we have both reliability and maintainability, we need a new metric that somehow incorporates both of them to calculate the probability that the component or system is operational. This metric is called availability, it is the probability that the component is operating at a given time. For example, if a system has an availability of *99%*, then every *100 hours*, there is *1 hour* when it is not operational. Figure 2.21 shows the uptime (system operational) and downtime (system non-operational) of a system and how it changes with respect to time.

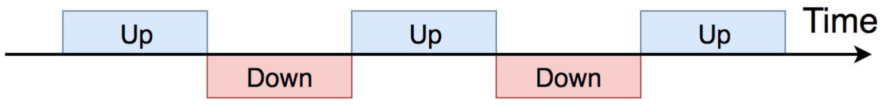


Figure 2.21: Up and down times of a system

There are quite of few classifications of availability, but in our case we look at two; operational availability and instantaneous availability. Operational availability is the ratio of the uptime (the time the system is operational) and the total time measured (see Equation 2.27).

$$A_o = \frac{Uptime}{TotalTime} \tag{2.27}$$

The operational availability is the true availability the customer or user experiences from the system, since it includes all sources of downtime experienced by the customer such as administrative downtime and logistic downtime. Administrative and logistic downtime are waiting downtimes before the repairing of the component begins [10, p. 118].

Instantaneous availability, also called point availability, is the probability that a component is operational with respect to time. Using instantaneous availability, it is possible to determine the probability that a system is operational at any given time. It is similar to the reliability of a system, however instantaneous availability takes into account repairs of a system.

2.3.5 Analytical Analysis

The analytical analysis of calculating the reliability of the system uses algebraic and probability theory methods. Mathematical expressions are determined from the reliability distribution of the components (such as Equation 2.23) and then propagated through the fault tree. Using analytical analysis, the reliability of the system is given as a mathematical expression. The resulting mathematical expression could be rather complex depending on the complexity of the fault tree.

The mean time to failure can also be calculated analytically if the reliability of the system is a mathematical expression. The MTTF can be calculated by integrating the reliability of the system from zero to infinity with respect to time (see Equation 2.28) [10, p. 67].

$$MTTF = \int_0^\infty R_s(t)dt \tag{2.28}$$

Analytical analysis is adequate when considering simple non-repairable systems. However, if the system we are considering to analyze is repairable and the components have maintainability distributions then using analytical analysis is extremely difficult if not impossible [10, p. 8]. In such cases we must use simulation to calculate the reliability, maintainability, and availability of a system.

2.3.6 Simulation

If he have a system with components that have reliability and maintainability information we have to use simulation to analyze the system. Simulation is the act of replicating the behavior of a system. To simulate a system we need a model that describes the behavior of the system, in our case this is the fault tree and the reliability and maintainability distributions of the components. More specifically we use discrete event simulation (DES) to analyze the system.

2.3.6.1 Discrete Event Simulation

Discrete event simulation models the behavior of a system as it changes through time. Each discrete event models the system in one particular state at a given time. After each event the system changes state, but between the events the state of the system does not change [19]. Therefore we disregard what happens between the discrete events since there is no change from the system's point of perspective.

To reproduce the behavior of a system the DES relies on random number generators. The random numbers cause slightly different results after each simulation. Calculating the confidence intervals on a collection of these results provide us, with a certain probability, the theoretical behavior of the system [20, p. 18].

The times-to-failure and times-to-repair are the random numbers generated from the theoretical reliability and maintainability distributions, respectively. Using our fault tree model, we calculate the times-to-failure and times-to-repair of the system to determine the reliability, maintainability, and availability.

However, discrete event simulation has its limitation, mostly due the random nature of the simulation. Since it relies on random number generators the quality of the results depend on the quality of the random number generators. This affects the width of the confidence interval, which needs to be narrow for accurate results. The width of the confidence interval is also affected by the number of replications of the simulation. The entire analysis can become computationally heavy with more replications.

Discrete event simulation is adequate for computing the reliability and maintainability separately and graphing their respective function with respect to time. However, availability can only be given as a percentage and calculated with Equation 2.27. Another method of simulation, called the proxel-based simulation, is needed to plot the instantaneous availability of a system with respect to time.

2.3.6.2 Proxel-Based Simulation

The proxel-based simulation is a numerical method of analyzing discrete stochastic models. It bypasses the need to calculate differential equations by incorporating the instantaneous rate function (failure rate function, hazard function) to calculate the flow of probability at each time step. The proxel-based simulation is an robust simulation method applicable in many applications, however we focus only on the aspects that are related to fault tree analysis. For a deeper understanding of proxel-based simulation refer to the PhD dissertation of Sanja Lazarova-Molnar [20]. This section takes inspiration from her work to explain how the proxel-based simulation can be used for fault tree analysis.

Proxel The term *proxel* is analogous to the word pixel. It is constructed from the words probability and element. A pixel is the smallest element of an image, the same way a proxel is the smallest computational element for a probability [20]. The proxel can be defined with the following information:

- Discrete state: A discrete state of an event
- Age intensity vector: The elapsed time since entering the discrete state
- Global simulation time: The time elapsed from the start of the simulation.
- Route: The sequence of states the model has been in leading up to the current discrete state.
- Probability: The probability of the event being in the current discrete state which is described by the route it took to reach this event and the age intensity since the start of the event.

Formally the proxel has the following format seen in Equation 2.29.

$$Proxel = (Discrete\ State, Age\ Intensity\ Vector, Time, Route, Probability) \tag{2.29}$$

The best way to see how the proxel-based method works is to demonstrate it with the help of an example. The example simulates the state of a component in a fault tree. The component can either be in an *OK* or *FAIL* state, and the probability distribution that describes the state change from *OK* to *FAIL* is the reliability distribution while the maintainability distribution describes the state change from *FAIL* to *OK* (see Figure 2.22).

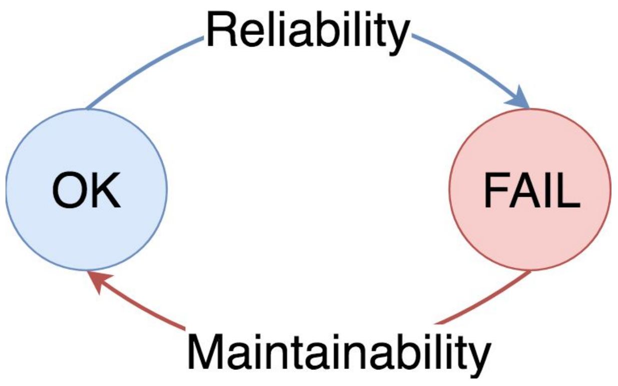


Figure 2.22: State diagram of a component

In our example, let the reliability distribution be an exponential distribution with a $\lambda = \frac{1}{4}$ and the maintainability be an exponential with a $\lambda = \frac{1}{2}$. In both cases the instantaneous rate function is constant therefore it is equal to λ, the failure rate is equal to $\frac{1}{4}$, while the repair rate is equal to $\frac{1}{2}$.

In our example, we omit the time and route property of the formally defined proxel, as seen in Equation 2.29. The reason we omit the time from the proxel, is because we keep track of the simulation time outside of the proxel. We omit the route since it is irrelevant for us to know what sequence of states the model has been in. The proxel in this example is defined as such:

$$Proxel = (State,\ Age\ Intensity\ Vector,\ Probability)$$

The proxel-based method algorithm starts with a single proxel, the initial proxel. Our initial proxel is in the *OK* state with an age intensity vector of *0*, and a probability of *1*. Since our component is guaranteed operating at time *0*.

$$Proxel = (OK,\ 0,\ 1)$$

Before the simulation begins we must agree on a the size of the time step (Δt). The time step is by how much we increment the time after expanding a proxel. Hypothetically, the time step should be small enough to capture every state change that can occur. During the span of one time step, at most one state change can happen. In this example, let $\Delta t = 0.5$.

The way the algorithm works, is that after each time step the current proxel is expanded into as many other proxels as there are valid state changes. From the start, the initial proxel expands to two proxels, one representing the component staying in *OK* state and the other representing a change to *FAIL* state. When a state change occurs the probability of the new proxel is calculated using the Equation 2.30.

$$P = P_{parent} \cdot \int_0^{\Delta t} \mu(x)dx \tag{2.30}$$

Where *μ(x)* is the instantaneous rate function of the distribution leading to this proxel. The P_{parent} is the probability of the parent proxel, where the current proxel was expanded from. Equation 2.30 is often approximated by Equation 2.31, since we assume Δt to be small.

$$P = P_{parent} \cdot \mu(t) \cdot \Delta t \tag{2.31}$$

The initial proxel is expanded to the following:

$$(OK,\ 0.5,\ 0.875) \quad and \quad (FAIL,\ 0,\ 0.125)$$

The second proxel is in the *FAIL* state, therefore the age intensity vector is reset to *0*. The probability is calculated using Equation 2.31, where the P_{parent} = *1*, $\mu(t)$ is constant in our example, and is equal to the failure rate of $\frac{1}{4}$, and finally Δt = *0.5*. This calculation yields a probability of *0.125* for the second proxel.

The probability of the first proxel is simply the probability of the parent proxel minus the probability of the second proxel. From here we see that the probability of the proxel which includes the state change must be computed first in order to calculate the probability of the proxel without the state change.

For each layer in the proxel tree, the proxels are expanded in a similar manner and the expansion of the first three layers of proxels are shown on Figure 2.23. The *Time* column on the left of the figure is increased by the time step as we go deeper in the layers of the proxel tree.

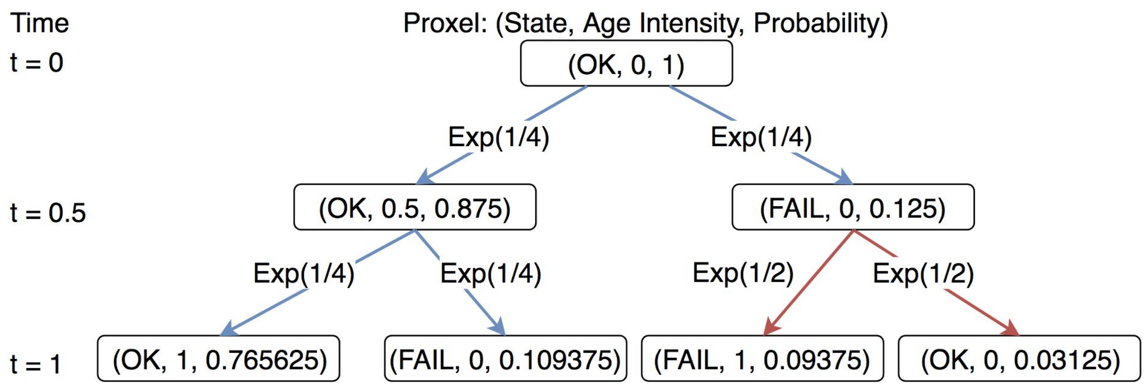


Figure 2.23: Example of proxel-based simulation

The proxel tree expands exponentially at first and depending on the distribution the width of the of proxel tree can reach a maximum width. The width of the proxel tree can be limited by discarding proxels with extremely low probability values (e.g. $P \leq 10^{-12}$). However, this should be done with caution since discarding too many proxels can obscure the results of the simulation.

The proxel tree is expanded until the time reaches the end of the simulation time. Afterwards, on each layer we sum up the probabilities of the proxels belonging to the same state. Table 2.1 shows the calculation for the probability of the states at the first three time steps.

Table 2.1: Probability of states at each time step

Time	Probability of OK	Probability of FAIL
0	1	0
0.5	0.875	0.125
1	0.76525 + 0.03125 = 0.796875	0.109375 + 0.09375 = 0.203125

Starting from the initial proxel, we see that at time *time* = *0*, the probability of the component being in *OK* state is equal to *1*. From this we conclude that the probability of the component being in *FAIL* state is equal to *0*. At *time* = *0.5*, the probabilities are straightforward, there is a *0.875* probability the component is in state *OK* and a *0.125* probability the component is in state *FAIL*. At *time* = *1*, we sum up the probability of the proxels with the same state, as seen in Table 2.1.

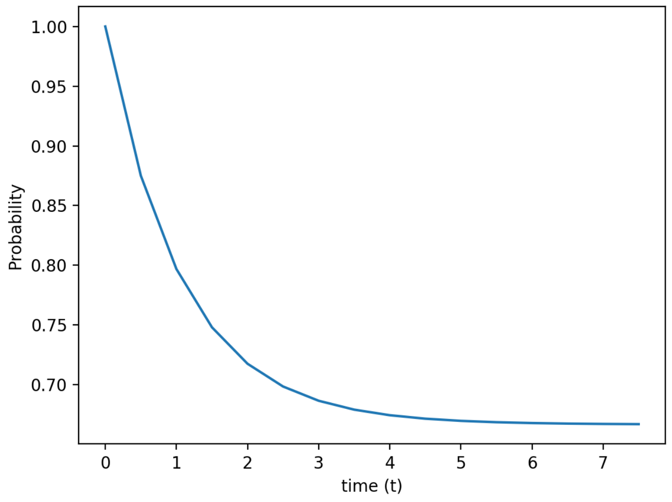


Figure 2.24: Probability of component in OK state (Availability)

By simulating the behavior of the component until *time* = *7*, we get the following plots: Figure 2.24 shows the probability of the component in the *OK* state with respect to time. It is actually the probability that the component is operational at a given time, which is the definition of availability. Hence, this figure displays the instantaneous availability of the component at a given time.

Figure 2.25 shows the probability of the component being in the *FAIL* state with respect to time, or another way to phrase it, it is the probability that the component is non-operational at a given time. It can be considered as the unavailability of the component.

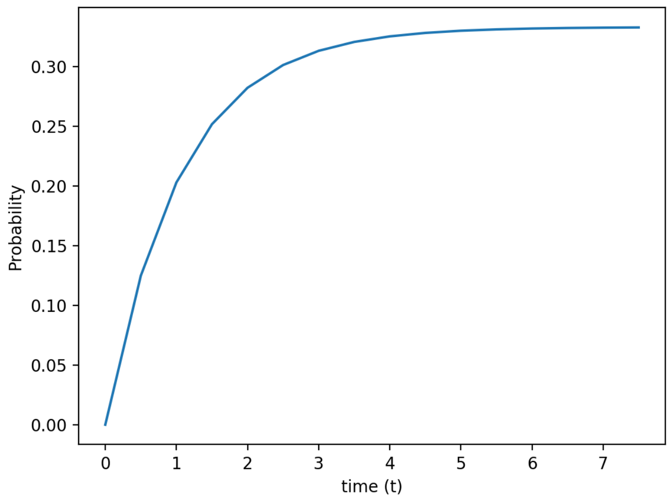


Figure 2.25: Probability of component in FAIL state

What we have looked at so far is calculating the probability that one component is either operation or non-operational. However, we are more curious about how a system behaves. So to calculate this for a system we have to use the same method we learned in Section 2.3.2 and propagate the probabilities of the components through the fault tree to determine the availability or unavailability of the system.

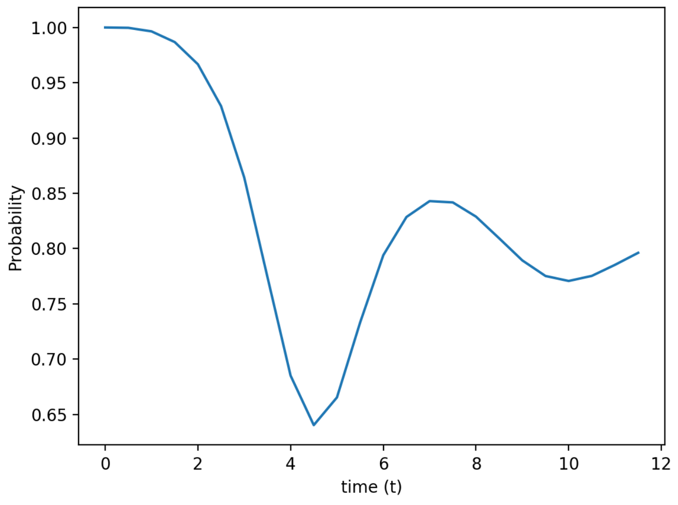


Figure 2.26: Availability of a system

Instantaneous Availability of a System We have discussed instantaneous availability before and using the proxel-based method we are able to calculate it for a system. If we were to simulate the availability of a system using proxel-based simulation it would look something like Figure 2.26. The exact shape of the plot heavily depends on the reliability and maintainability of the basic components, however usually it will first oscillate a few times and then converge to value called the steady-state availability. The availability function typically converges to the steady-steady availability after a time period that is four times the mean time to failure [21]. In Figure 2.26, the availability of a system has not converged to the steady-state availability as it is still unstable.

Chapter 3

Implementation

This chapter describes the implementation of the methods and techniques used in the thesis. I describe the entire process of the fault tree creation, data generation, and fault tree reconstruction. I provide ample examples to describe the algorithms in detail. Figure 3.1 shows an overview of the stages we cover in this chapter.

The first step in the overview which is the fault tree creation and time series generation is necessary in order to provide a framework to assess the research question of the thesis. The data learning has to be done on data which we know the source of to be able to evaluate the data learning techniques by comparing the original fault tree and reconstructed fault tree. The main implementation that assesses the research question is the learning of the time series data, reconstruction of the fault tree, and calculation of reliability and other metrics.

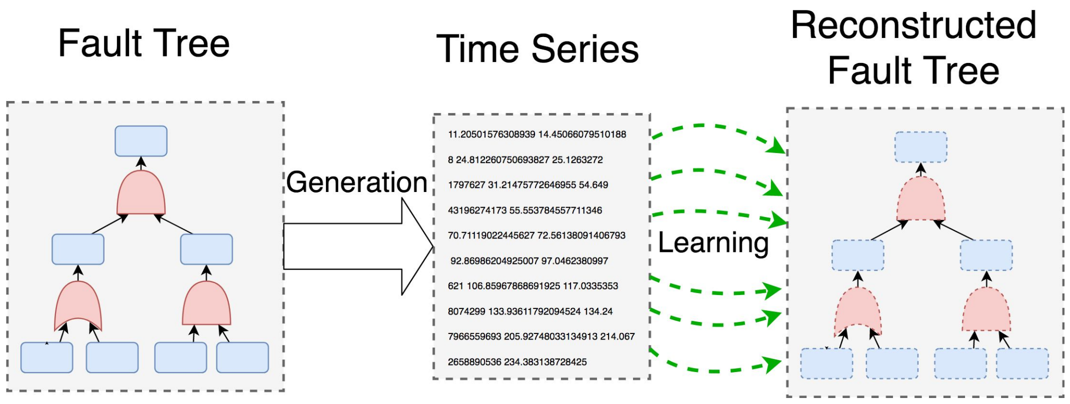


Figure 3.1: Overview of the implementation

3.1 Data Generation

This section explains the process of generating the data. Here, we look at the implementation of the fault tree in *Python*. We look at how the fault tree elements are instantiated and connected to form a proper fault tree. Then the process of generating the time series data from the fault tree is explained.

3.1.1 Tree Structure

The python package called **anytree** is used to create the tree structure of the fault tree [22]. The **anytree** module makes it possible to create nodes, using the **Node** class, which are connected to other nodes in a familiar tree structure manner. Nodes can have parents and children. To create a tree structure in **anytree**, one must create a node which is the root, without any parents. Afterwards, the children nodes are created and connected to the parent. Children are implicitly assigned to a node, since during the creation of a node the parent has to be explicitly given.

```python
from anytree import Node, RenderTree

n1 = Node("4")
n2 = Node("5", parent=n1)
n3 = Number("6", parent=n1)
n4 = Number("7", parent=n2)
n5 = Number("8", parent=n2)

print(RenderTree(n1))
```

Listing 2: Example for creating a tree structure in **anytree**

The resulting tree from the code in Listing 2. is shown in Figure 3.2.

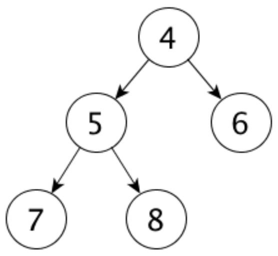


Figure 3.2: Example of a tree

The **anytree** module is responsible for creating the tree structure and it allows for the creation of custom nodes. To create custom nodes, a new class has to be created which inherits the **NodeMixin** class. The new class inherits all the properties of the **NodeMixin** class which makes it possible to connect them into a tree structure. Afterwards custom attributes and methods can be added to the class.

3.1.2 Fault Tree

Building on what we learned from the last section about the **anytree** package and how to make a tree structure, we use it to create a fault tree structure. As we know, fault trees consist of two types of nodes, events and gates. For this purpose, there are two separate classes called the **Event** class and the **Gate** class, and both of them are extended from the **NodeMixin** class.

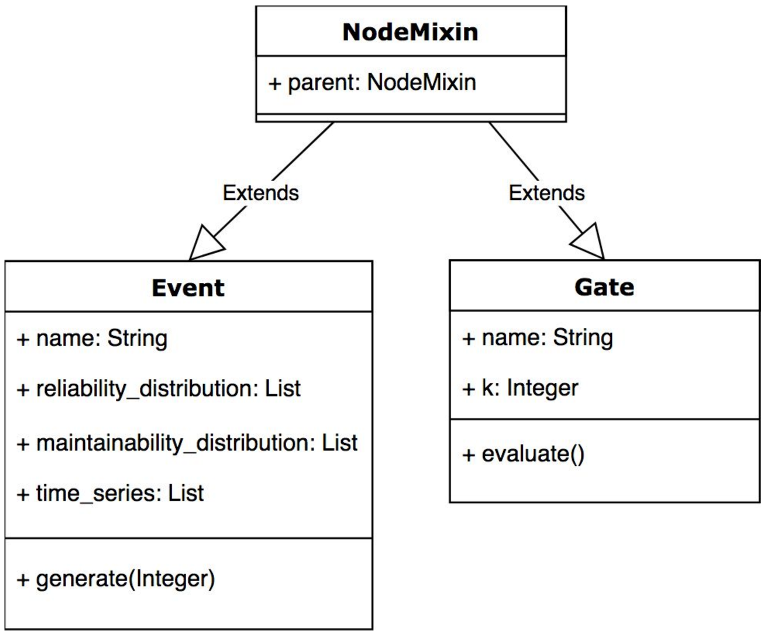


Figure 3.3: NodeMixin extends Event and Gate

Figure 3.3 is a class diagram that shows how the **NodeMixin** class extends the **Event** and **Gate** class. For the sake of simplicity, not all the attributes and methods are shown in the class diagram, only the ones relevant to this section are shown.

3.1.2.1 Event

The **Event** class is used to represent the basic events, the intermediate events, and the top event. Since it inherits the **NodeMixin** class it has to have a *parent* attribute. The rest of the attributes are all specific to represent an event in a fault tree.

The **Event** class has a *name* attribute which should be a unique identifier for that event only. For example, the event's name can be: 'Basic Event 3', 'Intermediate Event 1', or 'Top Event'. There are also two attributes indicating the reliability distribution and maintainability distribution for the event.

3.1.2.2 Gate

The **Gate** class represents the logic gate such as *AND*, *OR*, and *k/N VOTING*. This class also inherits the **NodeMixin** class, so it has to have the *parent* attribute.

The *name* attribute, in case of the gate, is not a unique identifier, but the type of logic gate the **Gate** class represents. Therefore, the values it can have are: 'AND', 'OR', and 'VOTING', to indicate the type of gate. In case of the *k/N VOTING* gate, the *k* value still needs to be given. A separate attribute called *k* is used for this purpose.

3.1.2.3 Fault Tree

The fault tree can be constructed using the **Event** class and the **Gate** class. The instances of these classes are connected by using the *parent* attribute. The *parent* attribute points to the parent of the node (event or gate) in the fault tree.

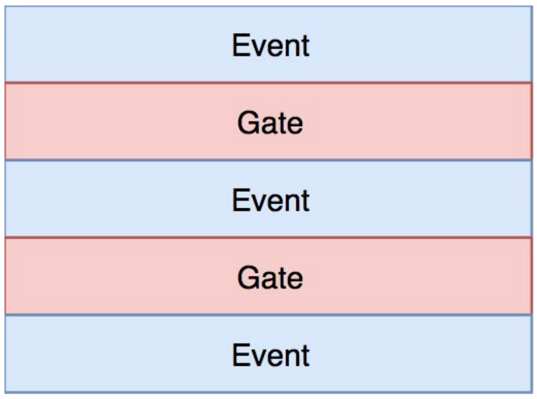


Figure 3.4: Event and Gate connections

Figure 3.4 shows how the events and gates are connected, which are shown as layers. The parent of a gate can only be an event, while a parent of an event can only be a gate. It is also important to note, that a gate can be a parent to more events, however a parent of a gate should only be one event, typically an intermediate event or the top event. This will result in the characteristic tree structure of a fault tree.

By connecting the instances of **Event** class and **Gate** class in such a manner, a resulting fault tree can be created as seen in Figure 3.5. As shown on the figure, the gates have only one parent but two or more children.

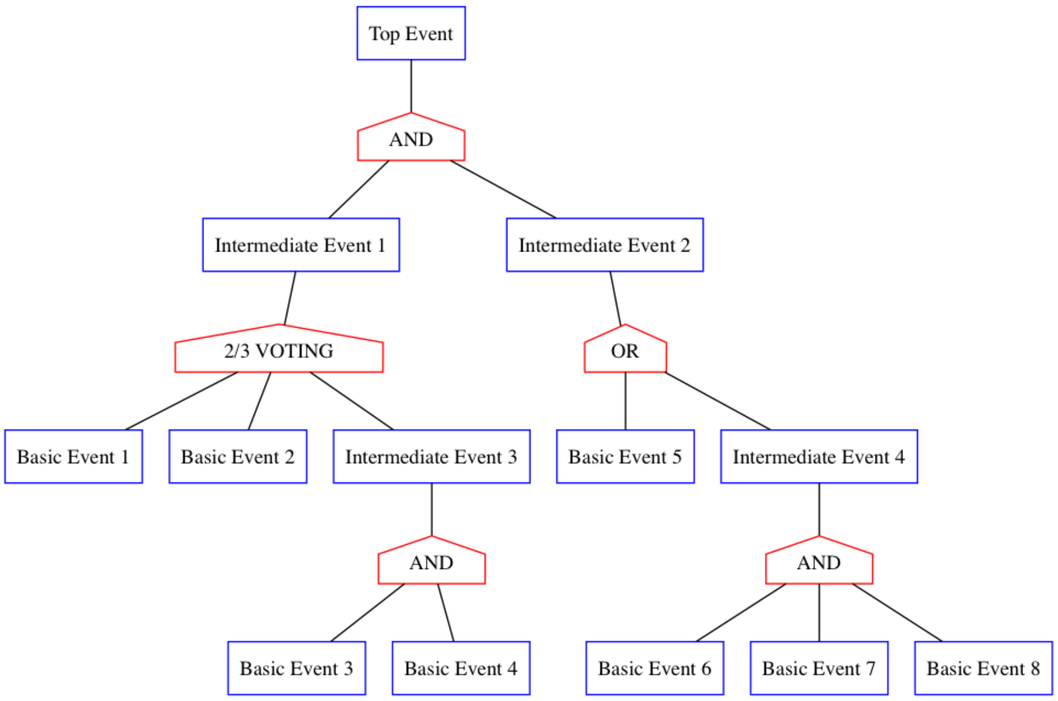


Figure 3.5: Example of a fault tree

3.1.3 Time Series Generation

After designing a fault tree, the next step is to generate the time series data for the basic events and the top event. The section below describes the process of generating the time series data.

3.1.3.1 Algorithm

In this section, the algorithm is explained using a simple example. For the sake of simplicity, only whole numbers are generated using a uniform distribution that generates random numbers from *1* to *10*. The implemented algorithm can handle decimal numbers and other distributions.

We generate two groups of random numbers one representing the numbers generated with the reliability distribution and the other representing the numbers generated with the maintainability distribution. The numbers generated using the reliability distribution are times-to-failure, the time it takes for a failure to occur. Similarly, the numbers generated from the maintainability distribution are the times-to-repair, the time it takes to repair the component.

Table 3.1: Time series generation algorithm

Time to failures	$[t_1, t_3, t_5]$	[2, 4, 3]
Time to repairs	$[t_2, t_4, t_6]$	[5, 8, 1]
Time to occurrences	$[t_1, t_2, t_3, t_4, t_5, t_6]$	[2, 5, 4, 8, 3, 1]
Time series	$[t_1, (t_1 + t_2), (t_1 + t_2 + t_3), (t_1 + ... + t_4), ...]$	[2, 7, 11, 19, 22, 23]

After the times-to-failure and times-to-repair are generated then we merge the two list of numbers into one list. We do this by taking one time stamp from the times-to-failure then one time stamp from the times-to-repair and so on and so forth. A simpler way to look at this is to consider the times-to-failures with odd indexes and the times-to-repair with even indexes and we merge them numerically in increasing order. This results in a list of numbers, however they are not yet time series, just times-to-occurrence (failures and repairs).

In order to make them into a time series, we leave the first number as is, since that is the time of the first failure. Then starting from the second number we always add the previous numbers to that number to make it into a time stamp. The time stamp represents the time when the event occurred starting from *time = 0*. Table 3.1 gives an overview of how the time series are calculated.

3.1.3.2 Distributions

The time series generation algorithm can handle four different probability distributions commonly used in reliability engineering [11, p. 18]. The four distribution are: exponential, weibull (2-parameter), normal, and lognormal.

Table 3.2: Distributions and its Parameters

Distribution	**Parameter(s)**
Exponential	λ (scale)
Weibull	α (scale), β (shape)
Normal	μ (location), σ (scale)
Lognormal	μ (shape), σ (scale)

Table 3.2 shows the distribution which can be used and their respective parameters that are needed to generate random numbers.

3.1.4 Logic Gate Operators

The time series of the top event depends on the configuration of the basic events and the type of gates in the fault tree. In order to determine the time series of the top event, the time series of the basic events must propagate through the gates of the fault tree. At each gate there are inputs of time series and the output time series depends on the type of gate. Below we look at how the output time series are calculated for each type of gate.

3.1.4.1 *AND* gate

The output event of an *AND* is a occurs if all the input events occur. From the time series of the basic events we also know the state of the basic components. Since we know the first occurrence in the time series is a time-of-failure, we know that up to that point the component is in a *OK* state. When the first failure happens than the component enters the *FAIL* state and it stays in this state until the next occurrence, which is the time-of-repair.

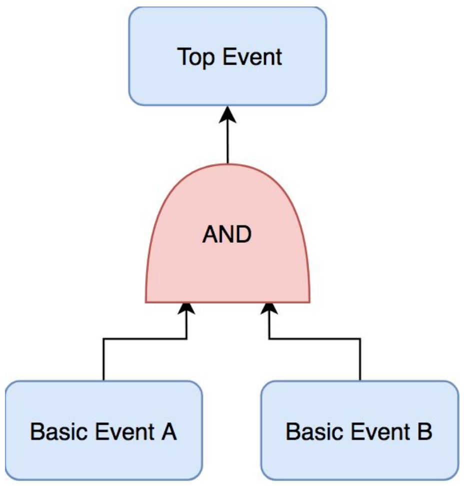


Figure 3.6: System with *AND* gate

Let us look at a simple example with two basic events and an *AND* gate as seen in Figure 3.6. The method of calculating the time series for the top event is shown in Figure 3.7. *Basic Event 1* fails at *time* = *2* and is repaired at *time* = *5*. *Basic Event 2* fails at *time* = *3* and is repaired at *time* = *6*. We know from Figure 3.6 that *Basic Event 1* and *Basic Event 2* are connected by an *AND* gate. Therefore, both these events have to fail for the *Top Event* to fail.

From the time series diagram on Figure 3.7, we see that by *time* = *3* both basic events have failed, therefore the *Top Event* fails as well. At *time* = *5*, *Basic Event 1* is repaired so the *AND* gate's requirement of having all the input events in the *FAIL* state is no longer satisfied. Therefore, the *Top Event* is repaired as well, since not all the input basic events are in the *FAIL* state.

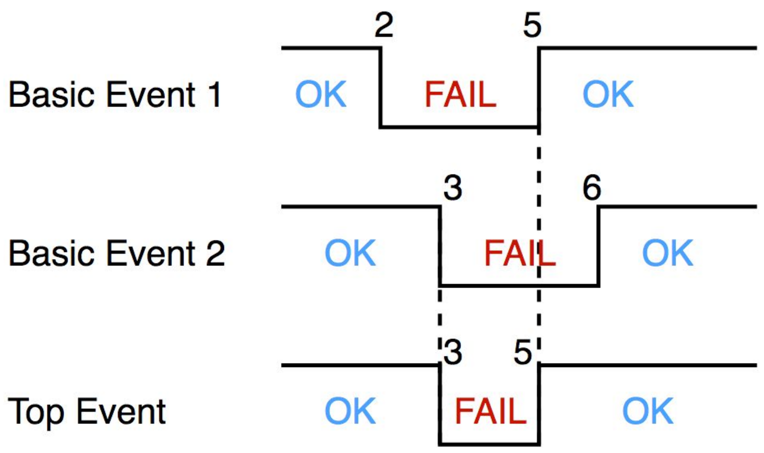


Figure 3.7: Time series calculation for output of *AND* gate

Note, Figure 3.7 is similar to digital timing diagram commonly seen in digital electronics which also incorporate logic gates such as *AND* and *OR*. However, the resulting diagrams are different for the same type of gate since in fault tree analysis the gate checks for failures while in digital electronics the gate checks for successes. The difference is fault tree analysis works with negative logic in digital electronics the default is usually positive logic [23]. Keep this in mind for the *OR* gate as well.

3.1.4.2 *OR* gate

The output event of an *OR* gate is a failure if any of the input events is a failure. Let us look at the simple fault tree on Figure 3.8 which has only one gate, an *OR* gate.

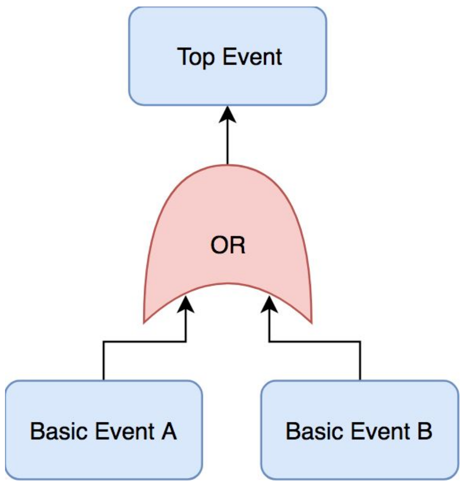


Figure 3.8: System with OR gate

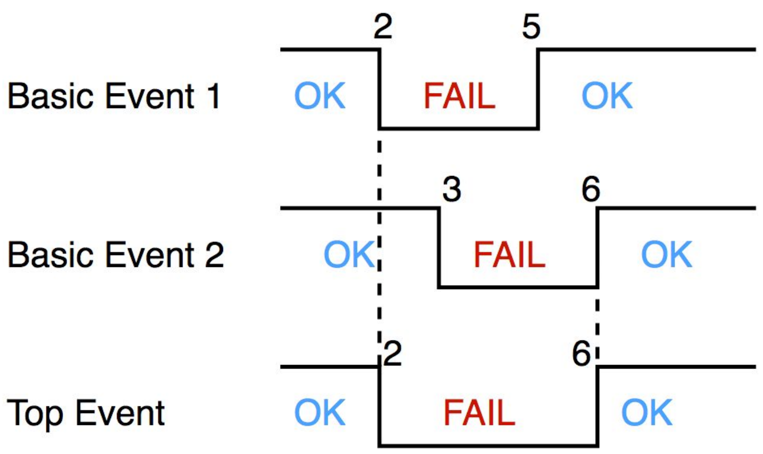


Figure 3.9: Time series calculation for output of *OR* gate

Figure 3.9 shows the method of calculating the time series of the *Top Event* for the fault tree on Figure 3.8. Here, *Basic Event 1* failure causes the *Top Event* to fail at *time* = *2*. The *Top Event* is not repaired until *time* = *6*, which is time by which all the basic events are repaired.

3.1.4.3 *k/N VOTING* gate

The output event of an k/N *VOTING* gate is a failure if at least k out of N input events are failures. We look at an example fault tree below which has a *2/3 VOTING* gate and three basic events. In this case, the top event fails if at least *2* out of *3* of the basic events fail.

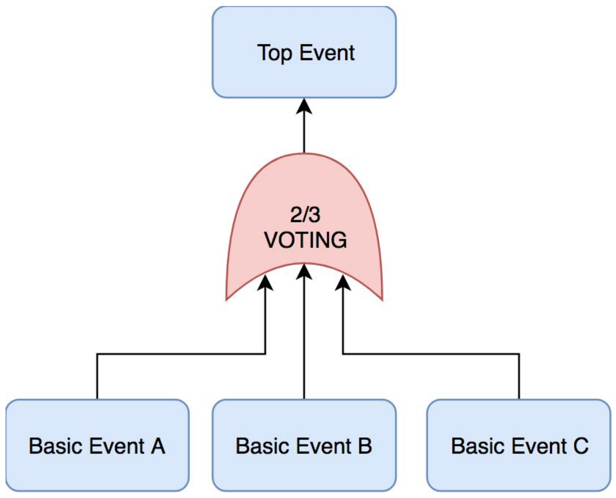


Figure 3.10: System with 2/3 VOTING gate

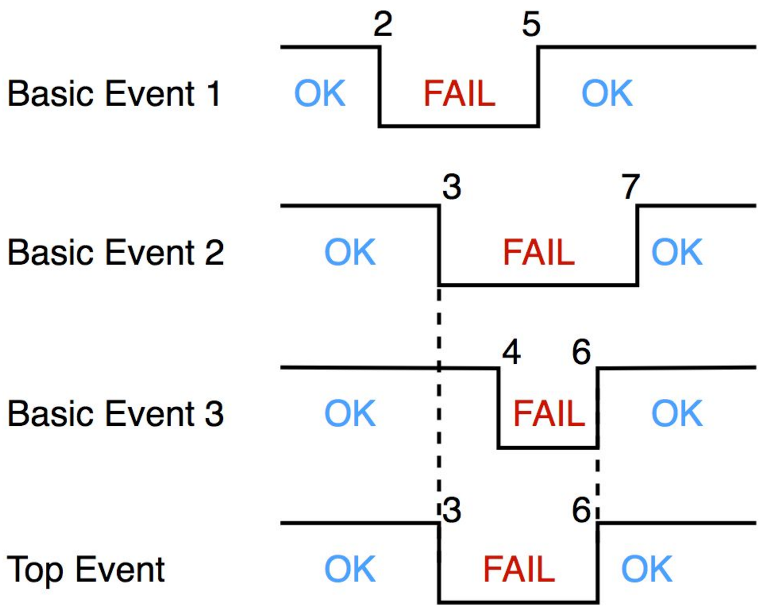


Figure 3.11: Time series calculation for output of k/N *VOTING* gate

Figure 3.11 shows the method of calculating the time series of the top event for the fault tree on Figure 3.10. The *Top Event* fails at *time* = *3* because at this point in time at least *2* of the *3* basic events have failed. The *Top Event* is repaired when there is less than *2* of the *3* basic events that have failed, which as at *time = 6*.

3.1.4.4 Algorithm

We take a look exactly how the algorithm works to determine the output time series. Let us look at the *OR* gate example in Figure 3.9. But before, we go over some abbreviations used in the example.

- BE1: Basic Event 1
- BE2: Basic Event 2
- TE: Top Event
- U: stands for Up, it means component is in *OK* state
- D: stands for Down, it means component is in *FAIL* state

We take the time series of *Basic Event 1*. Then we insert the state of the component in between the transition times (times-of-failure and times-of-repair), let us call this modified time series a data stream. This is done in such away that the transition times are in increasing order with respect to other transition time in the other data stream. If we look at Figure 3.12, at the data stream of *Basic Event 1*, the transition time of *5* has to be after the transition time of *3* which is in the data stream of *Basic Event 2*. The spaces in between are filled out with the state of the component between the transition times to ensure the data streams are the same length and the output data stream can be computed.

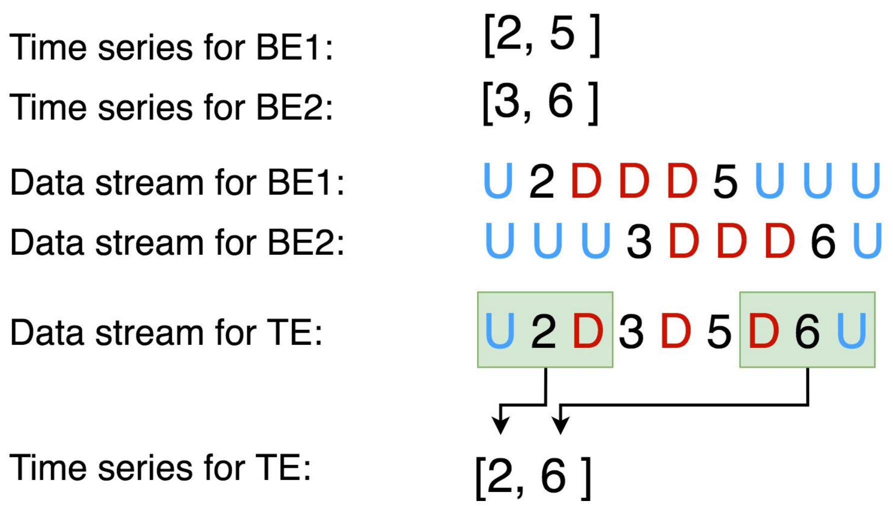


Figure 3.12: Time series calculation algorithm

We compute the data stream for the *Top Event* column-wise. If inside the column, there is only states (either *U* or *D*), then we compute it according to the specification of the gate. Since this is an *OR* gate, if any of the states are *D* then the resulting state is also *D*. If there is a transition time in the column, the result for that column will just be the transition time.

After calculating the data stream for the *Top Event*, we need to decode it to find the time series for the *Top Event*. We look at each transition time in the data stream and check if there is a state change during that time. Since at *time* = *2*, the state changes from *U* to *D*, so this is also a transition time for the *Top Event* and is then appended to the time series of the *Top Event*. At the next transition time, at *time* = *3*, the state remains in *D*, so it not added to the time series. In this manner, all the transition time in the data stream are checked to see if they should be added to the time series of the *Top Event*.

With the help of this algorithm we can calculate the output time series of any of the gate types. The only difference to keep in mind is the calculation of the output state is specific to the type of gate.

3.1.5 Time Series of the Top Event

In the previous section, we learned how to the output time series are calculated for a logic gate. Now, we learn how the time series propagates through a more complex fault tree with more than one gate. Consider the fault tree in Figure 3.13.

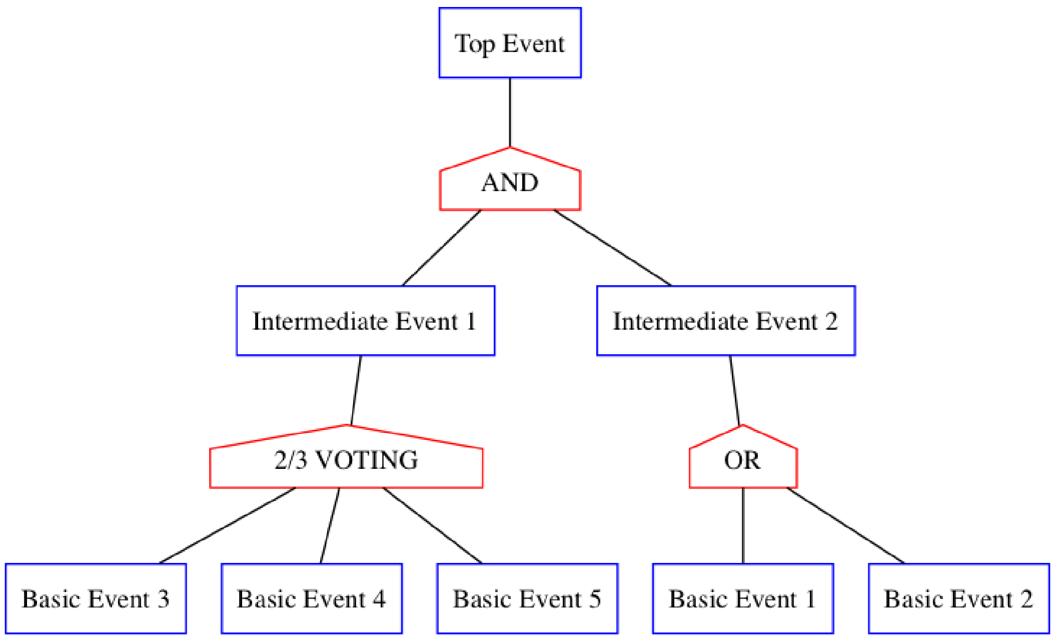
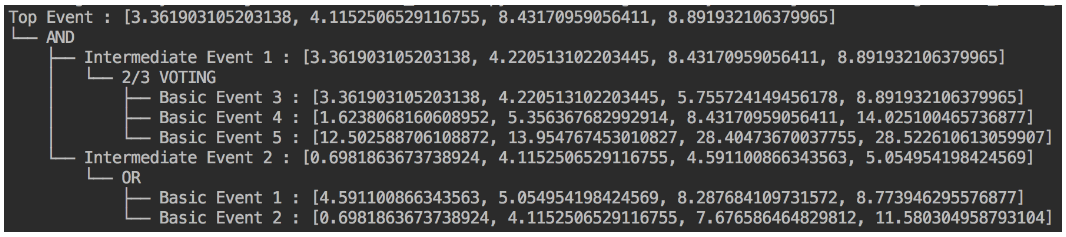


Figure 3.13: Example of a fault tree

We generate time series for each basic event as we learned in Section 3.1.3. The time series are saved as an attribute in the **Event** class (see Section 3.1.2.1). The **Gate** class (see Section 3.1.2.2) is then responsible for evaluating the time series of the input events and then producing the time series for the output event. We evaluate the gates from the bottom up so we propagate the time series of the basic events all the way up to the top event.

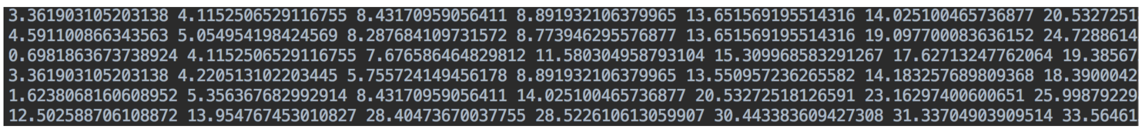


```
Top Event : [3.361903105203138, 4.1152506529116755, 8.43170959056411, 8.891932106379965]
└── AND
    ├── Intermediate Event 1 : [3.361903105203138, 4.220513102203445, 8.43170959056411, 8.891932106379965]
    │   └── 2/3 VOTING
    │       ├── Basic Event 3 : [3.361903105203138, 4.220513102203445, 5.755724149456178, 8.891932106379965]
    │       ├── Basic Event 4 : [1.6238068160608952, 5.356367682992914, 8.43170959056411, 14.025100465736877]
    │       └── Basic Event 5 : [12.502588706108872, 13.954767453010827, 28.40473670037755, 28.522610613059907]
    └── Intermediate Event 2 : [0.6981863673738924, 4.1152506529116755, 4.591100866343563, 5.054954198424569]
        └── OR
            ├── Basic Event 1 : [4.591100866343563, 5.054954198424569, 8.287684109731572, 8.773946295576877]
            └── Basic Event 2 : [0.6981863673738924, 4.1152506529116755, 7.676586464829812, 11.580304958793104]
```

Figure 3.14: Time series of the events in a system

Figure 3.14 shows the generated time series of the fault tree in Figure 3.13. The figure only displays the first four occurrences of each event and keep in mind that the values change after every execution of the program since it relies on random number generators to produce the time series.

The time series of the top event and basic events are then saved in a text file for later analysis. The time series of the intermediate events are omitted since they do not affect the result of the analysis [10, p. 32].



```
3.361903105203138 4.1152506529116755 8.43170959056411 8.891932106379965 13.651569195514316 14.025100465736877 20.5327251
4.591100866343563 5.054954198424569 8.287684109731572 8.773946295576877 13.651569195514316 19.097700083636152 24.7288614
0.6981863673738924 4.1152506529116755 7.676586464829812 11.580304958793104 15.309968583291267 17.62713247762064 19.38567
3.361903105203138 4.220513102203445 5.755724149456178 8.891932106379965 13.550957236265582 14.183257689809368 18.3900042
1.6238068160608952 5.356367682992914 8.43170959056411 14.025100465736877 20.53272518126591 23.16297400600651 25.99879229
12.502588706108872 13.954767453010827 28.40473670037755 28.522610613059907 30.443383609427308 31.33704903909514 33.56461
```

Figure 3.15: Time series saved in text file

Figure 3.15 shows the format of time series that are saved in the text file. The first line displays the time series of the *Top Event*. Starting from the second line are the time series for basic events in numerical order. So the second line displays the time series for *Basic Event 1*, the third line for *Basic Event 2*, and so on and so forth.

3.2 Learning from Data

The next step is to derive the structure of the the fault tree from the time series of the top event and the basic events. We only look at the text file with the time series to determine the structure. We learned that the qualitative analysis of fault tree is used to analyze the structure of fault tree. Specifically, cut sets and minimal cut sets can help us determine the structure [6]. Minimal cut sets give information about what basic events caused a system failure, and from this we can deduce the gate connections between the basic events. Before we can calculate the minimal cut sets we must calculate the cut sets of the fault tree.

3.2.1 Qualitative Analysis

From the time series we can calculate the cut sets of the fault tree. In order to calculate the cut sets, we examine the time series of the top event. At each time-of-failure of top event we determine which basic components caused the failure, or in other words, which basic components are in a failed state.

3.2.1.1 Determine State of Component

A demonstration of an example is used to show how to determine the state of a component from its time series, at a given time. Consider the event shown on Figure 3.16.

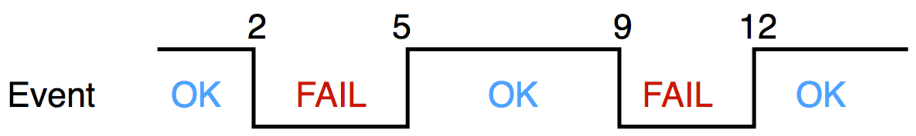


Figure 3.16: Example calculating cut sets

If we want to know if the component failed at *time* = *3*, we find the time in the time series, which is the first time that is greater than *3*. In this case it is *5*, and since *5* is the second value in the time series, more importantly the index of *5* is an even value, therefore we know it is a time-of-repair. But at *time* = *3*, the event has not reached the time of repair so we know the component is non-operational at that time. Therefore the state of the component at *3* is *FAIL*.

Now let us calculate the state of the component at *time* = *9*. However, at *time* = *9*, is a time of transition for the event where the state changes from *OK* to *FAIL*. But we still follow the same method as we used for *time* = *3*. The first number that is greater than *9* is *12*. The index of *12* in the time series is four which is a even value, so *12* is a time-of-repair. Therefore at state of the component at *time* = *9* is *FAIL* according to this algorithm. This makes sense because at *time* = *9* is when the component enters the *FAIL* state.

3.2.1.2 Cut Set Calculation

The status of each basic component in a fault tree is calculated the way it was demonstrated in the previous section. It is calculated at each time-of-failure in the top event time series to yield a list of cut sets. For example, let us determine the first cut set of the time series shown in Figure 3.15. We round the values in this example for simplicity.

We see the first time-to-failure of the top event is *3.362* and then we check whether or not the basic events components failed by this time. *Basic Event 1* has not failed yet since its first time-of-failure is at *4.591*. *Basic Event 2* failed at *0.698* and is not repaired until *4.115*, therefore at *time = 3.362* it is in *FAIL* state. *Basic Event 3* enters the *FAIL* state at *time = 3.362* so it is not operational anymore. *Basic Event 4* failed *1.624* and is not repaired until *5.356* so it is in a *FAIL* state. *Basic Event 5* has not failed yet since its first time-to-failure is at *12.503*, therefore it is still in *OK* state. From the above discussed information we conclude the first cut set to be *(2, 3, 4)*, since *Basic Event 2, 3, and 4* caused the system failure.

Ideally the top event time series should have enough time-of-failures to calculate all the cut sets of the fault tree. The algorithm produces multiples of the same cut set, but we simply remove any duplicates in our list of cut sets. From the time series, which is only partially shown in Figure 3.15, the following cut sets are calculated. The cut sets can further be reduced into minimal cut sets which are also shown below. The minimal cut sets are of more importance to us for the purpose of reconstructing the fault tree, since it is more precise compact form.

- Cut sets: (2, 3, 4), (1, 3, 4), (1, 2, 3, 4), (2, 3, 5), (2, 4, 5), (1, 4, 5), (1, 2, 4, 5), (1, 2, 3, 5), (1, 3, 5), (1, 3, 4, 5), (2, 3, 4, 5)
- Minimal cut sets: (2, 3, 4), (1, 3, 4), (2, 3, 5), (2, 4, 5), (1, 4, 5), (1, 3, 5)

3.2.2 Fault Tree Structure Reconstruction

Using the minimal cut sets we determine the structure of the fault tree. We examine an algorithm that can reconstruct the type of gate connections between the events using techniques from set theory. To demonstrate the algorithm we consider the fault tree on Figure 3.17, which we have previously calculated the minimal cut sets for.

3.2.2.1 Algorithm

Before the algorithm let us cover some abbreviations and notations.

- MCS: Minimal cut set
- BE: Basic event
- Set: Set of minimal cut sets, explained more later.

The first step of the fault tree reconstruction algorithm is to identify each minimal cut set with a unique identifier. In this example the identifiers will be letters for visual purposes, but it can be numbers as well. After this, we create a list of basic events and for each basic event we state what minimal cut set includes that particular basic event. The set of minimal cut sets that include a certain basic event are from now on referred to as a set. Figure 3.18 shows the method of creating a set for each basic event from the minimal cut sets.

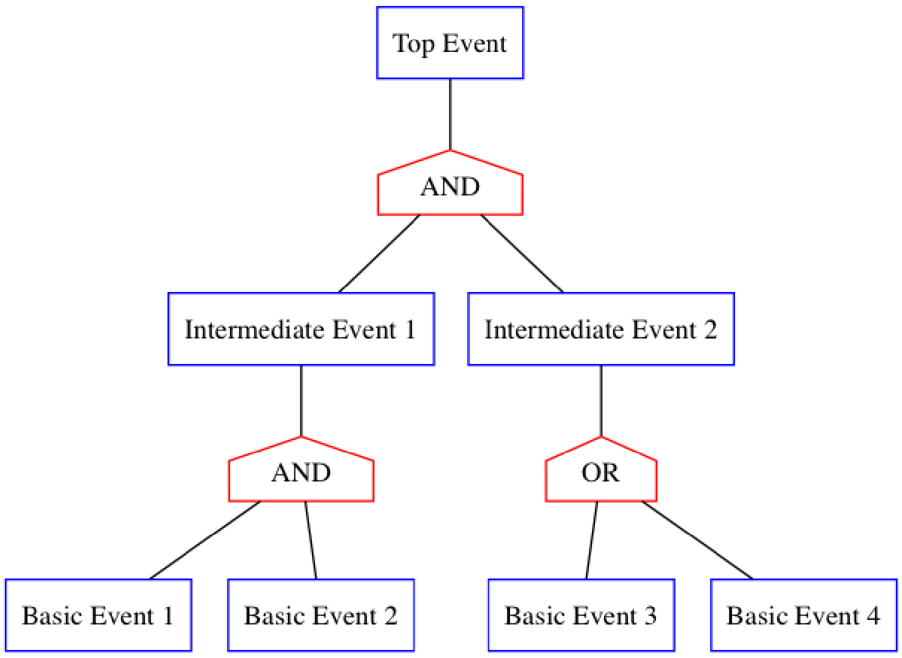
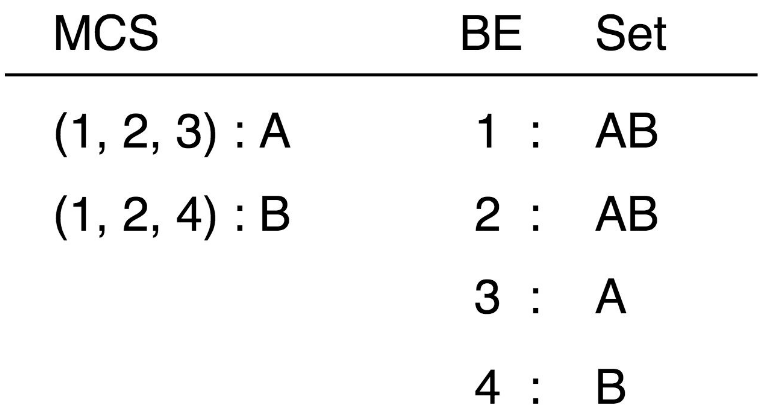


Figure 3.17: Example of fault tree

- Minimal cut sets: (1, 2, 3), (1, 2, 4)



MCS	BE		Set
(1, 2, 3) : A	1	:	AB
(1, 2, 4) : B	2	:	AB
	3	:	A
	4	:	B

Figure 3.18: Minimal cut set identification

The algorithm inspect the set of each basic event and determines the type of gate connection between the basic events. If there are sets that are identical then basic events that belong to those sets are connected by an *AND* gate. If the sets are mutual exclusive the basic events are connected by an *OR* gate. In Figure 3.19, we see that the sets for *BE1* and *BE2* are identical therefore we can conclude these events are connected by an *AND* gate. Afterwards, we merge the set of input events to construct the set of the output event. When we merge sets together we omit duplicates, therefore the output set is *AB*.

At the first gate level, there is still *BE3* and *BE4* remaining. The sets of these two basic events are mutually exclusive, since one set only has *A* and the other only has *B*, there is no overlap. Therefore *BE3* and *BE4* are connected by an *OR* gate. The two sets are merged together to create a set of *AB*.

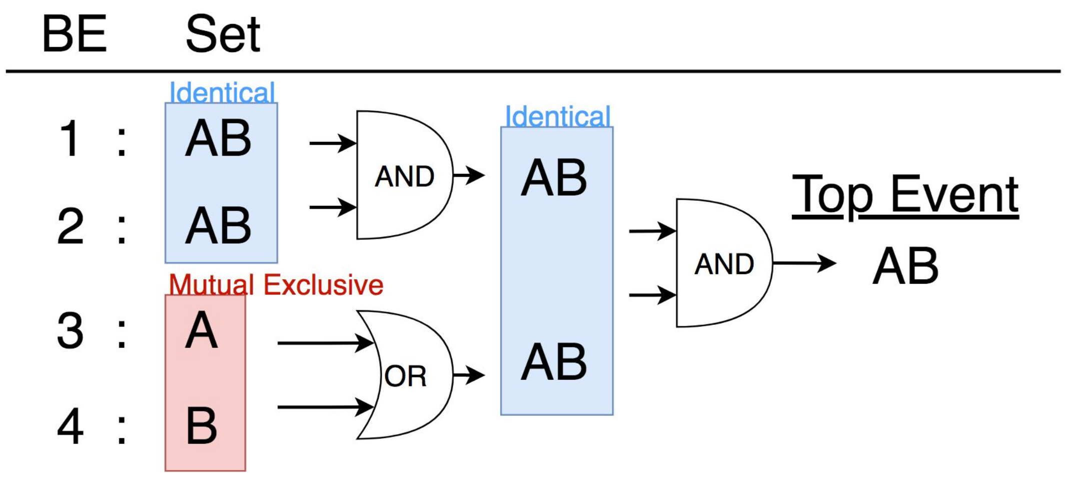


Figure 3.19: Fault tree reconstruction algorithm

At the next gate level, which represents the intermediate events, we have two identical sets. The two intermediate events are connected by an *AND* gate. After merging the sets the result is one set of *AB* representing the top event. If there is only one set remaining we have reached the top event and the end of the algorithm. With the help of this algorithm we successfully reconstruct the shape of a fault tree from the minimal cut sets. However so far we only looked at an example with only *AND* and *OR* gates. Now we look at how the algorithm works with a k/N *VOTING* gate.

The decision of determining if the events are connected by a k/N *VOTING* gate is more complex than for the *AND* or the *OR* gate. We look out the process of determining a k/N *VOTING* gate below. Consider the fault tree with only one gate, $2/3$ *VOTING* gate (see Figure 3.10). This fault tree has the following minimal cut sets: *(1, 2), (2, 3), (1, 3)*. We determine the sets in same manner as we did in the previous example.

MCS	BE	Set
(1, 2) : A	1 :	AC
(2, 3) : B	2 :	AB
(1, 3) : C	3 :	BC

Figure 3.20: Minimal cut set identification

In Figure 3.21, we see that the sets for each basic event are neither identical or mutual exclusive to each other. The sets are in a pattern we call "N choose k" (in this example *N* = *3* and *k* = *2*). If a binomial coefficient (N choose k) can be constructed out of the sets then events are connected by a *k* / *N VOTING* gate.

To investigate if a binomial coefficient can be constructed out of the sets, we first have to check the length of the sets. The length of all the sets must be the same. If they are the same, the number indicating the length of the sets is noted as the *k* in the binomial coefficient, in this example *k* = *2*.

The *N* in the binomial coefficient is determined by the number of unique identifiers (of minimal cut sets) in the sets under investigation. In this example, there are three unique identifiers: *A, B, and C*, therefore *N* = *3*. Another requirement here is that each unique identifier should have the same number of identifiers in the sets. This is also satisfied since there are two *A's*, two *B's*, and two *C's*.

In some cases, the binomial coefficient can be simplified. By checking how many sets are under examination it can determined if simplification is possible. In this example, there are three sets we are examining, and it matches the number we calculated for the *N* in the binomial coefficient. Therefore the binomial coefficient is in its simplest form. Later we look at an example where simplification is needed.

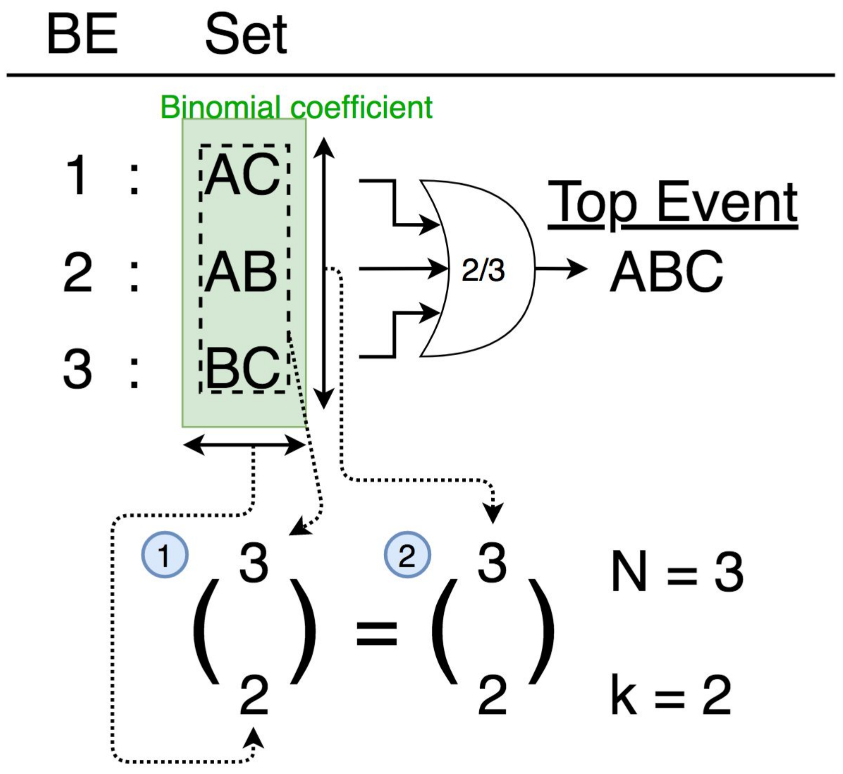


Figure 3.21: Fault tree reconstruction algorithm

If the binomial coefficient is in its simplest form, then the *N* and *k* will be the numbers in the *k*/*N VOTING* gate. The gate in this example is determined to be a *2*/*3 VOTING* gate. After this, we merge the sets of the events to produce a new set of *ABC*. Since there is only one set, we have reached the end of the algorithm and successfully determined the structure of the fault tree from its minimal cut sets.

Now we demonstrate the algorithm on a more complex fault tree. Consider the fault tree on Figure 3.13 and the minimal cut sets we calculated from its time series. The minimal cut set are identified and the sets for the basic events are shown in Figure 3.22.

MCS	BE		Set
(2, 3, 4) : A	1	:	BEF
(1, 3, 4) : B	2	:	ACD
(2, 3, 5) : C	3	:	ABCF
(2, 4, 5) : D	4	:	ABDE
(1, 4, 5) : E	5	:	CDEF
(1, 3, 5) : F			

Figure 3.22: Minimal cut set identification

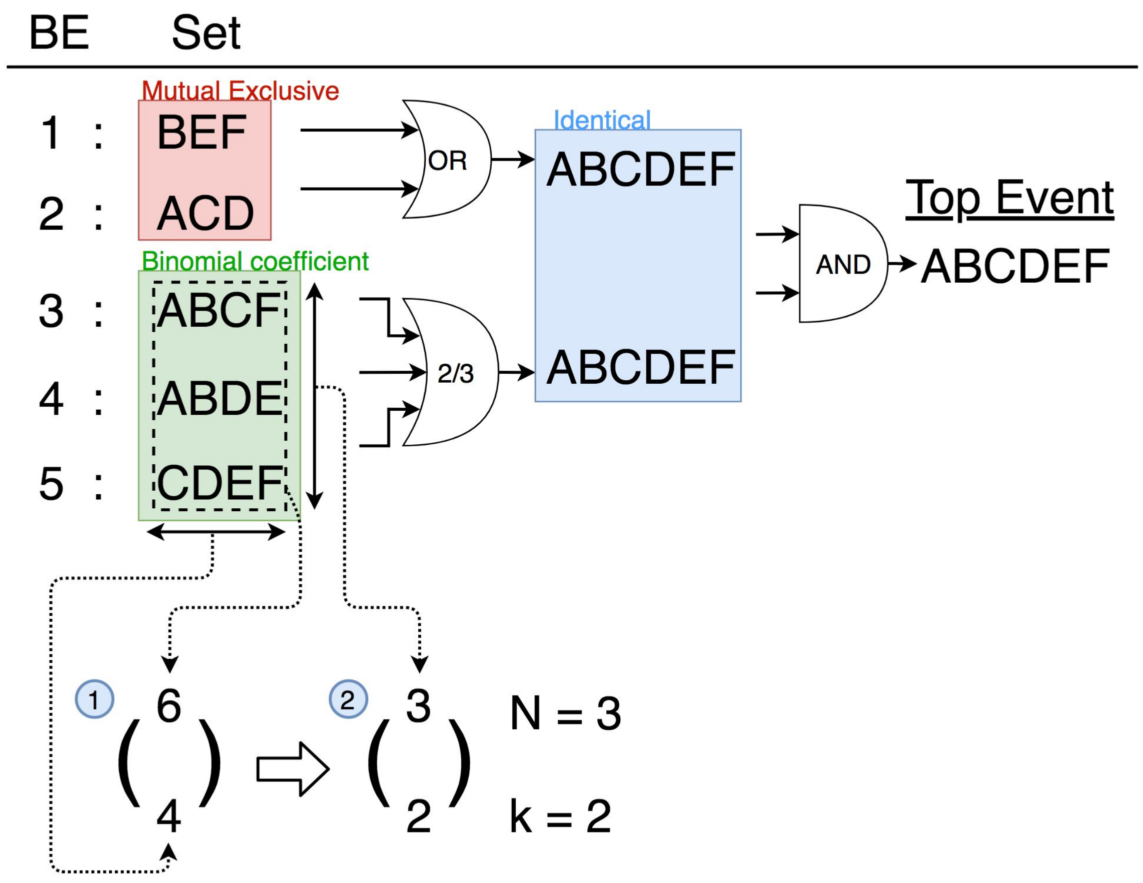


Figure 3.23: Fault tree reconstruction algorithm

Figure 3.23 shows the reconstruction of the fault tree using the algorithm. *BE1* and *BE2* are connected with an *OR* gate since the sets of these events are mutually exclusive. The output set for the *OR* gate is *ABCDEF*. The sets of *BE3, BE4, and BE5* sets are neither identical or mutually exclusive so we examine if a binomial coefficient can be constructed from it. The sets have six unique identifiers and the length of a set is four, which creates a binomial coefficient of *6 choose 4*. However since there are only three sets, the binomial coefficient can be simplified. The simplification is just solving a simple proportion with the values we determined so far.

$$\frac{6}{4} = \frac{3}{x}$$

By cross multiplying and solving the equation we get $x = 2$. Therefore the binomial coefficient is simplified to *3 choose 2*. The gate connecting *BE3, BE4, and BE5* is a *2/3 VOTING* gate. The output set for this gate is *ABCDEF*.

One the next level, which represent the intermediate events, we have two identical sets. This means the gate connecting the two intermediate events is a *AND* gate. The output of the gate is *ABCDEF* and it is the only set remaining. Therefore we reached the end of the algorithm and determined the shape of the fault tree. If we compare the reconstruction with the original fault tree on Figure 3.13, we see the algorithm successfully reconstructed the fault tree.

3.2.3 Quantitative Analysis

Besides the structure of the fault tree we can also learn from the data the distribution of the basic events. We do this through quantitative analysis methods to determine the reliability and maintainability distribution of the basic events. Other metrics such as mean time to failure, mean time to repair, and operational availability are also calculated.

3.2.3.1 Time Series Decoding

Before we can determine any distributions or calculate any metrics, we need to decode the time series into times-to-failures and times-to-repairs. The way this is done is essentially the reverse of the algorithm explained in Table 3.1. Consider the first six times in time series of *Basic Event 1* from Figure 3.15. We demonstrate the decoding of the time series into times-to-failure and times-to repair in Table 3.3 below. The time series values are rounded for simplicity.

Table 3.3: Time series decoding

Time series	[4.591, 5.055, 8.288, 8.774, 13.652, 19.098]
Time elapsed since last time stamp	[4.591, 0.464, 3.233, 0.486, 4.878, 5.446]
Times-to-failure	[4.591, 3.233, 4.878]
Times-to-repair	[0.464, 0.486, 5.446]

The first step is to calculate the difference between the current and next time stamp. Since the start time is zero, the first element in the time series is the first element in the 'time elapsed since last time stamp'. Afterwards the we take the difference between the time stamps, for example *5.055 - 4.591 = 0.464*, and so on and so forth.

After calculating the differences between the time stamps, we split the array in two. The odd indexed elements in the array are the times-to-failure and the even indexed elements are the times-to-repair.

The time series decoding is done for each event in the fault tree and the times-to-failure and times-to-repair are used for later analysis.

3.2.3.2 Distribution Fitting

Since we know the the times-to-failure and times-to-repair we determine the reliability and maintainability distribution of the component, respectively. We use the one sample Kolmogorov-Smirnov test to determine the distribution from the sample of times-to-failure and times-to-repair. The python **scipy.stats** package has many statistical methods that help determine the parameters of the distribution and fit a distribution to a sample [24].

The distribution fitting is done by testing four distributions (*exponential, normal, weibull, and lognormal*) and calculating the p-value for each of them using the Kolmogorov-Smirnov test. If the p-value is above the significant value of *0.05*, then we accept the null-hypothesis that the sample could have been generated by that distribution. The parameters for the distributions are also estimated when conducting the test. The distribution fitting is not always successful, if none of the p-values are above the significant value. In this case the distribution was not found to be any of the four distributions we cover.

In case a specific distribution cannot be determined a general distribution fitting is done with the help of a *Python* package called **Seaborn** [25]. It is used for the visualizing the distribution for the top event and for basic events where the distributions were not determined.

It is important to note the quality of the distribution fitting depends on the size of the times-to-failures and times-to-repairs. With ample amount of time series data the fitted distribution resembles the theoretical distribution.

3.2.3.3 Mean Time to Failure

After determining the times-to-failure from the time series, calculating the mean time to failure is simple. Calculating the average of the times-to-failure yields the mean time to failure. The larger the size of the times-to-failure array the more accurate the MTTF will be and it will approach the theoretical MTTF of the reliability distribution which is calculated using the distribution parameters.

3.2.3.4 Mean Time To Repair

Calculating the mean time to repair is done is similar manner, by taking the average of the times-to-repair. The calculated MTTF will also approach the theoretical MTTF of maintainability distribution if the size of the times-to-repair is large.

3.2.3.5 Operational Availability

The operational availability is the percentage of time a component or system is operational. To calculate it, we need to decide up to what time we want to measure the operational availability, we call this the total time. It is important the total time cannot be larger than the last time in the time series, since the component was not simulated past that point. In case of the operational availability of the system, it is best to use the largest time stamp in the shortest time series, of the basic events, as the maximum value for the total time to yield a valid operational availability.

We demonstrate the calculation of the operational availability from a time series with an example (see Figure 3.24).

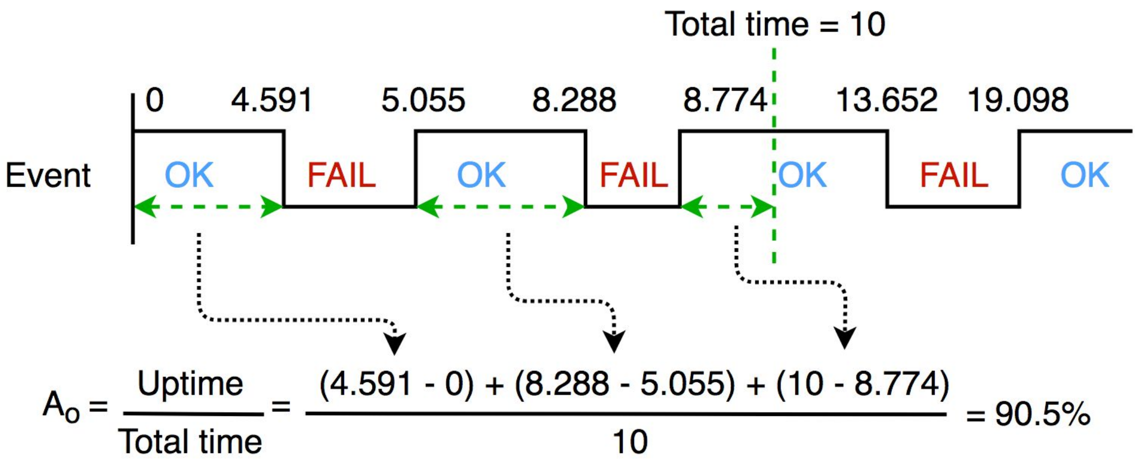

Total time = 10
0
4.591
5.055
8.288
8.774
13.652
19.098
Event
OK
FAIL
OK
FAIL
OK
FAIL
OK
$A_O = \frac{\text{Uptime}}{\text{Total time}} = \frac{(4.591 - 0) + (8.288 - 5.055) + (10 - 8.774)}{10} = 90.5\%$

Figure 3.24: Operational availability calculation

We want to calculate the operational availability up to *time = 10*. The formula shows we need to calculate the uptime first. Uptime is the time the component or system is operational. To calculate the uptime we first add the time to the first failure. Then we add the times between consecutive time-of-repairs and time-of-failures. This is done until we reach the total time. There can be cases that we need to add the remaining time to the uptime if the last time stamp in the time series is a time-to-repair. Then we simply add the difference between the total time and last time stamp to the uptime we calculated so far. The operational availability is then calculated by simply dividing the uptime by the total time and the result is usually given as a percentage.

Figure 3.24 shows the method of calculating the operational availability for the *Basic Event 1* in the time series on Figure 3.15, up to *total time = 10*. The operational availability for *Basic Event 1* is calculated to be *90.5%*. This means the component is operational *90.5%* of the time until *time = 10*.

3.2.3.6 Instantaneous Availability

The proxel-based simulation is used to determine the instantaneous availability of basic components and the entire system. After reconstruction of the fault tree, the reliability and maintainability distribution of each basic component is used to calculate the availability with respect to time.

We then propagate the availability of each individual component through the fault tree to calculate the availability of the system. The availability is treated as a probability since it is the probability that the component or system is operational. Therefore, we use the equations for probability to propagate the availability through the fault tree.

Chapter 4

Experiments

In this chapter I present the results from conducting the reconstruction algorithm for different scenarios. The chapter is divided into two main parts. A section is dedicated to comparing the structure of the fault trees, which are the results from the qualitative analysis. The other section is about quantitative analysis and how the reliability and maintainability distribution are determined from the times series compare to the theoretical distributions.

4.1 Qualitative Analysis

The algorithm is able to reproduce a fault tree that is behaviorally equivalent to the original fault tree however there can be some topological differences depending on the composition of the gates. We look at the structural differences between different configuration of fault trees and their reconstructed counterparts.

4.1.1 Scenario 1: "Simple" Fault Tree

The first configuration of a fault tree we investigate is one we have looked at multiple times before in the previous chapter to demonstrate examples. It is a relatively simple fault tree with only three gates consisting of only the two basic gate types *AND* and *OR*.

With a simple fault tree such as shown on Figure 4.1, the algorithm produces a reconstructed fault tree that is not just behaviorally equivalent but also topologically equivalent to the original fault tree.

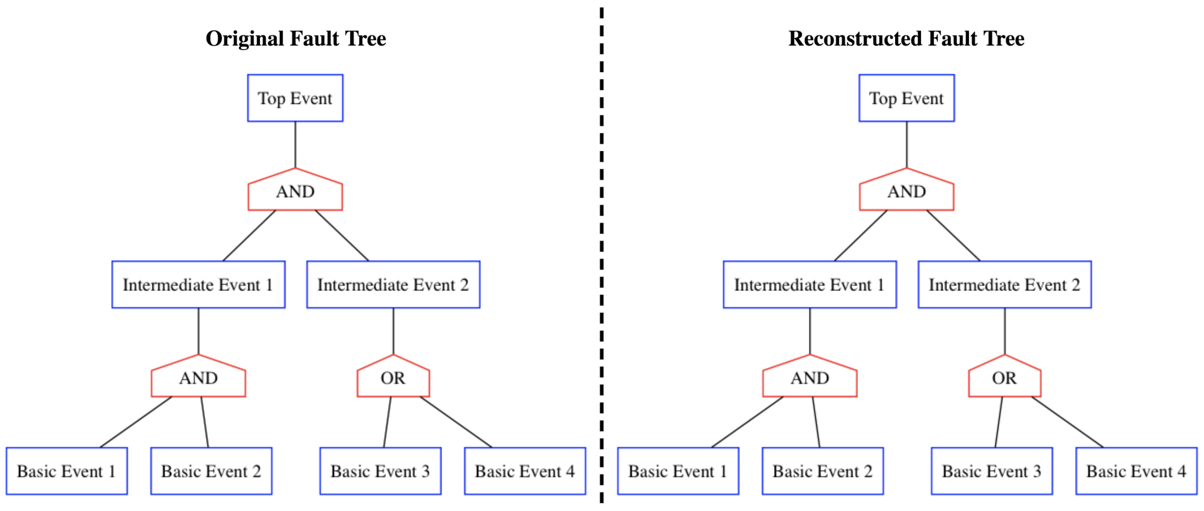


Figure 4.1: Reconstruction of a "simple" fault tree

4.1.2 Scenario 2: Fault Tree with special cases of *k/N VOTING* gates

The second configuration is one with the special cases of the *k/N VOTING* gate. We have discussed previously, that there are special cases of *k/N VOTING* gate, where it is reduced to one of the other gates. Below is an overview of how they can be reduced.

- *1/N VOTING* gate is equivalent to an *OR* gate
- *N/N VOTING* gate is equivalent to an *AND* gate

The algorithm successfully recognizes these special cases of the *k/N VOTING* gates and in the reconstructed fault tree they are replaced by their respective equivalent counterparts.

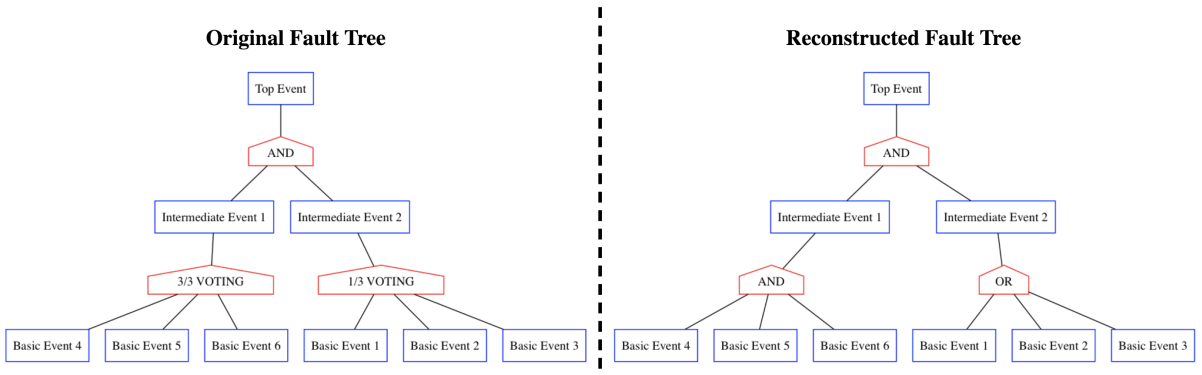


Figure 4.2: Reconstruction of fault tree with special cases of *k/N VOTING* gates

4.1.3 Scenario 3: Fault Tree with only *OR* gates

The third configuration is fault tree with two layers of *OR* gates only. Since logically, the top event of the original fault tree occurs if any of the basic events occur, the reconstructed fault tree reflects this behavior explicitly. The reconstructed fault tree has only one layer of gates, which means only one *OR* gate. The two fault trees behave exactly the same however topologically they are different.

The topological difference can cause problems if the fault tree is used for fault diagnosis [16]. Since the reconstructed fault tree might not reflect the physical architecture of the system under investigation, but rather only the behavioral representation of it.

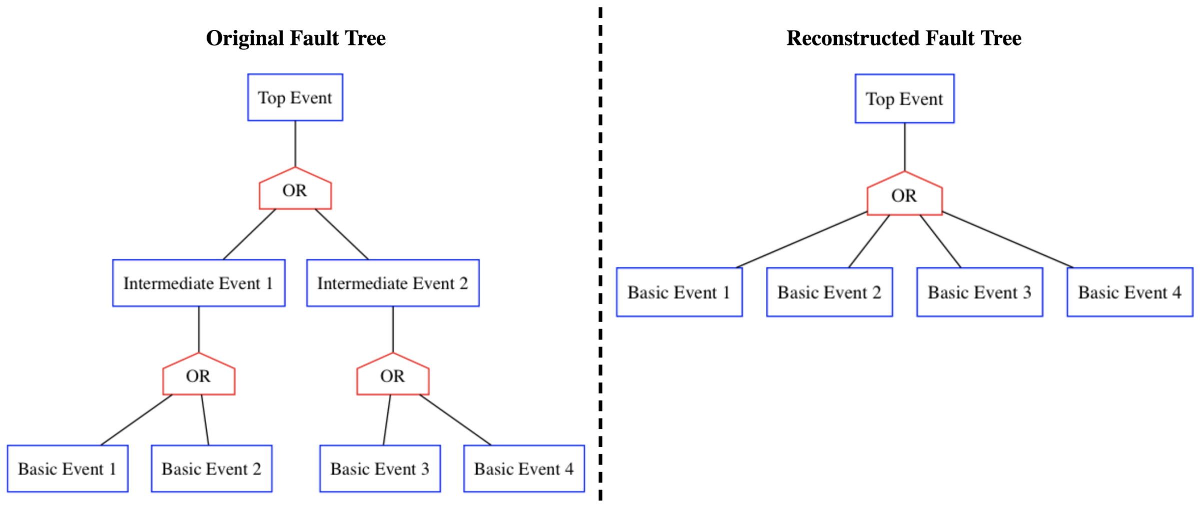


Figure 4.3: Reconstruction of fault tree with only *OR* gates

4.1.4 Scenario 4: "Complex" Fault Tree

The fourth and last scenario we look at is a fault tree configuration that can be considered more complex than the previous fault trees. The fault tree on Figure 4.4 has three layers of gates with at least one of each type of gate. The original fault tree and reconstructed fault tree have topological differences.

In fact the output of some of the gates are different intermediate events (notice *Intermediate Event 3* and *Intermediate Event 4*) on the reconstructed fault tree then on the original fault tree. However, the names of the intermediate events are not of importance, since they are identifiers for a combination of basic events occurring and do not directly influence the analysis [1, p. 32].

The combination of the basic events and their effect on the top event are exactly the same on both the original and reconstructed fault tree. Therefore the two fault trees behave the exact same way.

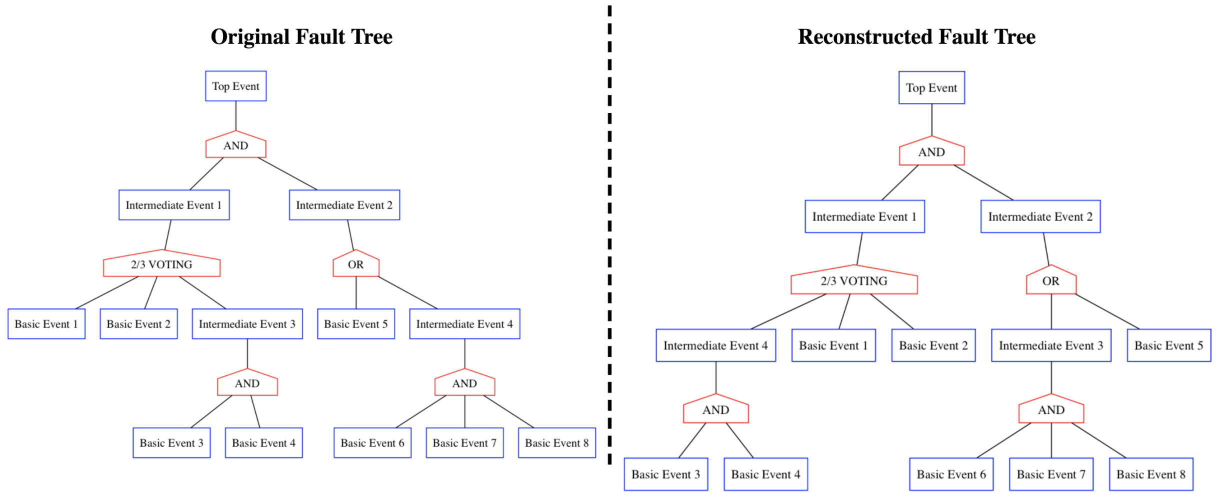


Figure 4.4: Reconstruction of "complex" fault tree

4.1.5 Boolean Manipulation

Boolean manipulation is a classic method of determining minimal cut sets [1, p. 35], but in our case it is a good way to confirm the behavior of two fault trees. In boolean manipulation, each basic component is given a state of *1 or 0* representing if the component is operational or not, respectively. Then based on the configuration of the gates the state of the system is computed. This information is written in a truth table with the input variable being the basic component and the output variable being the system. The truth tables of the original fault tree and reconstructed fault tree are then compared and checked to see if their behavior is the same.

4.1.6 Minimal Cut Sets

The minimal cut sets of the original and reconstructed fault tree are exactly the same since the minimal cut sets are basis of the reconstruction algorithm. The algorithm uses the minimal cut sets of the original fault tree to reconstruct the shape of the resulting fault tree.

4.2 Quantitative Analysis

To demonstrate the results of the quantitative analysis we consider the fault trees on Figure 4.1. The four probability distributions we have covered will be the reliability and maintainability distributions of each of the basic events. The distributions are assigned to the basic events of the original fault tree as shown in Table 4.1.

Table 4.1: Basic events and their reliability and maintainability distributions

	Reliability	**Maintainability**
Basic Event 1	Exponential	Exponential
Basic Event 2	Normal	Normal
Basic Event 3	Lognorm	Lognorm
Basic Event 4	Weibull	Weibull

From the time series generated by the original fault tree the distribution of the basic events and top event are determined and compared with the theoretical distributions of the events. In this demonstration, the generated time series has a total of *10000* time stamps.

4.2.1 Basic Events

Since the basic events were generated from defined probability distributions we attempt to uncover that distribution by applying distribution fitting on the times-to-failures and times-to-repairs. In case, none of the probability distribution match the data, an arbitrary univariate distribution fitting is conducted.

The theoretical mean time to failure and mean time to repair is calculated by using the parameters of the theoretical reliability and maintainability distribution, respectively. The reconstructed mean time to failure and mean time to repair are calculated from the time series. The larger the time series the more they will approach the theoretical MTTF and MTTR.

The operational availability is only calculated for reconstructed fault tree since it has to be simulated and cannot be computed analytically.

4.2.1.1 Exponential Distribution

The exponential distribution is a function that is fitted relatively easily and the reconstructed parameter is matched close to the parameter of the theoretical distribution. Both MTTF and MTTR approach the theoretical value.

Table 4.2: Metrics of Basic Event 1

	Theoretical	**Reconstructed**
Reliability	Exp(λ = 0.06667)	Exp(λ = 0.06756)
Maintainability	Exp(λ = 0.2)	Exp(λ = 0.20029)
Mean Time To Failure	15	14.81514
Mean Time To Repair	5	4.9932
Operational Availability		75.268%

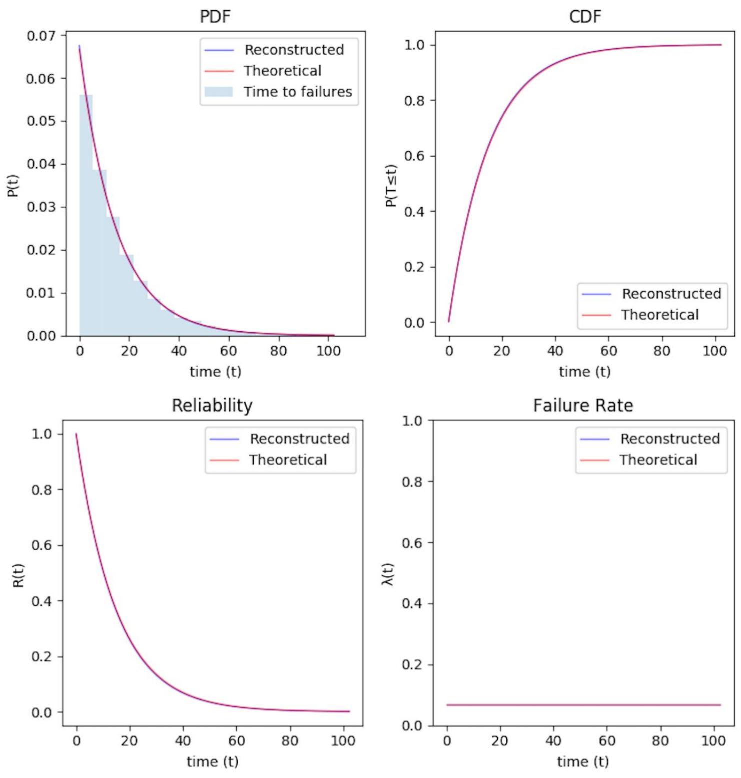


Figure 4.5: Reliability of *Basic Event 1*: Exponential distribution

As shown on both Figure 4.5 and 4.6 the distribution is fitted almost perfectly on the theoretical distribution. The constant failure and repair rate of the exponential distribution is also seen on both Figures.

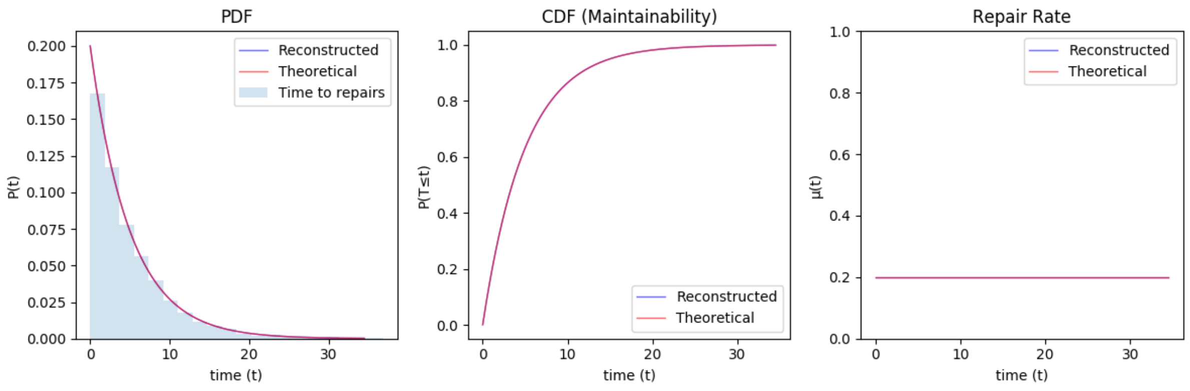


Figure 4.6: Maintainability of *Basic Event 1*: Exponential distribution

4.2.1.2 Normal Distribution

The normal distribution is also fitted with high accuracy due to its defined bell shape. The parameters of the distribution is determined to be close to the theoretical value of the parameters. The MTTF and MTTR have less than one percent error compared to the theoretical value. Figure 4.7 and 4.8 show the distribution is fitted close to the theoretical distribution with minor deviation at the peak of the bell curve.

Table 4.3: Metrics of Basic Event 2

	Theoretical	**Reconstructed**
Reliability	Normal(μ = 12, σ = 2)	Normal(μ = 11.99049, σ = 1.98088
Maintainability	Normal(μ = 5, σ = 1)	Normal(μ = 5.01062, σ = 0.98633)
Mean Time To Failure	12	11.99049
Mean Time To Repair	5	5.01062
Operational Availability		70.547%

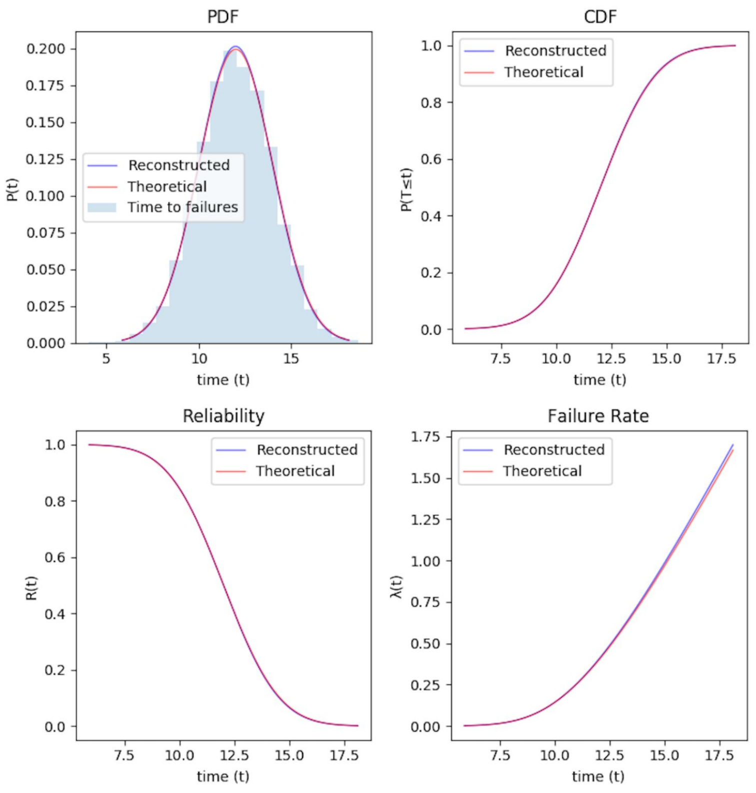


Figure 4.7: Reliability of *Basic Event 2*: Normal distribution

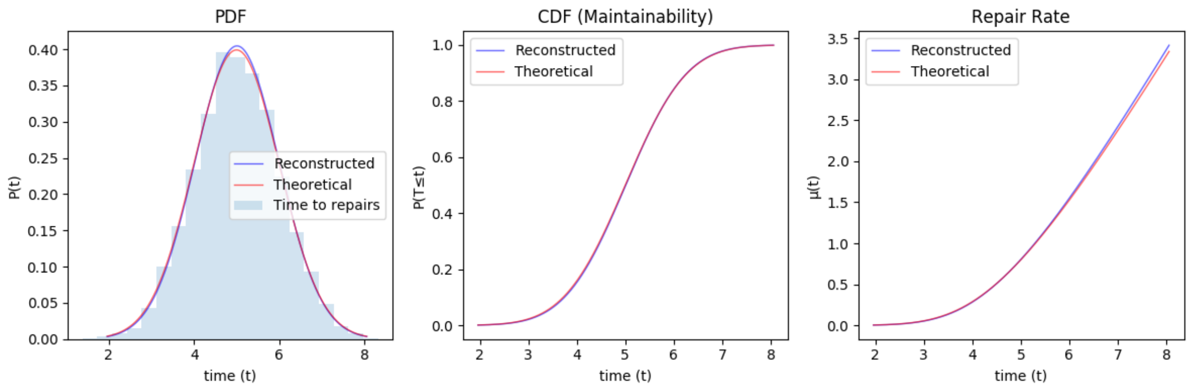


Figure 4.8: Maintainability of *Basic Event 2*: Normal distribution

4.2.1.3 Lognormal Distribution

The lognormal distribution is harder to fit then the previous two distributions. There are times when the distribution is unidentified even with a large sample as is the case for the maintainability distribution in this example. Despite not identifying the distribution, we still fit an arbitrary distribution on the times-to-repair, which is able to closely follow the theoretical distribution.

In case the distribution fitting is successful the determined parameters are close to their theoretical counterpart. The MTTF and MTTR also show the theoretical values.

Table 4.4: Metrics of Basic Event 3

	Theoretical	**Reconstructed**
Reliability	Lognormal(μ = 3, σ = 1)	Lognormal(μ = 3.00279, σ = 0.98794)
Maintainability	Lognormal(μ = 2, σ = 1)	N/A
Mean Time To Failure	33.11545	32.79888
Mean Time To Repair	12.18249	12.12119
Operational Availability		74.315%

In Figure 4.10, we notice that the repair rate dramatically increases after time passes $t = 300$. This is because the reconstructed PDF reaches zero, while the theoretical does not. In Figure 4.9 we see the failure rate should steadily decrease and converge to zero.

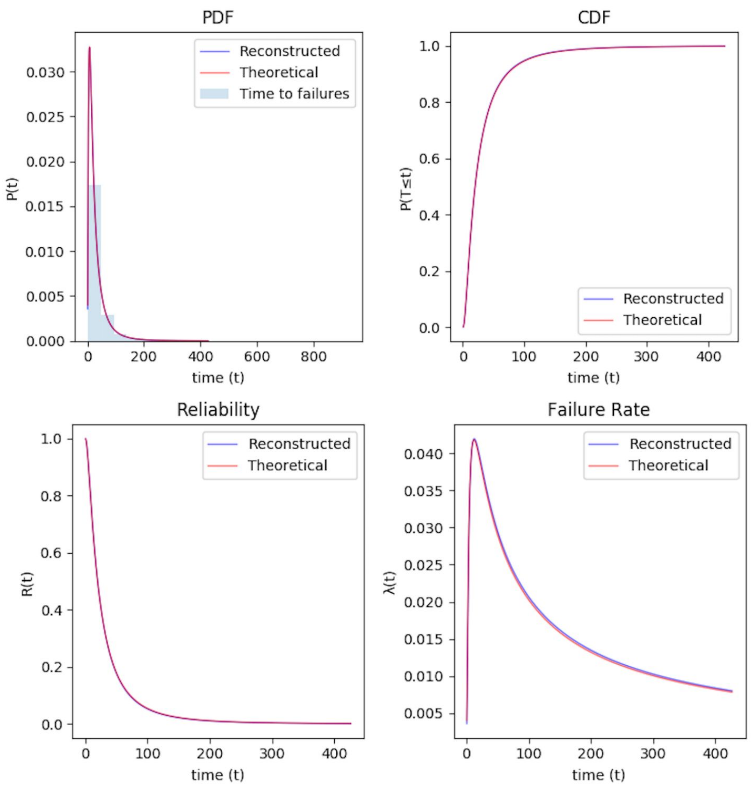


Figure 4.9: Reliability of *Basic Event 3*: Lognormal distribution

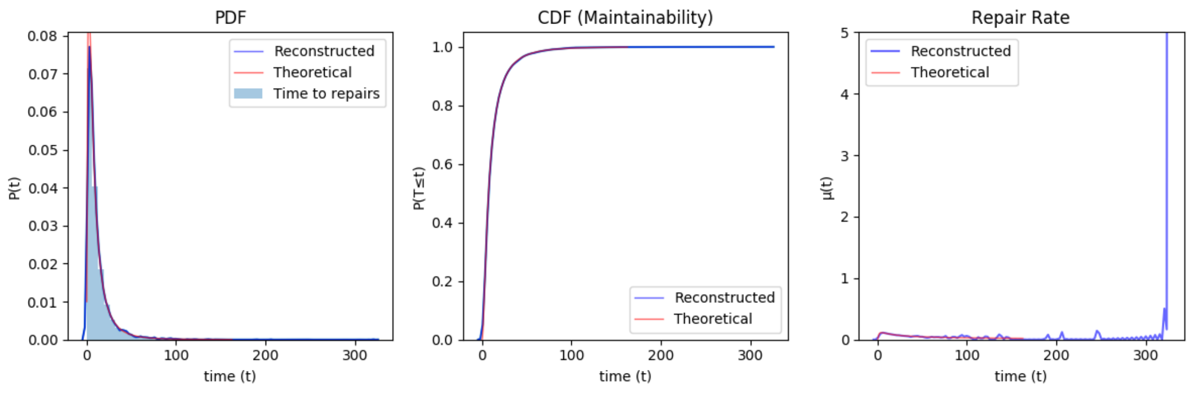


Figure 4.10: Maintainability of *Basic Event 3*: Lognormal distribution

4.2.1.4 Weibull Distribution

The weibull distribution is one of the more difficult distributions to determine from a sample due to its versatility. There are cases when the weibull distribution is not determined, for this example however, the algorithm successfully determined the distribution to be weibull.

The parameters of the distribution show a higher percentage of error than the previous distributions, with an error about five percent in case of the reliability distribution. This is also noticeable in Figure 4.11. However, the maintainability distribution shows less of an error as seen in Table 4.5 and Figure 4.12. It shows that the successfulness of the fitting is dependent also on the shape parameter of the weibull distribution. Nonetheless, the reconstructed MTTF and MTTR are close to the theoretical values.

Table 4.5: Metrics of Basic Event 4

	Theoretical	**Reconstructed**
Reliability	Weibull($\alpha = 10$, $\beta = 8$)	Weibull($\alpha = 9.55126$, $\beta = 7.53683$)
Maintainability	Weibull($\alpha = 5$, $\beta = 2$)	Weibull($\alpha = 4.98135$, $\beta = 1.98419$)
Mean Time To Failure	9.41743	9.41189
Mean Time To Repair	4.43113	4.45705
Operational Availability		67.775%

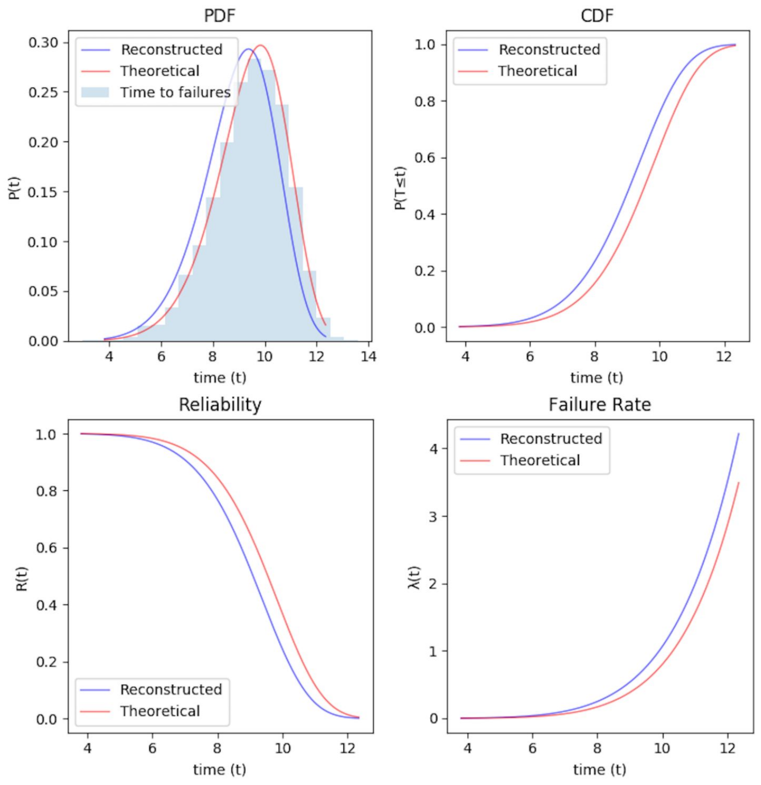


Figure 4.11: Reliability of *Basic Event 4*: Weibull distribution

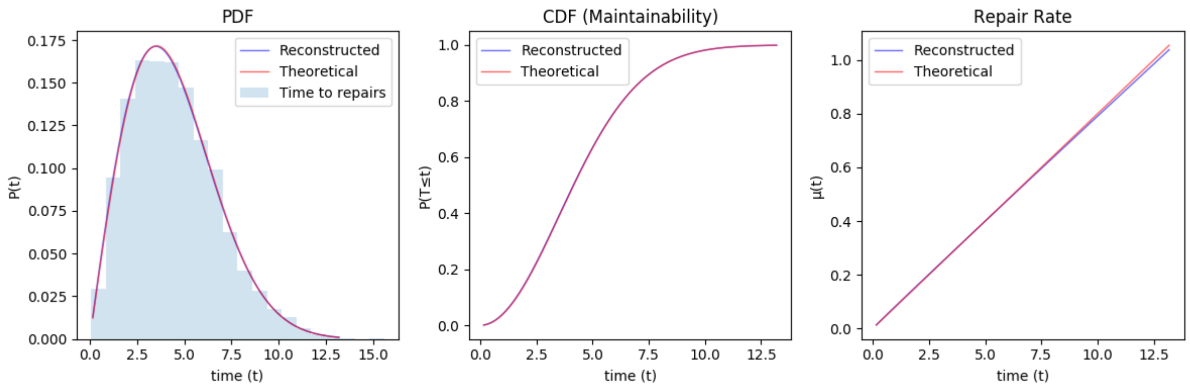


Figure 4.12: Maintainability of *Basic Event 4*: Weibull distribution

4.2.2 Top Event

The time series for the reconstructed top event is calculated from the time series of the reconstructed basic events using the fault tree on Figure 4.1. The theoretical metrics and theoretical distributions should not be considered for the top event. The reason is because the reliability and maintainability distributions are computed separately and do not take into account the random variables of the other distributions [10, p. 135].

For example, the analytical reliability distribution shows the probability that a component stays operational before a failure even occurs. Let us consider that the analytical reliability of the basic events would be propagated through a fault tree and we compute the analytical reliability of the top event. This reliability would only show the probability that the system is operation before the first failure. It does not consider if the components were to be repaired. Therefore this analytical reliability cannot be compared to the simulated reliability of the system computed through generated time series. If we were to compare the maintainability of the top event, for example, the result would look similar to Figure 4.15. The theoretical reliability shows the reliability of the system before the first failure, while the reconstructed shows the reliability when failures and repairs are taken into account.

The metrics calculated from simulating the behavior of the fault tree are shown below. Figure 4.13 and 4.14 show the reliability and maintainability of the top event, respectively.

- *Mean Time To Failure:* 54.14936
- *Mean Time To Repair:* 1.84789
- *Operational Availability:* 96.427%

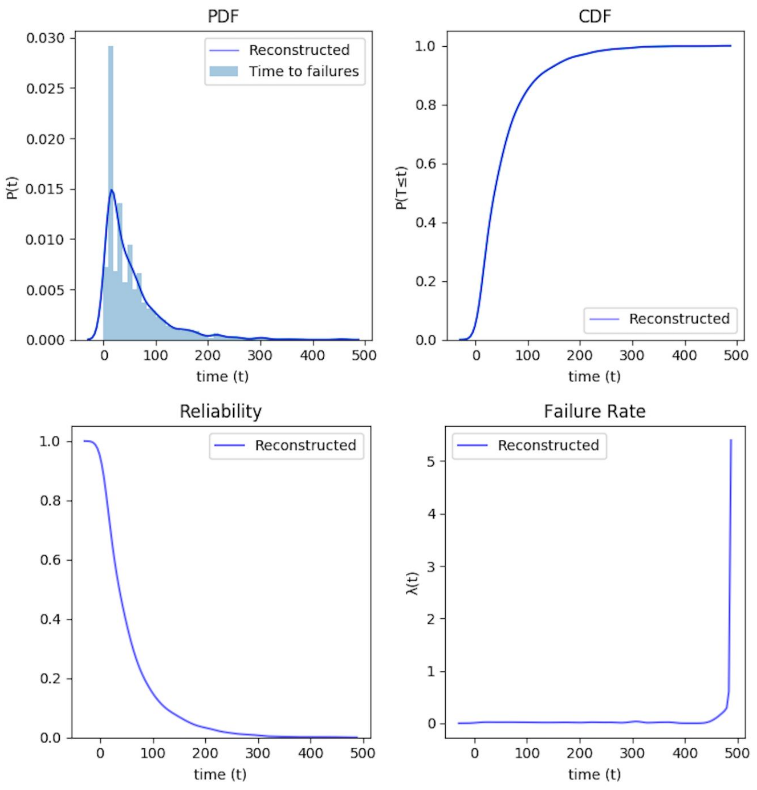


Figure 4.13: Reliability of *Top Event*

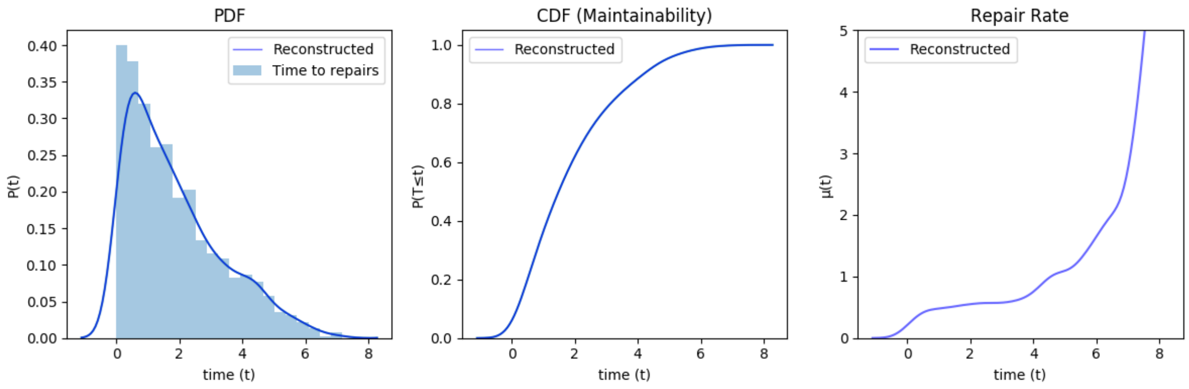


Figure 4.14: Maintainability of *Top Event*

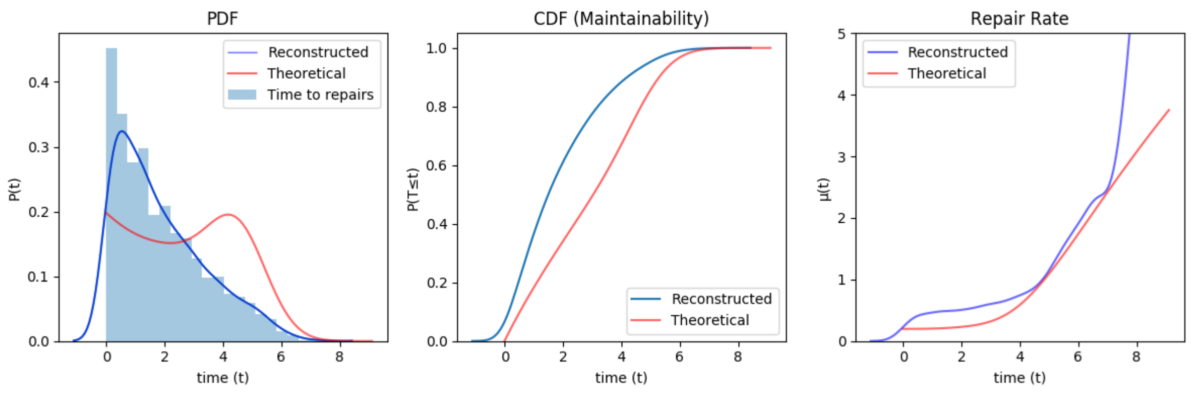


Figure 4.15: Improper comparison of maintainability of the *Top Event*

4.2.3 Proxel-Based Simulation

The proxel-based method is used to determine the instantaneous availability of a system at a given time. Consider the fault tree configuration on Figure 4.1. The time series is generated from the original fault tree with the reliability and maintainability distributions in Table 4.6. The fault tree is then reconstructed from the time series data, and the distributions and metrics are then determined from the reconstructed fault tree.

Instead of only calculating the operational availability, which is a single value, we want to determine the instantaneous availability at any given time. In order to calculate this, we perform the proxel-based simulation on the reconstructed fault tree.

Table 4.6: Original Fault Tree: Basic events

	Reliability	**Maintainability**
Basic Event 1	Normal($\mu = 4$, $\sigma = 1$)	Exp($\lambda = 0.5$)
Basic Event 2	Exp($\lambda = 0.33333$)	Normal($\mu = 4$, $\sigma = 1$)
Basic Event 3	Normal($\mu = 4$, $\sigma = 1$)	Exp($\lambda = 0.5$)
Basic Event 4	Lognormal($\mu = 2$, $\sigma = 1$)	Weibull($\alpha = 5$, $\beta = 2$)

Table 4.7 displays the metrics calculated for *Basic Event 1*. Figure 4.16 shows the availability for *Basic Event 1*. On the plot, we see how the availability changes over time and if compare it with the operational availability, we see the instantaneous availability is still oscillating but it starts to approach the operational availability. This is because the instantaneous availability approaches the steady state availability approximately around four times of the MTTF [21]. While the operational availability also approaches the steady state availability as the size of the time series increases.

Table 4.7: Metrics of *Basic Event 1*

Reliability	Normal(μ = 3.98301, σ = 0.98623)
Maintainability	Exp(λ = 0.48906)
Mean Time To Failure	3.98301
Mean Time To Repair	2.04603
Operational Availability	65.864%

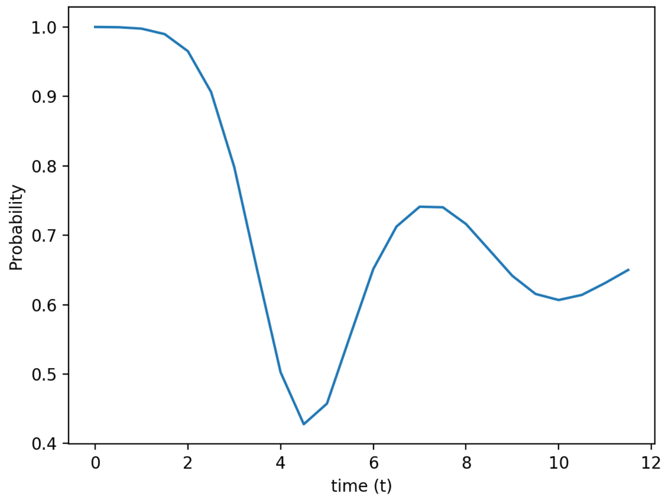


Figure 4.16: Availability of *Basic Event 1*

In the case of *Basic Event 2*, the mean time to failure is smaller and Figure 4.17 displays the time axis up four times the MTTF. Therefore we see that the instantaneous availability is stabilizing and is converging to the steady state availability, which is approximately between *40-45 %*. The operational availability is also in this range.

Table 4.8: Metrics of *Basic Event 2*

Reliability	Exp(λ = 0.34136)
Maintainability	Normal(μ = 4.02739, σ = 1.00933)
Mean Time To Failure	2.92958
Mean Time To Repair	4.02739
Operational Availability	42.235%

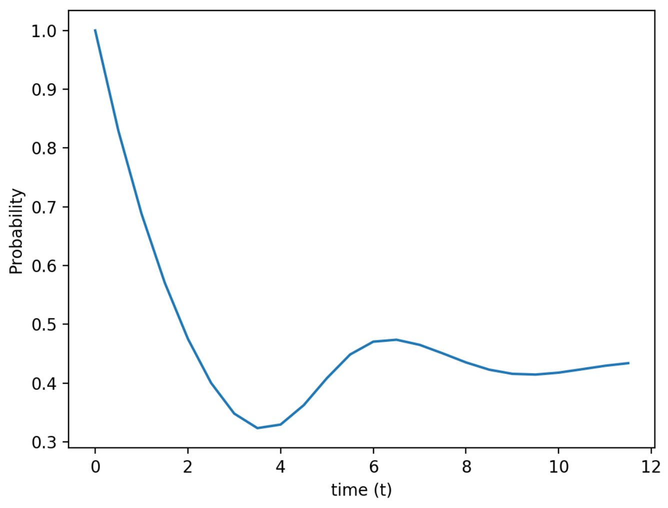


Figure 4.17: Availability of *Basic Event 2*

The metrics and availability plots for *Basic Event 3 and 4* are also shown below.

Table 4.9: Metrics of *Basic Event 3*

Reliability	Normal(μ = 3.99929, σ = 1.00017)
Maintainability	Exp(λ = 0.50643)
Mean Time To Failure	3.99929
Mean Time To Repair	1.97562
Operational Availability	67.174%

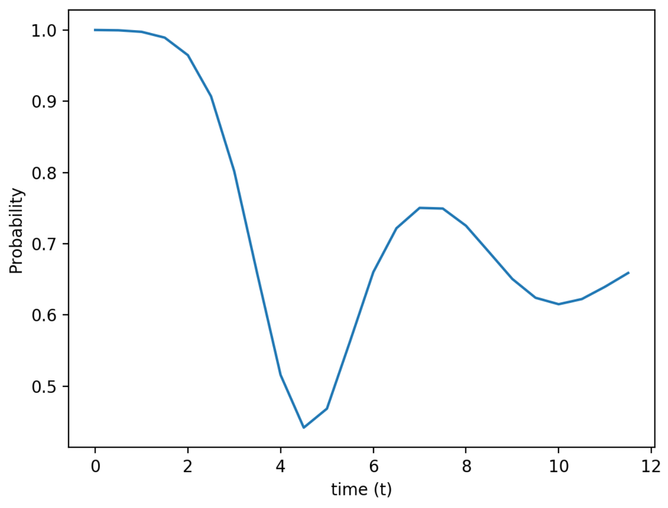


Figure 4.18: Availability of *Basic Event 3*

Table 4.10: Metrics of *Basic Event 4*

Reliability	Lognormal(μ = 1.98057, σ = 0.99026)
Maintainability	Weibull(α = 5.01452, β = 1.97327)
Mean Time To Failure	11.83888
Mean Time To Repair	4.47729
Operational Availability	71.617%

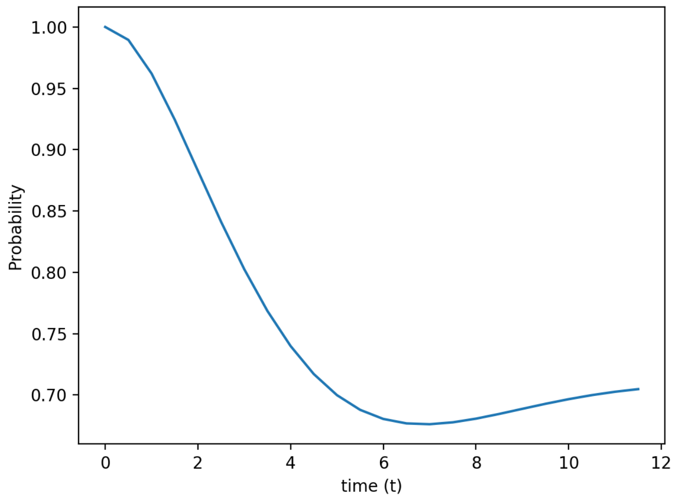


Figure 4.19: Availability of *Basic Event 4*

The metrics for the *Top Event* are shown in Table 4.11. Figure 4.20 shows the instantaneous availability and we see it is still in the process of stabilizing. The mean time to failure is *8.34644*, however Figure 4.20 only shows the time up to *12*, so we cannot see where exactly the availability converges to. It seems to approach a value between *85-95 %* which is supported by the value of the operational availability of the *Top Event*.

Table 4.11: Metrics of *Top Event*

Mean Time To Failure	8.34644
Mean Time To Repair	0.95131
Operational Availability	89.36%

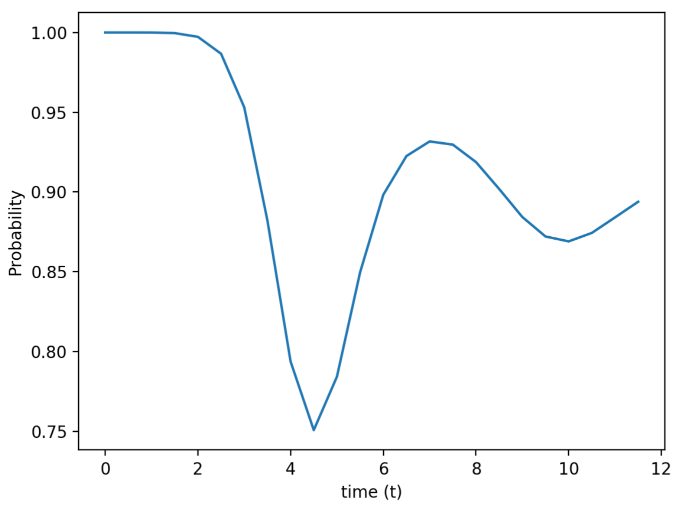


Figure 4.20: Availability of *Top Event*

Chapter 5

Evaluation

In this chapter, I evaluate the reconstruction algorithm of the fault tree. I explain the limitations the algorithm has and aspects where improvements can be made. I discuss the significance of my findings and discuss why is data-based generation of reliability models relevant and how it can contribute to system analysis.

5.1 Limitations

The limitations of the algorithm have to be considered. They are divided into two categories, the limitations of the qualitative analysis and the limitations of the quantitative analysis.

5.1.1 Qualitative Analysis

There are limitations with the qualitative analysis that we need to understand and consider before using the solution. These limitations can affect the structure of the resulting fault tree. The reconstruction algorithm might not successfully recreate the fault tree which leads to a false representation of a system.

5.1.1.1 Incomplete Minimal Cut Sets

The reconstruction algorithm works by calculating the cut sets from the time series, reducing cut sets to minimal cut sets, and reconstructing the shape of the fault tree using minimal cut sets. The algorithm must find all the minimal cut sets of the original fault tree.

It does not necessarily need to find all the cut sets. Say for instance, if one cut set is not found, and that one cut set would be reduced to a minimal cut set, the algorithm would not notice the difference. Consider the following example, the algorithm analyzes a time series and finds the following minimal cut set:

- Minimal cut sets: (1, 2, 3)

However there are four basic events that can occur, but let us assume *Basic Event 4* has a large enough MTTF and the time series does not contain any account of it occurring. This in turn means *Basic Event 4* is not reflected in the minimal cut sets. By running the reconstruction algorithm we produce a reconstructed fault tree however the original fault tree is entirely different (see Figure 5.1).

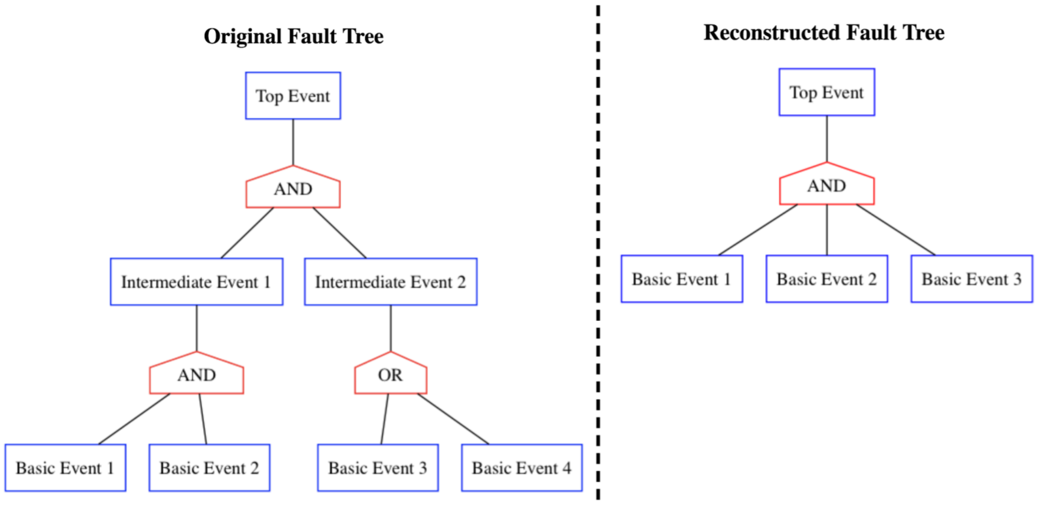


Figure 5.1: Incorrect reconstruction of fault tree

5.1.1.2 Irrelevant Events

Not only missing information, but too much information can disrupt the algorithm. If time series includes events which have no influence over the top event, it can produce false minimal cut sets and distort the resulting fault tree.

5.1.1.3 Delay Times

The reconstruction algorithm does not take into account any delay in the propagation of events in the fault tree. For example, a set of events occur at *time* = *20*, which cause a system failure. However, the system stays operational until the top event occurs at *time* = *22*. This means the system has a delay of *2* time units. The algorithm assumes the propagation of events is instantaneous.

In a real-life scenario, a system is very specific to the application on how long it takes for the events to influence the top event. The delay could be compensated for, but specific knowledge of the system is required, since the time series would not hold this information.

5.1.1.4 Complex Fault Trees

The reconstruction algorithm has not been tested on complex fault trees that have more than three layers of gates and more than eight basic events.

5.1.2 Quantitative Analysis

In this section, we discuss the limitations of the quantitative analysis techniques used and how it could be improved.

5.1.2.1 Goodness of Fit Testing

The quality of the distribution fitting is not empirically evaluated. A goodness of fit test, like the Pearson's chi-squared test could be used to determine how well the distribution fits the data. The distribution fitting could be improved by considering other techniques of fitting such as using a Q-Q plot. More distributions can be added which are tested for fitting the data.

5.1.2.2 Proxel-Based Method

The proxel-based method successfully represents the instantaneous availability of a component or system. A limitations of the proxel-based method stemmed from slow computational time of the implementation. The implementation was done using the *Python* programming languages, which is none to be slower than other languages such as *C, C++, etc*. The implementation shows the feasibility of using proxel-based simulation for visualizing the instantaneous availability of a reconstructed fault tree however it should be implemented on a different platform to be used more efficiently.

5.2 Software Testing

Unit testing is implemented on certain modules of the software. Unit testing is done on some of the methods responsible for the logic gate operations of time series and boolean data. It is also conducted on some of the methods used for the fault tree reconstruction algorithm. Unit testing insures the methods give the proper output for appropriate inputs. More testing could be done on the other modules to insure quality of the software.

5.3 Case Study

A case study on real-life data could be done, unfortunately it was out of the scope of this thesis. Case study could be conducted to assess the method to see how it generates a fault tree from time series to determine the behavior of the system and to calculate reliability, maintainability, etc. This is would be the main benefit of the reconstruction algorithm, however the feasibility of such a method had to be demonstrated first, which is the goal of this thesis.

5.4 Discussion

The thesis offers a proof of concept about fault tree extraction from time series data. It shows it is feasible to measure the reliability, maintainability, and availability of the reconstructed fault tree. The benefits of such a method lie in using it on time series data from a system where reliability is unknown and investigating the extracted fault tree to discover the reliability.

The minimal cut sets calculated during the algorithm help determine critical components in the system. It pin points where bottlenecks are in the fault tree. Combining this with calculating the reliability of components, weak components could be determined and replaced with more reliable ones to increase the overall reliability of the system.

The reconstruction algorithm can also be used for preventive maintenance in systems. By determining the reliability and failure rate of components, more optimal maintenance schedules can be put in place for replacing a component before it fails. This would increase the maintainability of the system therefore also increasing the availability [10].

The method I present is feasible approach to learning the behavior of a system from data, however the complexity of systems can hinder the performance of the reconstruction as there is still many aspects to consider that were discussed in the limitations of the algorithm.

Chapter 6

Conclusion

This chapter concludes my thesis. I summarize the work I have done and provide some direction for future work on the project.

6.1 Summary

Fault tree analysis and other reliability models help describe the behavior of a system and determine how reliable they are. Reliability is one of the most underlying quality for a system to perform properly. Accurate prediction of reliability of an operational system would help schedule maintenance and replace unreliable components. I developed a method to construct a a fault tree from the time series data of any arbitrary system.

To test the plausibility of the method, I constructed a framework around it to be able design fault trees that could generate time series data representing time-of-failures and time-of-repairs for both basic components and the entire system. Using the time series data I conduct qualitative and quantitative analysis techniques to reconstruct the fault tree. I evaluate the quality of the reconstruction by comparing the original fault tree with the reconstructed fault tree. I compare both structural aspects of the fault trees and the reliability and maintainability distributions of the individual components. I determine the reliability and maintainability of the system. Using the proxel-based method I also determine the instantaneous availability of the components and the system.

The entire reconstruction process is a data-driven modeling technique which models the behavior of a system with a fault tree. The algorithm constructs a white-box model, since the fault tree allows us to understand how the individual basic events cause the top event to occur. By using this method the behavior of a system can be determined by analyzing time series data.

6.2 Future Work

Directions for future work include improving the reconstruction algorithm by reducing its limitation that were mentioned in the previous chapter. The algorithm can be made to take into account delay times in the system and the distribution fitting can be improved. The algorithm can be implemented using another programming language that has a faster computation time in order to utilize the full benefits of the proxel-based method. A case study can be conducted on sensor data collected from a real-life system to determine the resulting fault tree by using the reconstruction algorithm. Fault tree analysis still has many features beyond the scope of this thesis which can also be explored, including dynamic fault trees and fault tree diagnosis.

Bibliography

[1] E. Ruijters and M. Stoelinga, “Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools,” *Computer Science Review*, vol. 15-16, pp. 29 – 62, 2015.

[2] V. B. Berikov, “Fault tree construction on the basis of multivariate time series analysis,” in *Proceedings. The 8th Russian-Korean International Symposium on Science and Technology, 2004. KORUS 2004.*, vol. 2, pp. 103–106 vol. 2, June 2004.

[3] D. Solomatine, L. See, and R. Abrahart, *Data-Driven Modelling: Concepts, Approaches and Experiences*, pp. 17–30.
Berlin, Heidelberg: Springer Berlin Heidelberg, 2008.

[4] H. WANG, T.-Y. CHAI, J.-L. DING, and M. BROWN, “Data driven fault diagnosis and fault tolerant control: Some advances and possible new directions,” *Acta Automatica Sinica*, vol. 35, no. 6, pp. 739 – 747, 2009.

[5] K. D. Rao, V. Gopika, V. S. Rao, H. Kushwaha, A. Verma, and A. Srividya, “Dynamic fault tree analysis using monte carlo simulation in probabilistic safety assessment,” *Reliability Engineering & System Safety*, vol. 94, no. 4, pp. 872 – 883, 2009.

[6] W. S. Lee, D. L. Grosh, F. A. Tillman, and C. H. Lie, “Fault tree analysis, methods, and applications #2013; a review,” *IEEE Transactions on Reliability*, vol. R-34, pp. 194–203, Aug 1985.

[7] S. Mukherjee and A. Chakraborty, “Automated fault tree generation: Bridging reliability with text mining,” in *2007 Annual Reliability and Maintainability Symposium*, pp. 83–88, Jan 2007.

[8] P. Angelov and R. Buswell, “Automatic generation of fuzzy rule-based models from data by genetic algorithms,” *Information Sciences*, vol. 150, no. 1, pp. 17 – 31, 2003.
Recent Advances in Soft Computing.

[9] M. Tabesh, J. Soltani, R. Farmani, and D. Savic, “Assessing pipe failure rate and mechanical reliability of water distribution networks using data-driven modeling,” *Journal of Hydroinformatics*, vol. 11, no. 1, pp. 1–17, 2009.

[10] *System Analysis Reference Reliability, Availability, and Optimization.*
ReliaSoft Corporation, 2015.
`http://reliawiki.com/index.php/System_Analysis_Reference`.

[11] *Life Data Analysis Reference.*
ReliaSoft Corporation, 2015.
`http://reliawiki.org/index.php/Life_Data_Analysis_Reference_Book`.

[12] *Introduction to Reliability.*
Portsmouth Business School, 2012.
`http://woodm.myweb.port.ac.uk/q/reliability.pdf`.

[13] L. Xing and S. V. Amari, *Fault Tree Analysis*, pp. 595–620.
London: Springer London, 2008.

[14] B. Narasimhan, “The normal distribution,” Jul 1996.
`http://statweb.stanford.edu/~naras/jsm/NormalDensity/NormalDensity.html`.

[15] A. Volkanovski, M. Čepin, and B. Mavko, “Application of the fault tree analysis for assessment of power system reliability,” *Reliability Engineering & System Safety*, vol. 94, no. 6, pp. 1116 – 1127, 2009.

[16] N. Viswanadham and T. L. Johnson, “Fault detection and diagnosis of automated manufacturing systems,” in *Proceedings of the 27th IEEE Conference on Decision and Control*, pp. 2301–2306 vol.3, Dec 1988.

[17] M. Pandey, “Fault tree analysis,” *CIVE 240 - Engineering and Sustainable Development*.
`https://www.slideshare.net/elsonpaul11/fault-tree-analysis-13114427`.

[18] R. Publishing, “Minimal cut sets,” *weibull.com – Free Data Analysis and Modeling Resources for Reliability Engineering*.
`http://www.weibull.com/hotwire/issue63/relbasics63.htm`.

[19] M. Allen, A. Spencer, and A. Gibson, “Chapter 5, what is discrete event simulation, and why use it?,” *Right cot, right place, right time: improving the design and organisation of neonatal care networks – a computer simulation study*, vol. 3, p. 9, May 2015.

[20] S. Lazarova-Molnar, *The proxel-based method: formalisation, analysis and applications*.
PhD thesis, 2005.

[21] R. Publishing, “Availability and the different ways to calculate it,” *weibull.com – Free Data Analysis and Modeling Resources for Reliability Engineering*.
`http://www.weibull.com/hotwire/issue79/relbasics79.htm`.

[22] c0fec0de, “anytree: Powerful and lightweight Python tree data structure,” 2017.
`https://github.com/c0fec0de/anytree`.

[23] M. Hieber, “Positive and negative logic,” *CS281 Introduction to Computer Architecture*.
`http://sce2.umkc.edu/csee/hieberm/281_new/lectures/basic-electrical-components/pos-neg-logic.html`.

[24] “Statistical functions (scipy.stats),” *NumPy Reference - NumPy v1.13 Manual*.
`https://docs.scipy.org/doc/scipy/reference/stats.html`.

[25] M. Waskom, “seaborn: statistical data visualization,” 2017.
`https://seaborn.pydata.org/index.html`.