Dependency modelling using fault-tree and cause-consequence analysis

This item was submitted to Loughborough University's Institutional Repository by the/an author.

Additional Information:

- A Doctoral Thesis. Submitted in partial fulfillment of the requirements for the award of Doctor of Philosophy of Loughborough University.

Metadata Record: https://dspace.lboro.ac.uk/2134/7350

Publisher: © Louise May Ridley

Please cite the published version.
This item is held in Loughborough University’s Institutional Repository (https://dspace.lboro.ac.uk/) and was harvested from the British Library’s EThOS service (http://www.ethos.bl.uk/). It is made available under the following Creative Commons Licence conditions.

For the full text of this licence, please go to:
http://creativecommons.org/licenses/by-nc-nd/2.5/
Dependency Modelling Using Fault Tree
And
Cause-Consequence Analysis

by

Louise May Ridley

A Doctoral Thesis
submitted in partial fulfilment of the requirements for the award of
Doctor of Philosophy of Loughborough University

April 2000

© by Louise May Ridley, 2000
ABSTRACT

The technique of fault tree analysis is commonly used to assess the probability of failure of industrial systems. During the analysis of the fault tree the component failures are assumed to occur independently. When this condition is not satisfied alternative approaches such as the Markov method can be used. Constructing the Markov representation of a system is not such as intuitive process for engineers as fault tree construction since the state-transition diagram does not readily document the failure logic. In addition to this the size of the Markov diagram increases rapidly as the number of components in the system increases.

This thesis presents the development of a new model which uses a combination of conventional fault tree methods with those of Markov methods to solve systems containing sequential or standby failures. New gates were developed in order to incorporate the dependent failures on the fault tree structure. The new assessment method was shown to efficiently solve these systems. With these extended fault tree capabilities in place the technique was embedded within an optimisation framework to obtain the best system performance for systems containing standby failures.

Sequential failures can be represented on a fault tree by using the Priority-And gate, however they can also be represented on a Cause-Consequence diagram. As with the fault tree analysis method, the Cause-Consequence Diagram method documents the failure logic of the system. In addition to this the Cause-Consequence Diagram produces the exact failure probability in a very efficient calculation procedure and has significant implications in terms of efficiency for static systems. Construction and analysis rules were devised for a cause-consequence diagram and used on systems containing independent and dependent failures.

KEY WORDS: Fault Tree Analysis, Markov Analysis, Cause-Consequence Diagram Method, Binary Decision Diagrams, Dependency Modelling, Sequential failures, Standby Failures, Genetic Algorithms
I would like to give my sincere thanks to Dr. John Andrews, my supervisor, who has provided me with invaluable support, guidance and friendship throughout the course of my PhD. I would also like to extend my gratitude to Dr. Joe Ward, for his good humour, and excellent proof reading. Lastly I would like to thank my friends and parents, Valerie and Norman Craske, for their unconditional support and love.
# CONTENTS

## NOTATION

## CHAPTER 1 INTRODUCTION TO RELIABILITY ANALYSIS

1.1 Introduction .................................................. 1
1.2 What is Reliability ? ........................................ 2
1.3 The History of Reliability ................................... 3
1.4 Obtaining the Reliability of a System ................. 4
1.5 Summary of Available Analysis Tools ................. 5
   1.5.1 Independent Modelling Tools ......................... 6
       1.5.1.1 Fault Tree Analysis ................................ 6
       1.5.1.2 Binary Decision Diagrams ....................... 8
   1.5.2 Dependent Modelling Tools ......................... 12
       1.5.2.1 Markov Analysis ................................ 12
       1.5.2.2 Petri Net Approach ............................. 15
   1.5.3 Consequence Identification Tools ................ 17
       1.5.3.1 Event Tree Analysis .......................... 17
       1.5.3.2 Cause-Consequence Analysis ................... 19
1.6 Summary of Analysis Methods ............................ 23
1.7 Scope of Thesis ............................................. 23

## CHAPTER 2 FAULT TREE ANALYSIS

2.1 Introduction .................................................. 25
2.2 Background of FTA .......................................... 25
2.3 Fault Tree Symbols ......................................... 26
2.4 Qualitative Analysis of a Fault Tree Structure ........ 27
   2.4.1 Obtaining the Reduced Boolean Expression ....... 31
2.5 Quantitative Analysis of a Fault Tree Structure ....... 33
   2.5.1 Component Failure Modes ............................ 34
       2.5.1.1 The Failure Process ............................ 34
       2.5.1.2 The Repair Process ............................ 36
       2.5.1.3 The Life of a Component: The Whole Process 37
   2.5.2 Calculating The Unavailability of a Component .... 41
   2.5.3 Calculating Minimal Cut Set Unavailability and Unconditional Failure Intensity 43
   2.5.4 Calculating System Unavailability .......................... 44
   2.5.5 Calculating System Unconditional Failure Intensity 47
2.6 Initiating and Enabling Events .......................... 47
2.6.1 Quantification using Initiator and Enabler Theory 48
2.6.2 Example System Containing Initiator and Enabler Events 48
2.7 Summary and Conclusion 53

CHAPTER 3 BINARY DECISION DIAGRAMS

3.1 Introduction 54
3.2 The Development of the BDD Approach 54
3.3 Application of the BDD approach to a Reliability Environment 58
  3.3.1 Conversion of a Fault tree Structure to a BDD Using the Structure Function 59
  3.3.2 Conversion of a Fault Tree Structure using an If-Then Else Approach 61
3.4 Qualitative Analysis of a BDD 63
3.5 Quantitative Analysis of a BDD 64
3.6 Summary and Conclusion 64

CHAPTER 4 MARKOV METHODS

4.1 Introduction 66
4.2 The Markov Process 66
4.3 Construction of the State Transition Diagram 68
4.4 Different Types of Markov Models 70
  4.4.1 Discrete State- Discrete Time Markov Model 71
  4.4.2 Discrete State- Continuous Time Markov Model 72
4.5 Quantitative Analysis of a Markov Diagram 73
  4.5.1 Obtaining the Probability of Residing in State i at time t 74
  4.5.2 Steady-State Solution for a Markov Diagram 76
  4.5.3 Transient Solution for a Markov Diagram 82
4.6 Conclusion 83

CHAPTER 5 CAUSE-CONSEQUENCE DIAGRAM METHOD

5.1 Introduction 85
5.2 The Cause-Consequence Diagram Method 86
  5.2.1 Symbols for Construction 88
  5.2.3 Rules of Construction for a Cause-Consequence Diagram 90
5.3 Construction Example 92
5.4 Quantitative Analysis of a Cause-Consequence Diagram 95
  5.4.1 Quantification of a System Containing Independent Failure Events 96
  5.4.2 Quantification of a System containing Dependent Failure Events 97
5.5 Further Applications of the Cause-Consequence Diagram Method 105
CHAPTER 6 ANALYSIS OF SYSTEMS WITH DEPENDENT FAILURE EVENTS

6.1 Introduction 107
   6.1.2 Fault Tree Analysis Program 108
6.2 Sequential Failures 113
   6.2.1 Construction of the Markov Diagram for a PAND Gate 116
   6.2.2 Quantification of a Markov Diagram for a PAND Gate 124
   6.2.3 Modification to the Sequential Analysis Procedure 127
6.3 Combination of Independent and Dependent Failures 131
   6.3.1 Identification of Independent Subtrees 132
6.4 Summary 138
6.5 Standby Failures 139
   6.5.1 The New Standby Gate 140
   6.5.2 Construction of the Markov Diagram for a System with Standby Components 143
   6.5.3 Quantification of the Markov Diagram for a System with Standby Components 152
6.6 Application of the Combined Model to an Industrial System with Standby Failures 153
   6.6.1 Fault Tree Construction for the Deluge System 154
   6.6.3 Analysis of the Deluge Pump System 158
   6.6.4 Construction of the Markov Diagram for the Deluge Pump System 159
   6.6.5 Quantification of the Markov Diagram for the Deluge System 164
6.7 Combination of Standby Gates with other Fault Tree Symbols 167
6.8 Summary 168
6.9 Conclusions 169

CHAPTER 7 OPTIMISATION OF A DEPENDENT SYSTEM

7.1 Introduction 170
7.2 Representation of Design Alternatives 170
7.3 Optimisation Techniques 174
7.4 Genetic Algorithms 175
   7.4.1 Background 175
   7.4.2 Methodology 177
      7.4.2.1 The Representation Scheme 177
      7.4.2.2 Fitness Measure 178
      7.4.2.3 Controlling Parameters 178
      7.4.2.4 Performance Measures and Termination Criteria for a GA 181
7.5 Application to an Industrial System 181
7.5.1 Deluge Pump System 181
7.5.2 Component Unavailability 183

7.6 The Optimisation Problem 184
7.6.1 Fault Tree Representation of the Deluge Pump System 184
7.6.2 Binary String Coding 186
7.6.3 Decoding a Design String 187
7.6.4 Objective Function Value of a String 189
7.6.5 Convergence Criteria for the GA 190

7.7 Exhaustive Search 192

7.8 Genetic Algorithm Implementation 192
7.8.1 The Feasible Region Search 192
7.8.1.1 Results and Conclusions for the Feasible Region Search 194
7.8.2 The Penalty Function Approach 198
7.8.2.1 The Cost Constraint 199
7.8.2.2 The Maintenance Constraint 200
7.8.2.3 The Overall Fitness Value of a String 200
7.8.2.4 Results and Conclusions for Penalty Function Approach 201

7.9 Summary of Conclusions 205

CHAPTER 8 APPLICATION OF THE CAUSE-CONSEQUENCE DIAGRAM METHOD TO STATIC SYSTEMS

8.1 Introduction 206
8.2 Application to Static Systems 206
8.2.1 Symbols used for Construction 206
8.2.2 Rules for Construction 207
8.3 Example 1: Three Component System 210
8.3.1 Repeated Events 212
8.3.1.1 Example System with Repeated Events 213
8.4 Algorithm for Restructuring Cause-Consequence Diagram With Common Failure Events 218
8.4.1 Scanning the Cause-Consequence Diagram 218
8.4.2 Common Failure Event Identification and Extraction 222
8.4.3 Removal of Common Failure Event from Fault Tree Structures 225
8.5 Algorithm for Construction and Analysis for a Static System 229
8.6 Application to an Industrial System 230
8.6.1 Cause-Consequences Diagram Construction and Extraction 233
8.6.2 Analysis 243
8.7 Comparison to the Binary Decision Diagram 243
8.8 Advantages of New Algorithm 245
CHAPTER 9 APPLICATION OF THE CAUSE-CONSEQUENCE DIAGRAM METHOD TO A MULTI-PHASE SYSTEM

9.1 Introduction 258
9.2 System Description 258
9.3 Cause-Consequence Diagram Construction 260
9.4 Algorithm for Reconstructing a Cause-Consequence Diagram which contains Inconsistent Failure Events. 270
  9.4.1 Identification and Manipulation of Inconsistent Failure Events 271
9.5 Application to Pressure Tank System 276
9.6 Summary and Conclusion 277

CHAPTER 10 CAUSE-CONSEQUENCE ANALYSIS OF DEPENDENT SYSTEMS

10.1 Introduction 281
10.2 Cause-Consequence Diagram development using the Time Delay Symbol 281
  10.2.1 Cause-Consequence Diagram Construction 284
10.3 Cause-Consequence Diagram Analysis 287
  10.3.1 Consequence 1: FIRE 288
  10.3.2 Consequence 2: Unmitigated Explosion 288
  10.3.3 Consequence 3: Mitigated Explosion 290
10.4 Conclusion 292

CHAPTER 11 CONCLUSIONS AND FUTURE WORK

11.1 Summary of Work 294
11.2 Conclusions 297
11.3 Future Work 299
  11.3.1 Modelling of Different Dependent Failures 299
NOTATION

\( \Lambda(t) \) Availability at time \( t \)
\( Q(t) \) Unavailability at time \( t \)
\( F(t) \) Cumulative Failure Distribution, Unreliability
\( f(t) \) Probability Density Function of failure distribution
\( G(t) \) Cumulative Repair Distribution
\( g(t) \) Probability Density Function of repair distribution
\( dt \) small time interval
\( w(t) \) Unconditional Failure Intensity
\( v(t) \) Unconditional Repair Intensity
\( \lambda(t) \) Conditional Failure Intensity
\( \mu(t) \) Conditional Repair Intensity
\( W(0, t) \) Expected number of failures in the interval \( 0 \) to \( t \)
\( V(0, t) \) Expected number of repairs in the interval \( 0 \) to \( t \)
\( M_{Ci} \) Minimal Cut Set \( i \)
\( Q_i \) or \( q_i \) Failure Probability of \( i \)
\( - \) Complement
\( G_i(q) \) Birnbaum's Criticality Function. The probability that the system is in a critical state for component \( i \)
\( \phi(x) \) Structure Function
\( P(i) \) Probability of \( i \) occurring
\( MTTR \) Mean time to repair
\( MTBF \) Mean time before failure
\( MUT \) Mean up time
\( Ft_i \) Fault tree \( i \)
\( \theta \) Maintenance Interval
\( \tau \) Mean time to repair
\( H_i \) House Event \( i \)
\( \lambda \) Constant Failure Rate
\( \mu \) Constant Repair Rate
CHAPTER 1

INTRODUCTION TO RELIABILITY ANALYSIS

1.1 Introduction

An accident is the result of a coincidence of events and/or failures of individual components. Industrial accidents can bring about different outcomes depending on their severity and may result in loss of profit, injury and in the worse case scenario one or several fatalities. During the last twenty years a number of significant accidents have occurred which have given rise to increased public concern. Discussed are two examples which are used to highlight the need for reliability analysis. On April 26th 1986 the most traumatic event in the history of civil nuclear power occurred at the Chernobyl site (ref. 1). The accident was triggered by a turbogenerator experiment, carried out when the reactor core contained water just below boiling point. At the start of the experiment half of the main coolant pumps slowed down which caused the water in the reactor core to boil. Due to “positive feedback” the power to the reactor increased which in turn caused the fuel to heat-up, melt down and disintegrate. Fragments of the fuel were ejected causing a steam explosion which led to the pile cap being blown off. Radionuclides escaped into the atmosphere where the wind carried them to nearby towns and cities as well as distant countries. Malpractice by the operators was later announced as an additional cause and safety systems were identified as being inadequate (ref. 2). The result of the accident was catastrophic with 31 deaths, many caused by fighting the fires caused by the accident, and 200 cases of acute radiation. The accident also incurred a financial loss, as a vast amount of money was spent evacuating and relocating 135,000 people from neighbouring areas and on distributing iodine tablets to millions. Today 30 sq. km of land is still limited in use and deaths due to radiation may still occur. In this example the effect of an inadequate design and no safeguard against operator error resulted in loss of life, money and land.
Closer to home, Britain has witnessed several horrific train accidents which have highlighted the need for tighter safety. The most recent accident involved the collision of two trains just outside Paddington station. At 8.11 a.m. on Tuesday 5th October 1999 a Thames Train collided with a Great Western Train. The Thames Train, travelling out of Paddington station, crossed the tracks from right to left in front of the Great Western which was travelling towards the station. The Thames train attempted to move out of the path of the Great Western but the two collided almost head on. The front of the Thames train was tossed in the air and ripped apart. The middle and rear carriages caught fire and the first class carriage of the Great Western was burnt to its shell. The accident resulted in 31 fatalities and several injuries ranging from cuts to severe burns. The cause of the accident was stated to be that the driver of the Thames train had passed signal ‘109’ when it had been red. It was disclosed that the Great Western had been given a green signal. Following an investigation into the cause of the accident it was reported that signal ‘109’ had been obscured by a pylon carrying overhead electric wires and was therefore very difficult to see. In addition to this an inappropriate safety system had been installed on the train to deal with the case of a red light being passed. Due to poor visual signals, an inadequate safety design, which allowed red lights to be passed, 31 people died and a train company lost money as well as its reputation.

The motivation for the analysis of industrial systems is thus to aid in the protection of important facilities and to help reduce the risk of such adverse events.

1.2 What is Reliability?

A manufacturer’s primary concern for their product is that it serves every user. Many modern products are complex constructions comprising many interrelating parts. This complexity creates a problem in how to prolong, predict and guarantee a product’s life. Reliability is the main discipline that has developed to address this concern, with the overall strategy to identify weaknesses or events which prevent the product from functioning as desired. Reliability is used as a measure of the system’s success and is defined, by British Standards, as:
"the ability of an item to perform the required function under stated conditions for a stated period of time"

1.3 The History of Reliability

Until the application of modern scientific theory, technical progress was achieved by the process of trial and error. As manufacturing technology increased, so the relationship between safety and reliability intensified. With the dawn of the Industrial Revolution new sources of power gave great potential for rapid development but also provided a proportional increase in the potential for death and injury. Early methods of analysis were purely deterministic, where structures were designed to a predetermined load factor, commonly referred to as the ‘factor of safety’ (ref. 3). As the complexity of systems increased the application of the factor of safety became inappropriate and towards the end of the 19th century statistical and probabilistic approaches were utilised.

Reliability practices can be traced back to the late 1940’s, early 1950’s, especially in the area of air-traffic control. At this time jobs that were previously performed manually, such as the control of take-off and landing, were now being performed by newly developed electronics. It was recognised that with such complex equipment it was not enough to expect the equipment to just meet satisfaction on delivery, it had to perform when needed. In addition to this, at different times failure of the equipment could result in consequences that were potentially more catastrophic than at other times. The equipment was therefore required to perform when needed with a small likelihood of failure, especially in potentially hazardous situations. The increase in the size and complexity of civil aircraft meant that the number of components that could fail increased and due to an increased passenger capability the number of lives at risk also magnified. The US Department of Defense (DOD) and the Federal Aviation Authority (FAA) recognised the need to establish reliability requirements and in 1952 the DOD funded a study into reliability issues.

In 1957 the Advisory Group on Reliability of Electronic Equipment (AGREE), funded by the DOD, issued a report which became the basis of reliability engineering (ref. 4).
At the same time the Air Force also established a study on the reliability of electronic components at the Rome Air Development Center (RADC) in New York. The databases produced aided in the development of the military standard, MIL-STD-217, which is commonly used today for reliability predictions. The military standard, MIL-STD-785, developed in 1965 evolved from the AGREE report.

The AGREE and RADC concentrated entirely on the reliability of electronic components. However, other organizations soon developed handbooks for reliability prediction of non-electronic components. During the 1970’s the newly established Nuclear Regulatory Commission (NRC) published a set of standards for nuclear power plants and in 1980 the Food and Drug administration commenced the regulation of all medical devices.

Using the reliability prediction handbooks the occurrence of an undesired event can be estimated. In addition to this, weak links in the system can be identified and suggestions made on how to reduce the frequency of failure.

1.4 Obtaining the Reliability of a System

Since the early 1960's various types of mathematical models have been used to perform reliability analysis (ref. 5) in order to predict the likelihood that a system will function given a demand. These include such techniques as: Reliability Block Diagrams, Fault Tree Analysis, Markov Analysis and Simulation. These techniques can be viewed as a continuum corresponding to the complexity of using each of the alternative methods to perform the analysis (Figure 1.1). Reliability Block Diagrams (RBD) and Fault Tree Analysis (FTA) are deemed as the least complex methods to implement for two reasons. Firstly the diagram construction for both techniques is logical and well developed and secondly both techniques assume independence between failures which can be analysed using basic probability theory. Markov analysis, Simulation and Petri Nets are more complex in that they are capable of analysing components that when failed effect the functionality of other components. These type of components are referred to as ‘dependent’ components. More sophisticated procedures are required to analyse such systems. The methods
represented in figure 1.1 show the causes of one particular outcome, other techniques exist which can illustrate all possible consequences from some initial event. Two such methods will be discussed: The Cause-Consequence Analysis method and the Event Tree Analysis method. These techniques deal with sequences of events that result in different consequences proceeding a particular initial event.

Figure 1.1 Continuum representing the complexity of available reliability techniques

1.5 Summary of Available Analysis Tools

Each analysis model has different features which make it more applicable to some systems than others and to achieve the most efficient analysis the simplest technique should be used. In order to be able to select the most appropriate method a knowledge of each technique is required. The available techniques can be segregated into different types:

1) Systems containing independent failures: Independent modelling tools.
2) Systems containing dependent failures: Dependent modelling tools.
3) Systems with varying levels of system outcome: Consequence identification tools.

This chapter aims to summarise the available techniques in terms of application highlighting their advantageous and disadvantageous features. A more in-depth description of the relevant techniques is presented in the proceeding chapters.

1.5.1 Independent Modelling Tools

1.5.1.1 Fault Tree Analysis

An underlying assumption common to both the RBD and FTA techniques is that the failure of individual components must occur independently. The RBD method
describes the system in terms of success and therefore considers the components in a working state. FTA utilises an opposite concept to that used in the RBD method. FTA describes the system in a failed state and represents the failure logic in a tree like structure. The undesired outcome being analysed is referred to as the **TOP EVENT** and the tree structure is developed by extending branches below this event. The development of the fault tree involves the use of Boolean expressions which represent the logical operators of **OR**, **AND** and **NOT**. The fault tree diagram consists of two main features: **gates** and **basic events**. The gates represent the Boolean logic of system failure and the basic events hold information regarding the failure characteristics of a component. A branch is considered complete when a basic event is encountered which represents the limit of resolution of the analysis. Figure 1.2 illustrates a simple fault tree structure containing three basic events A, B and C.

![Figure 1.2 Typical Features of a Fault Tree](image)

The top event can be represented by a Boolean expression using various techniques such as the “Top-Down” or “Bottom-Up” approach (ref. 6). For the example fault tree illustrated in figure 1.2 the Boolean expression for the top event is:

\[
\text{TOP} = A + B.C
\]

where the ‘+’ symbol represents the logical OR gate and the ‘.’ symbol represents the logical AND gate. The Boolean expression is made up of a number of terms, known as **Cut Sets** of the system. A cut set can be defined as:
'A collection of basic events such that if they all occur the top event occurs.'

The top event expression is reduced using Boolean algebra rules to produce a minimal expression which contains only **Minimal Cut Sets** of the system. A minimal cut set can be defined as:

'The smallest combination of component failures which if they all occur cause the system to fail.'

For the example fault tree in figure 1.2 no reduction is possible and the minimal cut sets can be found to be:

{A}

{BC}

meaning that the top event can be caused by the failure of component A alone OR the failure of components B AND C. The probability of failure for component i can be symbolised as \( Q_i \). The probability of failure for the example fault tree in figure 1.2 is therefore equal to,

\[
P(\text{System Failure}) = Q_A + Q_B Q_C - Q_A Q_B Q_C
\]

The minimal cut sets are utilised in the quantification of the system's failure probability and can also be used to highlight the most likely contributor to system failure.

FTA is the most commonly employed technique for reliability analysis and hence over the years a comprehensive set of rules for analysis has evolved. There are numerous advantages of the FTA technique. One of the main attractions lies within the fault tree structure itself. As the tree contains a textual description of the system failure logic it can be used as a means of communicating the model development to non-mathematicians. FTA also provides a means of system quantification, and the minimal cut sets can aid in highlighting deficiencies in the design.
The FTA method, however, also possess some disadvantages. Firstly the quantification of large systems cannot always be found exactly and in such situations approximations are used. The approximations rely on the components having a small likelihood of occurrence and when this is not the case errors can result. A second disadvantage with the FTA technique is that it is unable to deal with systems containing dependent failure events. The Kinetic Tree Theory (ref. 7), which forms the basis of FTA, requires statistical independence and violation of this assumption occurs in many systems not only in sequentially operating systems but also in standby systems and systems that are affected by common cause failures and secondary failures. FTA can be used for such systems but ignores any dependencies and assumes independence which can, in some cases, result in gross inaccuracies.

1.5.1.2 Binary Decision Diagrams

For large fault trees the identification of the minimal cut sets can prove to be difficult. In addition to this the approximation techniques used to calculate the top event probability can be inaccurate if the likelihood of component failure is not small. The problem of inaccuracies due to approximation techniques has been alleviated recently by the development of the Binary Decision Diagram (BDD) approach. BDDs are based on Bryant’s trees and obtain the exact top event probability efficiently by expressing the system failure modes as disjoint paths. The calculation of the top event probability is achieved by summing the probabilities of these disjoint paths.

A BDD is a directed acyclic graph and consists of two main features, a **non-terminal node** and a **terminal node**. Non-terminal nodes represent the components of a system and are represented on the diagram by a circle. Terminal nodes, represented by a square box on the BDD, indicate whether the system is in a working or failed state. A working state is symbolised by a ‘0’ in the terminal node and a failed state by a ‘1’. Each non-terminal node has two outlet branches which indicate whether the component is working, ‘0’ outlet branch, or failed, ‘1’ outlet branch. An example BDD is depicted in figure 1.3.
The BDD is constructed by considering the components of the system in a certain order starting from the top node (root node), which represents the first component in the variable ordering. The functionality and effect of each component on the system is then determined. The fault tree structure in figure 1.4 can be used to illustrate the construction process.

The top event expression for the example fault tree in figure 1.4 is:

\[
\text{TOP} = (A + C)(B + C) = A.B + C \text{ (following Boolean reduction)}
\]
Taking the variable ordering to be A<B<C, i.e. A before B before C, the root node contains component A. Failure of component A, represented on the ‘1’ outlet branch from the root node, reduces the top event expression to:

$$\text{TOP} = B + C$$

As both B and C are still present in the top event expression, component B is considered next. Failure of component B causes system failure and therefore the ‘1’ outlet branch terminates in the terminal node ‘1’. Given component B functions correctly the top event expression reduces to:

$$\text{TOP} = C$$

For the ‘0’ outlet branch of both components A and B the affect of failure or functionality of component C on the top event is determined. The BDD for the fault tree representation given in figure 1.4 is depicted in figure 1.5.

This process of construction is trivial for such a small system. For larger systems, though, this approach would be inefficient. Construction of the BDD from the fault tree diagram is generally performed using the IF-THEN-ELSE (ite) structure, which is discussed in detail in Chapter 3.

![BDD for Fault Tree Structure in figure 1.4](image-url)
The cut set identification of the system is obtained by determining the various paths, from the root node, which terminate in a '1'. Only the basic events that lie on a '1' outlet branch, indicating component failure, are included in the cut set. The cut sets for the BDD in figure 1.5 are therefore:

\{AB\}

\{AC\}

\{C\}

To obtain the minimal cut sets either the resulting list of cut sets is reduced using Boolean algebra or the BDD can undergo a minimisation process. However for the purpose of quantification the non-minimal BDD should be used.

The Top Event probability is obtained by summation of the probability of each sequence path that terminates in a '1'. For component i that lies on a '0' outlet branch, the probability of travelling down such a branch is given by \(Q_i\) which is equal to \((1 - Q_i)\). The top event probability for figure 1.5 is therefore equal to:

\[
P(TOP) = Q_A Q_B + Q_A \overline{Q_B} Q_C + \overline{Q_A} Q_C
\]

\[
= Q_A Q_B + Q_A (1 - Q_B) Q_C + (1 - Q_A) Q_C
\]

\[
= Q_A Q_B + Q_C - Q_A Q_B Q_C
\]

which can be shown to be identical to the exact top event probability that would be obtained via FTA.

This analysis procedure makes the BDD technique more efficient than the traditional FTA technique. The main advantages of the BDD method are thus that it produces the exact top event probability and reduces the amount of computing time if the ordering of the variables is efficient. In addition to this, as the BDD is created from the fault tree structure, the textual description of the system failure logic is not lost.

A disadvantage of the BDD method is that it is impractical to construct the diagram directly from the system description and it must be generated from the fault tree structure. As the BDD is created from the fault tree any limitations with the FTA technique will transfer to the BDD and limit the techniques ability. For example the
assumption of independence is apparent in the BDD analysis routines implying this technique is unable to accurately analyse systems containing dependent failure events.

1.5.2 Dependent Modelling Tools

If independence is an unrealistic assumption then more sophisticated techniques are required.

1.5.2.1 Markov Analysis

Markov models do not require the assumption of independence and can therefore be used to analyse systems containing dependencies, such as common cause failure and standby failures. The Markov diagram consists of two features, states and transitions between states. This modelling technique describes the system in a state-transition diagram. The states in the diagram represent the mutually exclusive and exhaustive states of the system. Each of these will correspond to either a working or failed state for the system. The individual components themselves, that are contained within each state, can represent different states of functionality, e.g. they can represent a working component, a failed component, a component that is in standby or a component undergoing repair. The states are represented by a circle on the Markov diagram and the transitions are illustrated by arcs between states.

The construction of the Markov diagram begins with the identification of the initial state of the system at time $t=0$. The initial state is generally the working state where each component is assumed to be functioning as desired. Therefore for a system comprising of two components, A and B that can be either working or failed, the initial state would be, A working, B working. This initial state can be seen as state 1, $S_1$, in figure 1.6, where ‘0’ represents a working component and ‘1’ a failed component. The remainder of the diagram is developed by creating new states due to all possible transitions out of any developed state, where only one transition (failure or repair) is considered at a time. Two new states can be developed from $S_1$ due to either the failure of component A or component B. Failure of component A results in the transition to state 2, $S_2$, where A is failed and B is working. The transition rate from $S_1$ to $S_2$ is equivalent to the failure rate of component A, $\lambda_A$. Similarly state 3,
S3, is developed from the initial state due to the failure of component B, represented on the Markov diagram by $\lambda_B$. The available transitions out of states 2 and 3 are then considered and the Markov diagram is deemed complete when no further transitions out of any newly developed states can be identified. The complete Markov diagram is given in figure 1.6 where $\mu_i$ represents the repair rate of component $i$.

Figure 1.6 Markov Representation of a 2-component System

The probability of system failure is calculated by determining the probability of residing in any state that represents system failure. The probability of residing in state $i$ at time $t$ can be achieved in numerous ways, depending on whether the system is discrete or continuous with respect to time, and is discussed in-depth in chapter 4. The identification of the failed states is problem dependent. For example, if components A and B were in a series network then the failed states in figure 1.6 would be S2, S3, and S4, as the system fails if either component A or B have failed.

Dependencies are modelled on the Markov diagram due to its ability to represent different functionality of each component by using appropriate states and transitions between these states. Boyd (1996) outlined that the primary advantage of the Markov method was that of its flexibility in expressing dynamic system behaviour (ref. 8). It has, however, several drawbacks. Primarily, the development of the diagram involves no logical explanation of the system. This decreases the communication value of the
diagram and can also result in diagrammatic errors for large systems which are
difficult to check or detect. A further disadvantage with the state-transition diagram is
that it grows rapidly as the number of components in the system increase. For a
component that can be in one of two states, either working or failed, the number of
states in the Markov diagram is equal to \(2^n\), where \(n\) is equal to the number of
components in the system. Therefore for a system containing only ten components the
Markov diagram would consist of 1024 states. Not only does the number of
components in the system increase the size of the diagram and therefore affect the
computation time, it also increases the complexity of the actual construction. As a
result of the advantages and disadvantages of the Markov method it has been
suggested that the technique is most useful when applied to systems containing
dependencies and of limited size, less than 5,000-10,000 states computer generated or
less than 50 if manually constructed (ref. 8).

A common assumption between the Markov technique and the FTA and BDD
methods is that the failure and repair rates for each component must be constant,
therefore failure or repair rates cannot vary with time. This assumption is not always
valid and in such situations a more complex technique, such as simulation, would be
required.

1.5.2.2 Petri Net Approach

Petri nets were introduced in 1962 by C.A Petri. Although originally designed to deal
with systems exhibiting concurrent behaviour they have over the last 10 years been
shown to be, among other things, particularly useful for analysing systems that contain
dependencies. The Petri net models two aspects of system performance, namely
\textbf{events} and \textbf{conditions}, on a bipartite directed multigraph. The Petri net is composed
of four parts, a set of \textbf{places} (\(P\)) representing a condition, a set of \textbf{transitions} (\(T\))
representing an event, an \textbf{input function} (I) and an \textbf{output function} (O). An example
of the Petri net structure is depicted in figure 1.7.
The dynamic behaviour of the system is modelled on the Petri net by the use of **markings**. A marking vector, $\mu$, is an assignment of tokens to the places of a Petri net. The tokens are passed from place to place to represent the change of state within the system. A token is represented on the Petri net by a dot present in a place node and can be transferred to another place once a transition is enabled. A transition can be defined as being enabled when all input places that are connected to a transition contain at least one token. An example of a marked Petri net is given in figure 1.8 and $\mu$ is equal to:

$$\mu = (1,0,0,0)^T$$

It can been seen that transition T1 is enabled. The state of the Petri net is therefore defined by its marking and any change in marking results in a change of state.
Following the execution of T1 the token present in place P1 will be removed and deposited in each of the output places of T1, i.e. P2, P3 and P4.

Analysis of the Petri net is achieved, almost always, by finding a finite representation for the reachability set of a Petri net. The reachability set can be defined as:

"the set of all states into which the Petri Net can enter by any possible execution" (ref. 9)

The finite representation of the reachability set is generally portrayed as a reachability tree whose nodes represent marking, µi, and the arcs represent the firing of a transition, Ti. The reachability tree, if identifiable, is then used to analyse the Petri net. In addition to this Peterson's next-state function (ref. 9) or Murata's matrix method (ref. 10) can be used to analyse the behaviour of a Petri net.

A fault tree structure can also be converted into a Petri net by conversion of each gate into appropriate place and transition nodes. Work completed thus far has created a Petri net form for every type of fault tree gate and techniques have been devised to determine the minimal cut sets and path sets from the bipartite graph. The probability of system failure can therefore be obtained directly from the Petri net graph.

The most useful application of the Petri net model is to computer hardware and software. The technique is advantageous in that the diagram strongly symbolises the various links between system components. The actual construction of a Petri net from a system description is not documented which is a definite disadvantage. The Petri net provides no system failure logic description, as with the FTA method, and even though it has been suggested that the technique is as efficient as the traditional technique of FTA (ref. 11), there is no evidence to support it for the static case. The advantage of this technique, over the conventional methods, lies in its ability to model dependent and concurrent systems, although the analysis techniques used to achieve this are complicated and generally the model is converted to a Markov diagram or solved using simulation. The Petri net model could be used to model dependent failure
events but would be the least favoured method to do so, due to its confusing construction process and limited analysis capability.

1.5.3 Consequence Identification Tools

Starting from an initiating event consequence identification tools identify all possible system outcomes, regardless of whether they represent good or bad scenarios.

1.5.3.1 Event Tree Analysis

A method frequently used when investigating the responses that occur from a system given an initial potentially hazardous event is Event Tree Analysis (ETA). The method was first applied in the early 1970's and is now the main method recommended in the American guide to performance of probabilistic risk assessment for nuclear power plant safety (ref. 12). The ETA method, unlike the FTA method, works by using forward logic where the development begins from a specific failure and is followed through to trace all possible outcomes. The tree itself consists of a series of nodes and branches listed sequentially. At each node a decision is made as to whether the component or sub-system under inspection is functioning correctly or not. After all sub-systems/components have been inspected a consequence is reached. One of the advantages of this method over that of the FTA technique is that multiple consequences can be analysed.

The first step in constructing an event tree involves the identification of the ‘initiating event’, i.e. the event that requires other systems to respond. The functionality or failure of the safety sub-systems or components are then considered and all consequences determined. Each branch point can be either represented by an individual component or by a sub-system. In the event of the nodes being based on sub-systems, failure to function is generally obtained via a fault tree structure.

By assigning a probability to each branch the probabilities of every possible outcome following the initiating event can be determined. The probability of each node’s failure is given by evaluation of the appropriate fault tree, given a sub-system representation, or by the single component failure data. The probability of each
node’s success is obtained via the complement rule, where \( P(\text{success}) = 1 - P(\text{failure}) \). For a particular sequence path then, the probability is obtained by multiplying together all branch probabilities present in that path. The overall probability of any particular consequence is obtained by summation of path probabilities that produce the consequence in question. An example event tree is illustrated in figure 1.9 and the frequency of each consequence, for a system containing independent failures, can be shown to be:

\[
\begin{align*}
P(\text{Consequence 1}) &= \lambda_i \cdot (1 - Q_{S1}) \cdot (1 - Q_{S2}) \\
P(\text{Consequence 2}) &= \lambda_i \cdot (1 - Q_{S1}) \cdot Q_{S2} + Q_{S1} \cdot (1 - Q_{S2}) \\
P(\text{Consequence 3}) &= \lambda_i \cdot Q_{S1} \cdot Q_{S2}
\end{align*}
\]

where \( \lambda_i \) is equal to the frequency of occurrence of the initiating event, i.

The ETA method is especially useful for systems that have a definite response sequence to a given initiating event. The main advantage with the ETA method is that it can deal with dependent failures as well as independent failures, and is used mainly with safety systems. The main disadvantage of the technique is that as the systems become more complex so the analysis required becomes more involved. In addition to this not all systems can be described in terms of event trees directly, for example.
systems which include the same component performing different functions in the same sequence (ref. 13).

1.5.3.2 Cause-Consequence Analysis

In 1971 Nielson presented a graphical method for analysing relevant accidents in a nuclear power plant (ref. 14). The method he proposed connects the events that cause “accidents” with their relevant consequences. The method consists of the union between FTA and ETA and is generally referred to as the Cause-Consequence Diagram method. The fault tree structure represents the causes of a critical event and the event tree represents the link between the causes of the critical event and the various consequences that can result. The cause-consequence diagram is structured around the decision box symbol, illustrated in figure 1.10, which is an identical representation of the ‘YES-NO’ branches seen on the event tree structure. The decision box asks whether a certain sub-system or component is functioning as desired. A connection point between the fault tree and event tree methods can also exist at the NO outlet path of these decision boxes, as the failure causes for sub-system i are represented by a fault tree structure, Fti. In the instance that a decision box represents a single component the NO outlet path is caused by the single component failure probability, Qi. As the component or sub-system will be either working or failed the probability of travelling down the YES outlet branch is given by 1 minus the probability of travelling down the NO outlet branch.

![Decision Box Symbol for a Cause-Consequence Diagram](image)

The procedure for development of a cause-consequence diagram involves identification of an initiating or critical event. For example a hazardous event which requires rectification or the initiating event in a start-up sequence. Given a critical event certain subsystems or components will be activated. The cause-consequence
diagram is developed by evaluating the state of the system depending on whether these components function correctly or not. The simple light switch circuit system, shown in figure 1.11, can be used to illustrate the construction of a cause-consequence diagram.

The light switch circuit functions when an operator depresses the push button, which sends a power source to the bulb. The initiating event is therefore that the operator depresses the push button. Having identified the initiating event, the next stage in the construction process is to identify all possible system consequences. Following the initiating event the circuit should close causing a current to be applied to the bulb. The cause-consequence diagram is hence completed by considering the functionality of the components that control the closure of the circuit and the current through the circuit. The cause-consequence diagram for figure 1.11 is shown in figure 1.12.
The causes of the circuit failing open is that the push button fails to close the circuit, hence a single component failure probability, \( Q_{PB} \), is attached to the NO outlet branch of the first decision box. The causes of the current failing are that the battery fails to produce power, \( BAT \), the bulb has blown, \( BB \), or the fuse is broken, \( F \). These failure causes are represented on the fault tree, \( Ft_1 \), which is given in figure 1.13.

![Fault Tree Ft1](image)

**Figure 1.13 Fault Tree Ft1 for Cause-Consequence Diagram shown in figure 1.12**

Quantification of the cause-consequence diagram, for a system containing independent failures, can be achieved via multiplication of each outlets branch probability leading to a consequence. The overall probability for any particular consequence is obtained by summing all sequence probabilities that lead to the particular consequence. For example the probability of light failure, 'NL', in figure 1.12 is equal to: \( Q_{PB} + (1-Q_{PB})(Q_{FU}) \).

The cause-consequence diagram method can also be applied to systems containing dependent failure events and has been used to analyse systems where sequence of failures is relevant. Quantification of such systems requires more advanced mathematical techniques which are discussed in Chapters 5 and 9. The method has greater capabilities than some of the traditional techniques such as FTA, as it can be applied to systems containing independent and dependent failures. The development of the cause-consequence diagram is logical and shares the attractiveness of the FTA.
method as it contains a textual description of not only system failure but also all other states that the system can attain. As with the FTA method, component weaknesses can be highlighted and hence system designs modified.

The main drawback with the cause-consequence diagram method is the lack of development for a generalised analysis procedure that can be universally applied to all systems. The technique was researched intensively in the late 1970’s but since has received little attention and therefore has advanced a minimal amount in the last 20 years. The technique has definite capabilities yet requires further research in order to increase its useability to that of the traditional methods.

1.6 Summary of Analysis Methods

The main advantages and disadvantages of each analysis technique described in this Chapter are highlighted in table 1.1.

<table>
<thead>
<tr>
<th>Method</th>
<th>Main Advantage</th>
<th>Main Disadvantage</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fault tree Analysis</td>
<td>1) Very efficient at representing system failure logic</td>
<td>1) Cannot accurately analyse systems that contain statistically dependent failures</td>
</tr>
<tr>
<td></td>
<td>2) Provides a textual description</td>
<td>2) Inefficient method for large systems</td>
</tr>
<tr>
<td>Binary Decision Diagrams</td>
<td>Efficient and effective at gaining exact top event probabilities</td>
<td>1) Cannot accurately analyse systems that contain statistically dependent failures</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) Difficult to construct from system description</td>
</tr>
<tr>
<td>Petri Nets</td>
<td>Models dependent and concurrent systems. Useful for application to computer</td>
<td>Analysis procedures are undeveloped if reachability set cannot be determined</td>
</tr>
<tr>
<td></td>
<td>hardware and software systems</td>
<td></td>
</tr>
<tr>
<td>Markov Model</td>
<td>Can analyse systems containing dependent failure events.</td>
<td>1) Diagram contains no textual description</td>
</tr>
<tr>
<td></td>
<td></td>
<td>2) Diagram grows rapidly as the number of components increase</td>
</tr>
<tr>
<td>Event Tree Analysis</td>
<td>1) Can analyse a number of different accident scenarios.</td>
<td>Analysis can become complex as system increase with complexity.</td>
</tr>
<tr>
<td></td>
<td>2) Can model systems containing both independent and dependent failures.</td>
<td></td>
</tr>
<tr>
<td>Cause-Consequence Analysis</td>
<td>1) Can analyse systems containing both independent and dependent failures.</td>
<td>Construction and analysis procedure underdeveloped as no generalised procedures exist.</td>
</tr>
<tr>
<td></td>
<td>2) Retains textual description</td>
<td></td>
</tr>
</tbody>
</table>

Table 1.1 Summary of Advantages and Disadvantages
1.7 Scope of Thesis

The majority of the methods used in reliability analysis assume independence between individual component failures. FTA is the most commonly used technique and although the type of logic diagram used in this method is very good at representing the system’s failure logic, it is not necessarily an efficient or accurate quantification technique. Other techniques, such as Markov models, have the capability of dealing with dependencies yet are more complex to construct and do not give a textual description for the system’s failure logic.

The main problem in reliability analysis is that generally only a small part of any system contains components of a dependent nature. To represent the entire system using a method that can analyse dependent failures could prove to be computationally expensive. It is at times acceptable to assume that the dependencies do not excessively alter the result of quantification and for this reason can be assumed as independent. However, when the dependencies are ‘heavy’, i.e. when the correct functioning of a component is highly dependent on the functionality of another, this factor must be accounted for in the analysis in order to produce an accurate result. The problem that exists is that no method has been devised that fully documents the failure logic of the system and can analyse systems containing both independent and dependent failure events efficiently.

This thesis is concerned with the development of a modelling technique that can successfully describe and analyse systems comprising both independent and dependent parts. The new method will utilise the most advantageous parts of presently existing techniques and will attempt to solve systems with minimal amount of computational effort.

The objectives of the work program are to:

1) Review existing modelling techniques. Focus mainly on FTA and Markov methods due to their popularity but also identify capabilities of other less conventional tools.
2) Identify main deficiencies in the currently applied techniques and main features which would be retained in any future developed model.

3) Identify how dependencies can be modelled. Attention will focus on standby and sequential failures due to their relatively common occurrence in systems.

4) Develop the computer algorithm of the proposed new modelling technique to quantify the reliability of a system design where dependencies feature in combination with independent elements.

5) Thoroughly investigate the application of the proposed new modelling technique and the results obtained from the application.

6) Utilise the proposed method in the traditional design process in order to aid in the identification of an acceptable design.

7) Test the new method/s developed by analysing test case systems.

8) Apply the new technique/s to real-life systems and discuss the relative advantages/disadvantages of the method.

9) Investigate the potential of developing the method further and highlight any further work which would be advantageous.
CHAPTER 2

FAULT TREE ANALYSIS

2.1 Introduction

With the increased complexity of industrial systems, engineers need to predict system reliability in order to design safe and efficient systems. One way of achieving this is to identify the causal relationships between events, which result in system failure. Once the effect of certain components has been established the system can be redesigned in order to reduce the impact of such component failures. In the last 30 years various mathematical models have been used to identify the effect of component failures on the system’s performance. The most frequently used technique for system reliability assessment is Fault Tree Analysis (FTA) and this method has been shown to be a very effective reliability analysis tool.

This chapter aims to describe the construction and analysis procedures of the FTA method and highlight its application to the reliability environment.

2.2 Background of FTA

In 1961, the fault tree concept was introduced by Watson of Bell Telephone Laboratories in connection with a US Air force contract to study the Minuteman missile launch control system (ref. 15). In 1965 Haasl gave guidelines for the construction of fault tree models (ref. 16) and since the mid 1960’s many improvements have been completed on the FTA method. The technique is thus a well-accepted means of predicting the reliability of a system and can be used to improve the efficiency of a system by identifying critical components. The area of fault tree construction and analysis is discussed in detail in Andrews and Moss (ref. 6).

The various types of modelling tools used for reliability analysis can be broadly split into two types, those incorporating forward logic (inductive process) and those involving backward logic (deductive process). An inductive process begins at
component level and works forward to produce a list of system outcomes due to component failure. Alternatively a deductive process works backward from an undesired event to the failures of components which cause it. FTA can be seen as a deductive process as the construction process initiates from an undesired event. The fault trees are built up by expressing system failure mode in terms of combinations of individual component level events and operator actions (ref. 6). To construct a fault tree the analyst must have a detailed understanding of how the system functions. The fault tree development is systematic, going from high level to low level in the system structure.

When using FTA the undesirable outcome is labelled the 'TOP EVENT'. The fault tree is then developed by extending branches below the top event, which represent the various immediate causes of the top event. The new events are in turn developed and the tree events are continually refined until a level is reached which contains only component failures or events representing the limit of resolution for which data is available. Events found in the final level of the tree structure are known as BASIC EVENTS.

Analysis of a system may involve developing many different fault trees, one for each failure mode, as it is imperative for an effective analysis that the definition of the tree top events are not too broad or too narrow. If the top events are too broad they may involve irrelevant information that does not fit the objectives of the study. Similarly if the top events are too narrow all relevant failure information may not be present leading to an inaccurate result.

The development of the fault tree involves the use of Boolean expressions which represent the logical operators of OR, AND and NOT. The fault tree structure involves two basic elements, GATES and EVENTS.

2.3 Fault Tree Symbols

The gate types that are most commonly used in fault tree construction are shown in figure 2.1. These gates either allow or inhibit the passage of fault logic up through the
tree. If the inputs to a gate occur within the gate specification then the gate will open, if not the gate will remain closed. For example in the case of the AND gate in figure 2.1, the gate specification is that all inputs to the gate must occur for the gate to open.

<table>
<thead>
<tr>
<th>GATE TYPE</th>
<th>GATE SPECIFICATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>AND GATE:</td>
<td>Allows causality to pass up through the tree if at any time all inputs to the gate occur</td>
</tr>
<tr>
<td>OR GATE:</td>
<td>Allows causality to pass up through the tree if at any time at least one input to the gate occurs</td>
</tr>
<tr>
<td>VOTE GATE:</td>
<td>Allows causality to pass up through the tree if at least k out of N possible inputs occur</td>
</tr>
</tbody>
</table>

Figure 2.1. Gate Types used in Fault Tree Construction.

For the scope of this thesis only coherent fault trees will be utilised. Coherent fault trees comprise basic events combined with the logic gates AND, OR and combinations of these two, i.e. VOTE gates. Non-coherent fault trees contain working components and therefore include NOT logic, where a component is considered NOT failed.

### 2.4 Qualitative Analysis of a Fault Tree Structure

The fault tree representation of a system illustrates the relationship between component failures and an undesired system event. Qualitative analysis of a fault tree structure involves the identification of each unique combination, or single occurrence, of event failures which cause the top event.
The top event can be expressed in an algebraic form by using Boolean algebra, as each logical gate has a corresponding Boolean expression (Figure 2.2). Boolean algebra can therefore be used in order to obtain an expression for the causes of system failure in terms of combinations of individual component failures.

\[
\text{TOP} = A \cdot B
\]

\[
\text{TOP} = A + B
\]

\[
\text{TOP} = A \cdot B + A \cdot C + B \cdot C
\]

Figure 2.2 Table containing equivalent Boolean Expression for Logical Gates.

Figure 2.2 represents the equivalent Boolean expression for the AND, OR and VOTE gates. The dot or product symbol represents logical AND and the OR gate is represented algebraically by the ‘+’ symbol. For the VOTE gate in figure 2.2, TOP will occur if at least two out of the three inputs occur.
By using the Boolean expressions the fault tree failure logic can be translated into an algebraic form and the top event is described in terms of combinations of component failures. Each single or multiple component failure that causes the top event occurrence can be referred to as a cut set. A **Cut Set** can be defined as:

'A collection of basic events such that if they all occur the top event occurs'

Hence,

$$TOP = C_1 + C_2 + \ldots + C_n$$

where $C_i$ represents cut set $i$ and $n$ is the total number of cut sets in the system.

For example the top event of the fault tree shown in figure 2.3 is represented by the Boolean expression:

$$TOP = A \cdot B + A$$

For even relatively simple systems the number of cut sets can be large. In any system the largest cut set will comprise all component failures, however, the system will generally be caused by failure of fewer components. This implies that the cut set containing all component failures does not require all of the individual components in the cut set to fail to actually cause system failure, i.e. all components in the cut set are not both necessary and sufficient to cause system failure. The cut sets for the fault tree in figure 2.3 depicts this as failure of component A alone causes the top event,
illustrating that in the cut set AB, B is not necessary to cause the top event. This introduces the concept of a minimal cut set which can be defined as:

'the smallest combination of component failures which if they all occur cause the system to fail'

The Boolean expression of the top event is minimal only when all redundancy have been removed. A set of Boolean Algebra rules are used to aid in the reduction of the top event expression to gain its minimal form (Table 2.1).

<table>
<thead>
<tr>
<th>RULE NUMBER</th>
<th>RULES</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A + A = A</td>
</tr>
<tr>
<td>2</td>
<td>A + 1 = 1</td>
</tr>
<tr>
<td>3</td>
<td>A + 0 = A</td>
</tr>
<tr>
<td>4</td>
<td>A.A = A</td>
</tr>
<tr>
<td>5</td>
<td>A.1 = A</td>
</tr>
<tr>
<td>6</td>
<td>A.0 = 0</td>
</tr>
<tr>
<td>7</td>
<td>A + A.B = A</td>
</tr>
<tr>
<td>8</td>
<td>A + B = B+A</td>
</tr>
<tr>
<td>9</td>
<td>A.B = B.A</td>
</tr>
</tbody>
</table>

Table 2.1 Boolean Algebra Rules.

Rule number one is known as the first IDEMPOTENT rule and aids in the removal of repeated cut sets from the system failure expression.

Rule two and three include the binary constants 1 and 0 which represent event occurrence and non-occurrence respectively.

Rule four represents the second idempotent rule and is used to remove repeated events in a cut set expression.

The seventh rule is the absorption rule, its function is to remove any redundant cut sets. It states that if event A alone can cause top event failure then any combination of A with any other component results in the removal of the non-minimal cut set. By applying these rules to the Boolean expression for the top event, the system failure can be expressed in terms of the tree’s minimal cut sets. The reduced Boolean
expression is often referred to as the SUM OF PRODUCTS form or disjunctive normal form, where:

$$\text{TOP} = MC_1 + MC_2 + \ldots + MC_n$$

$MC_i = \text{Minimal cut set } i$

The ‘products’ represent the minimal cut sets where:

$$MC_j = c_1, c_2, \ldots, c_n$$

$c_j = \text{failure of component } j$

The ‘sum’ represents that the top event will occur if any of the minimal cut sets occur.

A minimal cut set containing $n$ component failures is referred to as a minimal cut set of order $n$. Generally the lower the order of the cut set the greater the influence that a cut set has on the system failure.

The minimal cut sets for a system are hence found by expressing the top event as a Boolean expression and then reducing that expression by using the Boolean algebra rules given in table 2.1.

2.4.1 Obtaining the Reduced Boolean Expression

There exists two main methods that are used to obtain the logic expression for the top event, these are the top-down approach and the bottom-up approach. The top-down approach is usually used in computer software as this approach reduces the Boolean expression at each level, therefore minimising the number of cut sets that need to be stored before the final reduction (ref. 6). The bottom-up approach involves tracing up through the tree from the bottom left-hand corner, expressing each gate with a Boolean expression containing only basic events. Once the top gate is reached the Boolean expression obtained from the bottom-up trace is reduced by applying the Boolean algebra rules (Table 2.1). The simple fault tree illustrated in figure 2.4 can be used to demonstrate the bottom-up approach in detail.

Bottom-Up Approach

Starting at the bottom of the tree, left hand side, the first gate encountered is GATE 3, which is an OR gate and is therefore equivalent to $B + C$.  

31
GATE 3 is itself an input to GATE 1, which is also an OR gate with GATE 3 and component A as its inputs. As GATE 1 is an OR gate a passage can be made through GATE 1 as long as one of the inputs to the gate occurs. GATE 1, therefore becomes B OR C OR A. By moving up another level the top level is encountered, which is an AND gate with inputs GATE 1 and GATE 2. Prior to obtaining an expression for the top gate an expression to represent GATE 2 needs to be determined. Following an identical procedure as that used for the development of GATE 1 an expression for GATE 2 is determined and the top event expression becomes:

$$\text{Top} = (B + C + A)(A.B + C)$$

Now that an expression has been obtained to represent the failure of the system it is expanded in order to obtain the Sum of Products form and to identify the minimal cut sets of the system. By expanding the brackets out the top event Boolean expression is obtained as:

$$\text{TOP} = B.C + B.A.B + C.C + C.A.B + A.C + A.A.B$$
By applying rule 4 to C.C, the cut set C is developed. Any cut set which contains events in combination with C can now be removed by applying the absorption rule, rule 7, which leads to:

\[ \text{TOP} = C + B.A.B + A.A.B \]

Redundancies are still present and by applying rule 4 and rule 7, the top event expression becomes:

\[ \text{TOP} = C + A.B \]

From the sum of products form the minimal cut sets can be extracted as \{C\} and \{AB\}.

This example highlights the effort required if the qualitative evaluation of the fault tree is performed manually. The need for a computerised procedure to determine the minimal cut sets of a real-life system, which may contain hundreds of components, is evident. Over the years various algorithms have been devised to determine the minimal cut sets of a fault tree structure. One of the most popular, which is utilised in many of the software packages, is the MOCUS algorithm developed by Fusell & Vesely (ref. 17).

2.5 Quantitative Analysis of a Fault Tree Structure

The quantitative analysis of a fault tree structure is based on a probabilistic method known as ‘Kinetic Tree Theory’ introduced by Vesely in 1970 (ref. 7). The underlying assumption of the Kinetic Tree Theory is that all basic events in the tree structure occur independently of one another.

The minimal cut sets of a system are used in the quantitative assessment which determine, amongst other measures, the overall system reliability, availability, number of expected failures and the rate of failure. Although the overall system characteristics are important, component or minimal cut set contributions can also aid in the discovery of weaknesses in the system and therefore assist in the redesign process.
In order to perform the quantitative assessment of a fault tree the likelihood of minimal cut set and component occurrence is required.

2.5.1 Component Failure Modes

For each individual component it is assumed that the component can only exist in one of two states at any one time. The component can either be functioning correctly or failed. In a short time interval, \( dt \), it is assumed that only one transition can occur. Therefore if a component is in a working state at time \( t \), then the component can either remain working or fail in the interval \( dt \). Similarly if the component is failed at time \( t \) it can either be repaired in \( dt \) or remain in a failed state. For any repairable component it is assumed that following repair the component is ‘as good as new’.

Each component’s life cycle is therefore made up of two different elements: the failure process and the repair process. Each is described separately in the following section.

2.5.1.1 The Failure Process

A failure distribution can be described as an attempt to represent mathematically the length of the life of a device (ref. 18). To describe a failure distribution several different functions can be used such as a cumulative failure distribution, a probability density function (pdf) and in certain cases the failure rate of the device. For the failure process the cumulative failure distribution is represented by \( F(t) \), the unreliability of a component, where unreliability is defined as:

\[
\text{The probability that the first failure occurs in the time interval } 0 \text{ to } t, \text{ given it worked at time zero}
\]

To obtain the pdf the following relationship holds:

\[
f(t) = \frac{dF(t)}{dt} \quad (2.1)
\]

and hence

\[
F(t) = \int_{0}^{t} f(u)du \quad (2.2)
\]
The expression $f(t)dt$ represents the probability that a failure occurs in the small interval $(t,t+dt)$.

From the cumulative failure distribution and the pdf of the failure process the failure rate, $r(t)$, of a component can be developed via the determination of $r(t)dt$ which represents the probability of failure in the interval $(t,t+dt)$ given that the component worked continuously from zero to $t$. In order to obtain an expression for the failure rate conditional probability theory is utilised, where

$$P(A|B) = \frac{P(A \cap B)}{P(B)} \quad (2.3)$$

Now if we let $A$ represent a failure in the interval $(t,t+dt)$ and $B$ represent the component working at time zero up to time $t$ then $r(t) \, dt = P(A|B)$. It can be seen that $B$ represents the reliability of a component where reliability, $R(t)$, is defined as:

*The probability that a component experiences no failures in the interval 0 to $t$, given it worked at time zero.*

Therefore:

$$r(t)dt = \frac{f(t)dt}{R(t)} \quad (2.4)$$

which equals:

$$r(t) = \frac{f(t)}{1-F(t)} \quad (2.5)$$

For many devices plotting $r(t)$ against time results in the curve referred to as the 'Bath-Tub' Curve (Figure 2.5).
The bath-tub curve is characterised by three individual phases, a decreasing failure rate phase, a constant failure rate phase and finally an increasing failure rate phase (ref. 19). Integrating equation (2.5) gives:

\[
F(t) = 1 - \exp\left(-\int_0^t r(u)\,du\right) \tag{2.6}
\]

2.5.1.2 The Repair Process

In order for a repair to be initiated a failure must have already occurred. The cumulative repair distribution for a component is represented by \(G(t)\), where \(G(t)\) represents the probability that a repair is completed before time \(t\) given the component was in a failed state at time zero.

Therefore the pdf for the repair process, \(g(t)\), is given by

\[
g(t) = \frac{dG(t)}{dt} \tag{2.7}
\]

where \(g(t)\,dt\) can be defined as the probability that a repair occurs at time \(t+dt\).

From the cumulative distribution function and the pdf of the repair process the repair rate, \(m(t)\), can be developed where \(m(t)\) is the probability of repair given that the component is in a failed state from time zero to time \(t\).

As with the failure rate derivation, an expression for the repair rate is obtained:
\[ m(t) = \frac{g(t)}{1 - G(t)} \]  

(2.8)

and following integration and rearrangement of equation (2.8) a formula to represent
the probability that the repair of a component is completed before time \( t \) is obtained
(equation 2.9).

\[ G(t) = 1 - \exp \left[ - \int_0^t m(u) \, du \right] \]  

(2.9)

2.5.1.3 The Life of a Component: The Whole Process

For repairable components the unconditional failure intensity, \( w(t) \), can be defined as:

*The probability that a component fails per unit time at time \( t \), given it worked at time zero*

and the unconditional repair intensity, \( v(t) \) can be defined as:

*The probability that a failed component is repaired per unit time at \( t \) given it worked at time zero*

In order to obtain an expression for the unconditional failure intensity of a component
the probability that the component is working at time \( t \) and then fails in the time
interval \( dt \), must be determined. This scenario can be achieved through two distinct
ways. Firstly the component works continuously from zero up to time \( t \) and then fails
in the small interval \( (t, t+dt) \). This can be evaluated using the pdf for the failure
process, \( f(t) \, dt \). The second independent way that can lead to component failure in the
small time interval \( (t, t+dt) \), given it worked at time zero, is that the component fails
prior to time \( t \) but is last repaired at time \( u \) prior to time \( t \). This situation is illustrated
in figure 2.6. The history of a components life prior to \( u \) is irrelevant. From figure 2.6
the last repair occurs in the time interval \( (u, u+du) \) and the component then works
continuously up to time \( t \). The probability that a component is repaired per unit time
at \( t \) given that it worked at time zero is represented by the unconditional repair
intensity \( v(t) \). The failure of a component can occur anywhere between \( u \) and \( t \), as the
last repair occurs in the small time interval \( u+du \). The probability of failure between
times \( u \) and \( t \) is represented by \( f(t-u) \, du \) and the probability of repair in the time
interval \( (u, u+du) \) is equal to \( v(u) \, du \). Since the last repair time \( u \) can occur at any point
between 0 and \( t \) the probability of the second scenario is given by:
The unconditional failure intensity is therefore determined from:

\[ w(t)dt = f(t)dt + \int_0^t v(u)du f(t-u)dt \]

therefore,

\[ w(t) = f(t) + \int_0^t v(u)f(t-u)du \]  \hspace{1cm} (2.10)

To calculate the unconditional failure intensity an expression for the unconditional repair intensity is required. In order for a repair to be initiated a failure must have already occurred. Therefore a component must have failed prior to time \( t \) and remain failed up to time \( t \). This situation is shown in figure 2.7. The failure in the time interval \((u, u+du)\), given that the component worked at time zero, is equivalent to the unconditional failure intensity of the component, \( w(u) \). The repair of the component can occur anywhere between \( u \) and \( t \), as the last failure occurs in the small time interval \( u+du \).
The probability of repair between times \( u \) and \( t \) is given by \( g(t-u) \) and since the last failure time \( u \) can occur at any point between 0 and \( t \) the unconditional repair intensity is given by:

\[
v(t) = \int_0^t g(t-u)w(u)du \tag{2.11}
\]

It can be seen by inspection of equations (2.10) and (2.11) that in order to obtain \( w(t) \) and \( v(t) \) these two equations need to be solved.

The unconditional failure intensity can be used to determine two important measures:

The conditional failure intensity, \( \lambda(t) \), of a component and the expected number of failures \( W(t_0, t_1) \) in the time interval \( t_0 \) to \( t_1 \). The conditional failure intensity can be defined as:

The probability that a component fails per unit time at \( t \) given it worked at time \( t \) and at time zero.

The difference between this and the unconditional failure intensity, \( w(t) \), is that \( \lambda \) is the failure rate based on those components which are working at time \( t \), whereas \( w \) is based on the whole population. \( \lambda(t)dt \) is the probability of failure between \( (t, t+dt) \) given the component worked at time \( t \) and at time zero. Using equation (2.3), where

Figure 2.7 Component fails prior to time \( t \) and is repaired in \( t + dt \)
event A represents failure in the interval (t, t+dt) and event B represents that the component worked at time t and at time zero, the conditional failure intensity satisfies:

$$\lambda(t) dt = \frac{P((\text{component fails between}(t, t+ dt) \cap \text{component works at time } t \text{ and time zero}))}{P(\text{component works at time } t \text{ and time zero})}$$

The above expression can be reduced by noting the fact that event A actually implies that the component must be working at time t, therefore

$$\lambda(t) dt = \frac{P(\text{component fails between}(t, t+ dt))}{P(\text{component works at time } t)}$$

The probability that a component works at time t represents the availability, $$A(t)$$, of a component. Therefore:

$$\lambda(t) dt = \frac{w(t) dt}{A(t)} \text{ or } \lambda(t) dt = \frac{w(t) dt}{1-Q(t)}$$, where $$Q(t)$$ represents component unavailability

Hence,

$$\lambda(t) = \frac{w(t)}{1-Q(t)} \quad (2.12)$$

The expected number of failures is defined by:

$$W(t, t+dt) = \text{Expected number of failures in the time period } (t, t+dt) \text{ given that the component worked at time zero}$$

Using expectation (ref. 20)

$$W(t, t+dt) = \sum_{i=1}^{\infty} i \cdot P(i \text{ fails during } t, t+dt | \text{worked at } 0)$$

as only one failure can occur in a small time interval the above expression is reduced to:

$$W(t, t+dt) = P(\text{one failure in } t, t+dt | \text{worked at } 0)$$
therefore
\[ W(t, t+dt) = w(t)dt \] (2.13)

The expected number of failures over a certain time span is determined from:
\[ W(t_1, t_2) = \int_{t_1}^{t_2} w(t)dt \] (2.14)

As with the failure process, the unconditional repair intensity can be used to calculate
the conditional repair rate, \( \mu(t) \), and the expected number of repairs, \( V(t_1, t_2) \).

The conditional repair rate, \( \mu(t) \), can be defined as the probability that a component is
repaired per unit time at time \( t \) given it worked at zero and failed at time \( t \) and
expressed mathematically as:
\[
\mu(t) = \frac{\text{P(component repaired between}(t, t + dt)\text{)}}{\text{P(component failed at time } t)}
\]

\[
\mu(t) = \frac{v(t)}{Q(t)} \text{ or } \mu(t) = \frac{v(t)}{1 - A(t)} \] (2.15)

The expected number of repairs in the interval \( (t, t+dt) \), \( V(t, t+dt) \), can be shown to be
equal to:
\[ V(t_1, t_2) = \int_{t_1}^{t_2} v(u)du \] (2.16)

2.5.2 Calculating The Unavailability of a Component

An expression can be obtained for a components unavailability, \( Q(t) \), in terms of the
expected number of repairs and expected number of failures, \( W(0,t) \) and \( V(0,t) \)
respectively.
\[ Q(t) = W(0,t) - V(0,t) \] (2.17)

hence
\[ Q(t) = \int_0^t [w(u) - v(u)]du \] (2.18)

The distribution most commonly used in FTA to represent times to failure and repair
is the Exponential distribution. A feature of this distribution is that it is characterised
by a single parameter, $\lambda$. The distribution assumes that equipment does not age and for this reason the failure rate is deemed constant. This distribution, therefore, represents the constant, useful life period of a component in the bath tub curve (Figure 2.5). The relationship $\lambda(t) = \lambda$ now holds and similarly $\mu(t) = \mu$. With these failure and repair rates in place the unavailability can be determined. Implementing the exponential distribution gives:

$$F(t) = 1 - e^{-\lambda t}, \quad f(t) = \lambda e^{-\lambda t}, \quad G(t) = 1 - e^{-\mu t}, \quad g(t) = \mu e^{-\mu t}$$

Using $f(t)$ and $g(t)$ two coupled equations are obtained from equation (2.10) and (2.11).

$$w(t) = \lambda e^{-\lambda t} + \int_0^t \lambda e^{-\lambda(t-u)} v(u) du$$
$$v(t) = \int_0^t \mu e^{-\mu u} w(u) du$$

These equations can be de-coupled and solved by using Laplace transforms (ref. 6), which yields the following results:

$$w(t) = \frac{\lambda \mu}{\lambda + \mu} + \frac{\lambda^2}{\lambda + \mu} e^{-(\lambda+\mu)t}$$
$$v(t) = \frac{\lambda \mu}{\lambda + \mu} - \frac{\lambda \mu}{\lambda + \mu} e^{-(\lambda+\mu)t}$$

From equations (2.14) and (2.16) the expected number of failures and repairs can be determined respectively.

$$W(0,t) = \frac{\lambda \mu}{\lambda + \mu} + \frac{\lambda^2}{\lambda + \mu} (1 - e^{-(\lambda+\mu)t})$$
$$V(0,t) = \frac{\lambda \mu}{\lambda + \mu} - \frac{\lambda \mu}{\lambda + \mu} (1 - e^{-(\lambda+\mu)t})$$

Finally equation (2.17) gives the unavailability as:

$$Q(t) = \frac{\lambda}{\mu + \lambda} (1 - e^{-(\lambda+\mu)t}) \quad (2.19)$$

This expression represents the situation where the failure of the component is revealed, i.e. immediately apparent. Equation (2.19) reduces to $1 - e^{-\lambda t}$ when the component is non-repairable and to $\frac{\lambda}{\lambda + \mu}$ as $t \to \infty$. 

42
When the failure of a component is unrevealed, meaning that the failure will not be detected until an inspection of the component has been initiated, the calculation for the component unavailability is different. The unavailability is dependent on the time between inspections and is illustrated in figure 2.8. In this case the average unavailability is given by:

\[ Q_{AV} = 1 - \frac{(1 - e^{-\lambda \theta})}{\lambda \theta} \]  

(2.20)

where \( \theta \) is the inspection interval.

![Figure 2.8 Unavailability for an Unrevealed Failure](image)

2.5.3 Calculating Minimal Cut Set Unavailability and Unconditional Failure Intensity

As with individual component failures the unavailability of a minimal cut set can be determined. For minimal cut sets greater than order one the probability of occurrence is calculated by using the Multiplication Law (Equation 2.21) for \( n \) independent events, \( A_i \).

\[ P(A_1 \cdot A_2 \cdots A_n) = \prod_{i=1}^{n} P(A_i) \]  

(2.21)

The unconditional failure intensity of minimal cut set \( i \), the expected number of times the minimal cut set occurs per unit time at time \( t \), can also be obtained. As only one failure can occur in the small time interval \( t \) to \( t + dt \), all but one component in the minimal cut set must exist failed at time \( t \) and the remaining single component must
occur in the time $t$ to $t + dt$ to cause the minimal cut set to occur at this time. This represents the unconditional failure intensity of a minimal cut set, $w_{MCi}$, and can be expressed as:

$$w_{MCi}(t)dt = \sum_{i=1}^{n} \{w_i(t)dt\prod_{j \neq i} q_j(t)\}$$

dividing through by $dt$ yields:

$$w_{MCi}(t) = \sum_{i=1}^{n} \{w_i(t)\prod_{j \neq i} q_j(t)\} \quad (2.22)$$

### 2.5.4 Calculating System Unavailability

Once the minimal cut sets have been found and the failure and repair process for each component obtained, the top event probability can be calculated. There exists several different methods for obtaining an expression for the top event probability of a system (ref. 5), the three most commonly used are the **Exact Method**, the **Minimal Cut Set Upper Bound** and the **Rare Event Approximation Method**.

#### 2.5.4.1 The Exact Method

The exact solution method is based on the **Inclusion-Exclusion Principle**, equation (2.23), which is generated from basic probability theory. The inclusion-exclusion principle relies on the assumption that all components fail independently.

$$P(T) = \sum_{i=1}^{N} P(MC_i) - \sum_{i=2}^{N} \sum_{j=1}^{N} P(MC_i \cap MC_j) + \cdots + (-1)^{N-1} P(MC_1 \cap MC_2 \cap \cdots MC_N) \quad (2.23)$$

The expanded inclusion-exclusion formula can also be achieved by first producing the structure function for the top event, $\phi(x)$, and taking its expectation. The structure function for the top event with minimal cut sets $i$ to $n$ is:

$$\phi(x) = 1 - \prod_{i=1}^{n} (1 - \rho_i) \quad (2.24)$$

where $\rho_i$ is the binary indicator variable for minimal cut set $i$ such that:
\[ \rho_i = \begin{cases} 1 & \text{if minimal cut set } i \text{ is failed} \\ 0 & \text{if minimal cut set } i \text{ has not occurred} \end{cases} \]

In the case that each minimal cut set is statistically independent, i.e. no minimal cut sets share any component, the top event probability equals the expectation of the structure function, i.e.

\[ P(\text{TOP}) = E[\phi(x)] = \Phi[E[x]] \]

However this case is rare as minimal cut sets often share common component failures. Full expansion of equation (2.24) followed by reduction using Boolean algebra rules prior to taking the expectation deals with the common failure problem. An alternative, more efficient way of dealing with repeated events is to use Shannon’s Decomposition formula (ref. 21):

\[ \phi(x) = X_i \phi(1_i, x) + X_i \phi(0_i, x) \quad (2.25) \]

where:

\[ \phi(1_i, x) = \phi(X_1, \ldots, X_{i-1}, 1, X_{i+1}, \ldots, X_n) \quad \text{represents } X_i \text{ has failed} \]

\[ \phi(0_i, x) = \phi(X_1, \ldots, X_{i-1}, 0, X_{i+1}, \ldots, X_n) \quad \text{represents } X_i \text{ works} \]

Taking expectation then gives

\[ \phi_{\text{SYS}} = Q(X_i)Q(1_i, x) + (1-Q(X_i))Q(0_i, x) \]

where:

\[ Q(1_i, x) = Q(X_1, \ldots, X_{i-1}, 1, X_{i+1}, \ldots, X_n) \quad \text{represents component } i \text{ has failed} \]

\[ Q(0_i, x) = Q(X_1, \ldots, X_{i-1}, 0, X_{i+1}, \ldots, X_n) \quad \text{represents component } i \text{ works} \]

### 2.5.4.2 Approximation Techniques

In large fault trees where there may be several thousand minimal cut sets the use of the inclusion-exclusion expansion is too CPU intensive and for this reason approximations have been developed. The inclusion-exclusion expansion adds odd numbered terms and subtracts even numbered terms, where each term will be numerically less significant than its preceding term. Therefore truncating the expansion after an odd numbered term will produce an upper bound and truncating after an even numbered term will produce a lower bound.
The Rare Approximation Method

A commonly used upper bound approximation is the **Rare Event Approximation** (ref. 5). The rare event approximation method yields an upper bound of the exact answer by using only the first term in the inclusion-exclusion expansion. Equation (2.26) illustrates how the probability of the top event is obtained using the Rare Event approximation technique.

\[ P(T) = \sum_{i=1}^{N} P(MC_i) \]  

(2.26)

**Minimal Cut Set Upper bound Method**

A more accurate upper bound is given by finding the minimal cut set upper bound, equation (2.27). This method, as its title suggests, is based on the systems minimal cut sets. The method uses the logic that the system will fail if at any time at least one of the minimal cut sets has occurred. By using basic probability theory this situation can be expressed by:

\[ P(T) = 1 - P(\text{No minimal cut sets occur}) \]

Thus, if all minimal cut sets, \( MC_i \), are independent then the above expression becomes:

\[ P(T) \leq 1 - \prod_{i=1}^{N} P(\text{minimal cut set } i \text{ does not occur}) \]

\[ P(T) \leq 1 - \prod_{i=1}^{N} (1 - P(MC_i)) \]  

(2.27)

This method produces an exact answer if no components are repeated in any cut set \( i \). However if a component is present in more than one minimal cut set then equation (2.27) produces an upper bound approximation as the repeated component is considered more than once.

The relationship between the three expressions for \( Q_{SYS} \) is shown in inequality (2.28).

\[ \text{EXACT} \leq \text{MINIMAL CUT SET UPPERBOUND} \leq \text{RARE APPROXIMATION METHOD} \]  

(2.28)
The process of finding the exact solution can be extremely quick for a small fault tree, however, for trees that yield a large number of minimal cut sets the process can be computationally expensive. In most fault tree software packages the minimal cut set upper bound is assumed to produce an accurate result and is implemented instead of the exact method.

2.5.5 Calculating System Unconditional Failure Intensity

The unconditional failure intensity for a system, $w_{SYS}(t)$, can be obtained using Birnbaum’s Criticality function (ref. 6). The criticality function for component $i$ is defined as the probability that the system is in a critical state for component $i$ and is determined by:

$$G_i(q) = Q(1_i, q) - Q(0_i, q)$$

where $Q(q)$ is the probability that the system fails and,

$Q(1_i, q) = Q(q_1, ..., q_{i-1}, 1_i, q_{i+1}, ..., q_n)$ represents component $i$ has failed

$Q(0_i, q) = Q(q_1, ..., q_{i-1}, 0_i, q_{i+1}, ..., q_n)$ represents component $i$ has worked

In order to obtain the unconditional failure intensity of the system, due to component $i$, the criticality function is multiplied by the unconditional failure intensity for component $i$. This process is continued for all $n$ components in a system and $w_{SYS}(t)$ is given by:

$$w_{SYS}(t) = \sum_{i=1}^{n} G_i(q(t)) \cdot w_i(t)$$  \hspace{1cm} (2.29)

The system unconditional failure intensity can be used to determine the expected number of system failures in a designated time period. The expected number of system failures in the time period $t_0$ to $t_1$ is given by:

$$W(t_0, t_1) = \int_{t_0}^{t_1} w_{SYS}(t) dt$$  \hspace{1cm} (2.30)

2.6 Initiating and Enabling Events

The previously outlined procedure for quantification of a fault tree structure assumed that the order in which the events in any minimal cut set occurred was irrelevant. In safety systems, the order in which certain failures occur will determine whether the
top event happens or not. In general, safety protection systems comprise several shutdown options given a hazardous scenario. However if the shutdown components have failed prior to a demand then any failure of the system itself will result in a catastrophic situation. On the otherhand, if any component not part of the shutdown process fails prior to the failure of the shutdown system then a full shutdown will be initiated and the hazardous situation avoided. This type of situation is modelled by considering failures as either initiating or enabling events (ref. 6).

Initiating events can be considered as events that place a demand on the safety systems to shutdown the system and are formally defined as:

*Events that perturb system variables and place a demand on control/protection systems to respond*

Enabling events represent the failure of devices placed in a system to protect against hazardous events and are defined as:

*Events that are inactive control/protective systems which permit initiating events to cause the top event*

### 2.6.1 Quantification using Initiator and Enabler Theory

The expected number of failures, $W(0, t)$, can be used as an upper bound for the unavailability of a system. For a system containing initiator and enabling events $w_{\text{SYS}}(t)$ can be obtained via a modification to equation (2.29), where $i$ corresponds only to initiating events.

It is essential that in order to obtain an accurate measure of the chance of system failure that initiating and enabling events are identified and the appropriate quantification performed.

### 2.6.2 Example System Containing Initiator and Enabler Events

Since the focus of this thesis is to develop ways of modelling systems which feature types of dependency, the initiator/enabler theory was applied to an example to
highlight its ability to deal with sequential failures. The system chosen contains both initiating and enabling events for a simple pressure tank system. The function of the system is to control the operation of a pump which pumps fluid from a reservoir into a tank. The system configuration is illustrated in figure 2.9 and the component’s individual functions and rate of failure per hour are represented in table 2.2.

Initially the system is in a dormant de-energised state with the switch closed, pressure relief valve closed, operator and pressure gauge available and the timer relay contacts open. The system is started by the operator re-setting the timer. The timer relay opens when the required amount of time has passed which will insure that the tank is fully pressurised. For this example it is assumed that once the timer registers 10 minutes of continuous power the timer contacts open breaking the circuit. In the occurrence of the timer failing to time out, a pressure gauge is attached to the tank which is monitored by the operator. If the pressure gauge reaches a pre-set value the operator manually opens the switch which removes power to the pump. In addition to this the pressure relief valve, PRV, opens prior to the tank becoming overpressurised. The system functions twice daily, i.e. once every 12 hours.
<table>
<thead>
<tr>
<th>Component</th>
<th>Function</th>
<th>Failure Modes</th>
<th>Failure Rate, $\lambda$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch (SW)</td>
<td>A safety feature to remove power to the pump when manually opened.</td>
<td>SWC: Switch failed closed when operator attempt to open</td>
<td>$1 \times 10^{-5}$</td>
</tr>
<tr>
<td>Pressure Relief Valve (PRV)</td>
<td>Releases pressure from tank prior to overpressure</td>
<td>PRV: Fails closed</td>
<td>$1 \times 10^{-4}$</td>
</tr>
<tr>
<td>Operator (OP)</td>
<td>Inspects Pressure Gauge and opens switch if pressure exceeds pre-set value</td>
<td>OP: Operator fails to respond</td>
<td>$1 \times 10^{-6}$</td>
</tr>
<tr>
<td>Timer Relay (TIM)</td>
<td>Provides reset for each operation. Timer opens when tank is full breaking circuit to the motor</td>
<td>TCC: Timer contact fails closed</td>
<td>$1 \times 10^{-4}$</td>
</tr>
<tr>
<td>Pressure Gauge</td>
<td>Monitors the pressure of the tank</td>
<td>PG: Gauge Failed low</td>
<td>$1.5 \times 10^{-6}$</td>
</tr>
<tr>
<td>Motor</td>
<td>Pumps fluid into tank</td>
<td>Assumed to be perfectly reliable in this study</td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2 Component Functions and corresponding Failure Rates

The fault tree structure for this system is illustrated in figure 2.10 and for this system the only initiator event can be identified as the Timer, TIM. Failure of the Timer places a demand on the remainder of the components to protect the tank. The remaining components are all enablers. From the fault tree structure it can be seen that there exists three minimal cut sets for the pressure tank system, all of which are of order three.

$$MC_1 = \{TCC, PRV, OP\}$$

$$MC_2 = \{TCC, PRV, SWC\}$$

$$MC_3 = \{TCC, PRV, PG\}$$
2.6.2.1 Quantification for Pressure Tank Example

The probability of failure for each individual component is calculated firstly on the basis of whether the component is an initiator or an enabler. If the component is an initiator then a frequency of failure is required not a probability of failure, and this is equal to the failure rate, i.e. \( w_{\text{TIM}} = \lambda_{\text{TIM}} \). For an enabler the probability of failure is dependent on the type of failure, i.e. unrevealed or revealed failure. All the enablers in the pressure tank system are unrevealed failure, as the system will function correctly whether these have failed to function or not until the initiating event fails, placing a demand on them to function. The probability of failure for each of the enablers is obtained using equation (2.20) with the inspection interval, \( \theta \), equal to one year, i.e. 8760 hours.
The unconditional failure intensity for each individual component is obtained via equation (2.12) and tabulated with the probability of failure in table 2.3.

<table>
<thead>
<tr>
<th>Component</th>
<th>Probability of failure, Q</th>
<th>Unconditional Failure Intensity, w</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch (SW)</td>
<td>0.04255</td>
<td>$9.5745 \times 10^{-6}$</td>
</tr>
<tr>
<td>Pressure Relief Valve (PRV)</td>
<td>0.3338</td>
<td>$6.66 \times 10^{-5}$</td>
</tr>
<tr>
<td>Operator (OP)</td>
<td>4.3672 x 10^{-3}</td>
<td>$9.956328 \times 10^{-7}$</td>
</tr>
<tr>
<td>Timer Relay (TIM)</td>
<td>---------</td>
<td>$1 \times 10^{-4}$</td>
</tr>
<tr>
<td>Pressure Gauge (PG)</td>
<td>6.54132 x 10^{-3}</td>
<td>$1.49 \times 10^{-6}$</td>
</tr>
</tbody>
</table>

Table 2.3 Unconditional Failure Intensity for components in Pressure Tank Example

To obtain an expression for the unconditional failure intensity of the system equation (2.29) is used, where i is equal only to initiating events, and hence an approximation to the system's unreliability is found. For this system,

$$Q(q) = q_{TIM} q_{PRV} q_{OP} + q_{TIM} q_{PRV} q_{SW} + q_{TIM} q_{PRV} q_{PG} - q_{TIM} q_{PRV} q_{OP} q_{SW}$$

$$- q_{TIM} q_{PRV} q_{OP} q_{PG} - q_{TIM} q_{PRV} q_{PG} q_{SW} + q_{TIM} q_{PRV} q_{OP} q_{SW} q_{PG}$$

The only initiating event is the Timer, therefore,

$$w_{SYS}(t) = G_{TIM}(q(t)) w_{TIM}(t)$$

The criticality function for the timer is given by:

$$G_{TIM}(q) = Q(1_{TIM}, q) - Q(0_{TIM}, q)$$

where

$$Q(1_{TIM}, q) = q_{PRV} q_{OP} + q_{PRV} q_{SW} + q_{PRV} q_{PG} - q_{PRV} q_{OP} q_{SW} - q_{PRV} q_{OP} q_{PG}$$

$$- q_{PRV} q_{PG} q_{SW} + q_{PRV} q_{OP} q_{SW} q_{PG}$$

$$Q(0_{TIM}, q) = 0$$

Therefore the unconditional failure intensity for the pressure tank system is,

$$w_{SYS} = [q_{PRV} q_{OP} + q_{PRV} q_{SW} + q_{PRV} q_{PG} - q_{PRV} q_{OP} q_{SW} - q_{PRV} q_{OP} q_{PG}$$

$$- q_{PRV} q_{PG} q_{SW} + q_{PRV} q_{OP} q_{SW} q_{PG}], w_{TIM}$$

$$= 0.125 \times 10^{-12}$$

The expected number of failures in a year can be calculated using equation (2.30)
If the unconditional failure intensity is calculated for the pressure tank system, without
the inclusion of the initiator/enabler distinction, then:
\[ W_{SYS}(t) = 0.259 \times 10^{-6} \]

The error factor that results from the exclusion of initiator and enabler events was
found to be 236 in a year. Therefore for an accurate analysis it is essential that
initiators and enablers are identified and the appropriate quantification procedure
adopted.

2.7 Summary and Conclusion

FTA is the most commonly used method for system performance analysis. The
technique represents the failure logic of the system in a diagrammatic tree structure
which is not only used as a basis for the analysis but is also essential as a
communication tool. The method can produce a qualitative or quantitative system
analysis, both of which can be used to highlight weaknesses in a system design.

The fundamental disadvantage with the fault tree method is that it cannot accurately
analyse systems containing dependent failures. The two main dependencies
investigated in this thesis are that of sequential and standby failures. It has been
shown that the fault tree method can account for sequential failures, to a limited
degree, by employing initiator/enabler theory. The example included illustrates the
inaccuracy that can result if dependent failure events are modelled as independent
events. The fault tree method has no capability of modelling warm or cold standby
failures, where the failure of the standby component is dependent on the failure of the
on-line component. FTA can, however, model hot standby events as the standby
component has a failure rate which is equal to the on-line component, hence no
dependency between the two exists.
3.1 Introduction

For systems containing independent failure events it has been shown that the FTA technique produces a logical description of the failure process and can also yield, among other things, the systems unreliability. It has been highlighted, however, that this technique has limitations even when applied to systems containing independent failure events. Qualitatively, if the fault tree is complex then finding the minimal cut sets can be CPU intensive. In addition to this the exact top event probability, found via the inclusion-exclusion formula, may also be computationally expensive if the system contains even a moderate number of minimal cut sets. In the past this problem has been solved by using approximations for the top event probability. These approximations, however, can be inaccurate if the likelihood of component failure is not small. Both problems can be overcome using the development of the Binary Decision Diagram (BDD) approach. Although the theory behind the BDD’s has existed since the late 1930’s, the first application to reliability and more specifically FTA, was in the mid 1980’s by Schneeweiss (ref. 22).

This chapter aims to describe the theory behind the BDD approach and highlight its role in a reliability environment.

3.2 The Development of the BDD Approach

In 1938 Shannon’s paper on switching circuits showed how electrical circuits could be represented using symbolic logic and simplified using Boolean algebra (ref. 21). Each circuit was represented as a set of equations which corresponded to the various relays switches in the circuit. The switching function, which symbolised the circuit configuration, was then manipulated using Boolean algebra to the form which represented the simplest expression for the components in the circuit. In calculus it can be shown that any continuous function, with continuous derivatives, may be expanded in a Taylor’s series. Shannon applied this to the equation of a switching
function and produced a formula referred to as ‘Shannon’s Decomposition Formula’ shown in equation (3.1).

\[ f(X_1, X_2, \ldots, X_n) = X_1 \cdot f(1, X_2, \ldots, X_n) + \overline{X_1} \cdot f(0, X_2, \ldots, X_n) \quad (3.1) \]

In equation 3.1 the function is said to have been expanded or pivoted about \( X_1 \) and the formula can be used to obtain the exact top event probability of a fault tree structure, as suggested in chapter 2.

In 1959 C. Lee concluded that a binary-decision programming representation of the circuit led to a superior computation of the switching function than that of the usual Boolean representation (ref. 23). A binary decision program is based on one line of instructions for each variable. An example of such an instruction is:

\[ T \ x: A, B \]

which implies that if the variable \( x \), at address \( T \), is ‘0’ then the instruction at address \( A \) is followed, else if the variable \( x \) is ‘1’ then the instruction at address \( B \) is followed. Lee’s paper showed how a switching circuit could be described exactly by a binary decision program.

In 1978, forty years after Shannon’s work with switching functions, S. Aker defined a digital function in terms of a ‘diagram’ which told the user the output value of the function by examining the values of the inputs (ref. 24). The diagram, referred to as the Binary Decision Diagram (BDD), is useful for finding the binary value of a function given the binary values of the function’s inputs. The BDD is hence developed by examining the effect of the input variables on the output function. The BDD for the function \( f = A + B \cdot C \) is depicted in figure 3.1.

Inspection of figure 3.1 shows that the input variables are represented by circles with two outlet branches. These circles are referred to as non-terminal nodes and the top non-terminal node is known as the ‘ROOT NODE’. The ‘1’ outlet branch from a non-terminal node symbolises that the variable equals 1 and similarly the ‘0’ outlet branch symbolises that the variable equals 0. The output or function value is represented as a terminal box, known as terminal nodes, which either contain the value ‘1’ or ‘0’.
It should be noted here that the values ‘1’ and ‘0’ can imply different scenarios, i.e. in switching theory they represent that a circuit is opened or closed and in reliability theory they represent component failure and functionality respectively. The BDD can be seen to be identical to the Binary decision program in that each variable has 2 options depending on whether it equals ‘0’ or ‘1’.

Aker’s paper highlighted that the BDD could be derived using different formulations, such as using a truth table which holds all possible variable combinations of 1’s and 0’s. Derivation from a truth table results in the development of a binary decision tree with $2^n - 1$ nodes, however Aker showed that a more efficient approach could be used in the development of the BDD. Instead of developing a truth table the function’s Boolean expression is involved in a ‘top-down’ procedure which includes repeated application of Shannon’s decomposition formula.

A further observation made by Aker was that the BDD is superior to the Binary decision tree, as any node in the BDD may have more than one input branch. This feature implies that the size of the BDD can be reduced. For example the BDD in figure 3.2a can be reduced by allowing the ‘1’ outlet branch of the non-terminal node
representing component B to enter the same non-terminal node as the ‘0’ outlet branch, as each branch enters the same non-terminal node representing component D. This reduction is illustrated in figure 3.2b. Figure 3.2b can be reduced further by the realisation that component B is superfluous or redundant. Regardless of whether B is equal to ‘0’ or ‘1’ component D is considered next. Removal of the non-terminal node containing B reduces the BDD to the form shown in figure 3.2c. It can therefore be defined that:

A node is superfluous and can be replaced by its output if it possess identical non-terminal or terminal nodes on both outlet branches.

Figure 3.2 Reduction of a BDD
Being able to reduce the size of the BDD implies that the analysis will be more efficient than the binary decision tree. A further reduction in size, and hence in the amount of computer memory required to store a BDD, can be made if common modules exist. Therefore if two nodes are equivalent then one can be deleted and all of the incoming branches to the deleted node directed to the remaining common node. For example figure 3.2c contains a common node representing component D. Both the ‘1’ outlet branch from the root node and the ‘0’ outlet branch of component C stem to the non-terminal node containing D. Removing the node containing D which is attached to the node representing C results in the ‘0’ outlet branch from the non-terminal node containing C to be directed to the node attached to the ‘1’ outlet branch of component A. This reduces the BDD by one non-terminal node and the new form is shown in figure 3.2d.

Aker highlighted that one of the main advantages of this technique was the efficiency in which the BDD could be stored in a computer. For each variable only two pieces of information are required: where to go if the variable equals ‘1’ or ‘0’. As with the binary decision program, the BDD can be shown to lead to a superior computation of a function than that of a Boolean representation.

3.3 Application of the BDD approach to a Reliability Environment

The BDD approach was first applied to a reliability environment in the mid 1980’s where Schneeweiss transformed a fault tree structure into a binary decision tree. From this tree it was shown that the BDD was a more concise representation, than that of the binary decision tree. Since Schneeweiss’s publication many investigations have been carried out using the BDD method. One of the most influential was by Randal E. Bryant who represented a function as a BDD but restricted the ordering of the decision variables (ref. 25). The ordering is defined as: A<B, where A and B are both represented by non-terminal nodes and the effect of A is considered before the effect of B. The paper also contains a set of manipulation algorithms for the BDDs. As with Aker, Bryant used Shannon’s decomposition to represent pivoting about a variable present in a function. The work completed by Bryant, Aker and Lee led Rauzy to develop an algorithm which achieved increased efficiency for FTA (ref. 26). The
method is based on the BDD approach and involves converting the fault tree structure to a BDD in order to complete an efficient qualitative and quantitative analysis.

3.3.1 Conversion of a Fault tree Structure to a BDD Using the Structure Function

The BDD can be constructed from the structure function, \( \phi(x) \), of a system. Each basic event is considered and the effect of component i failing (binary ‘1’) and working (binary ‘0’) is identified. The fault tree structure shown in figure 3.3 can be used to illustrate the construction process.

![Figure 3.3 Simple Fault Tree Example](image)

The minimal cut sets for the system can be shown to be:

\[
MC_1 = \{AB\}, MC_2 = \{CD\}
\]

and using equation (2.24) the structure function is equal to:

\[
\phi(x) = 1 - (1 - X_A X_B)(1 - X_C X_D)
\]

Before the construction can begin the basic events need to be placed in an ordering scheme. For this example a top-down, left-right ordering scheme would yield:

\[A<B<C<D\]

The root node for the BDD, therefore, represents basic event A. A is considered first and the effect on the structure function is determined. Following the failure of component A, i.e. \( X_A = 1 \), the structure function becomes:

\[
\phi(x) = 1 - (1 - X_B)(1 - X_C X_D)
\]

and when A functions correctly, i.e. \( X_A = 0 \), the structure function is:
\[ \phi(x) = 1 - (1 - X_C \cdot X_D) \]

The BDD is developed further by considering the effect of the next relevant basic event in the ordering scheme. The BDD for the fault tree in figure 3.3 is depicted in figure 3.4. By inspection of figure 3.4 it can be seen that there exists a common node starting from:

\[ \phi(x) = 1 - (1 - X_C \cdot X_D) \]

The final BDD is obtained by removing all redundant and repeated features. The '0' outlet branch from the root node, then, is re-directed to the non-terminal node attached to the '0' outlet branch of component B. The reduced BDD is given in figure 3.5.
3.3.2 Conversion of a Fault Tree Structure using an If-Then Else Approach

The approach outlined above is inefficient for large fault trees. The most frequently employed approach utilises the If-Then-Else structure (ite), which is derived from Shannon’s formula. Each non-terminal node in the BDD has an ite structure of the form ite(x,f1,f2), implying that if x fails then consider function f1 else consider function f2. This structure is represented as a BDD in figure 3.6.
Constructing the BDD from a fault tree includes obtaining an ordering of the basic events, which will be taken from the top of the tree downwards, and manipulating the ite structures for each basic event using the following rules:

**RULE 1:**

when \( x < y \)

If \( J = \text{ite}(x, f_1, f_2) \) and \( H = \text{ite}(y, g_1, g_2) \)

then \( J \hat{\circ} H = \text{ite}(x, f_1 \hat{\circ} H, f_2 \hat{\circ} H) \)

where \( \hat{\circ} \) is a binary connective which symbolizes either `+` or `·`.

**RULE 2:**

when \( x = y \)

If \( J = \text{ite}(x, f_1, f_2) \) and \( H = \text{ite}(x, g_1, g_2) \)

then \( J \hat{\circ} H = \text{ite}(x, f_1 \hat{\circ} g_1, f_2 \hat{\circ} g_2) \)

In addition to these rules the following properties are used:

\[
1 + H = 1 \quad 0 + H = H \quad 1.H = H \quad 0.H = 0
\]

The example fault tree given in figure 3.7 can be used to demonstrate the construction of a BDD from a fault tree using ite structures.

Initially an ordering is required. For figure 3.7, reading from the top-down in a left-to-right fashion, the ordering is \( B<C<A \). The ite structures for each basic event are:
\[ A = \text{ite}(A, 1, 0) \]
\[ B = \text{ite}(B, 1, 0) \]
\[ C = \text{ite}(C, 1, 0) \]

Working from the bottom-up, Rule 1 gives:

\[ \text{GATE 1} = \text{ite}(B, 1, 0) + \text{ite}(C, 1, 0) \]
\[ = \text{ite}(B, 1 + \text{ite}(C, 1, 0), 0 + \text{ite}(C, 1, 0)) \]
\[ = \text{ite}(B, 1, \text{ite}(C, 1, 0)) \]

The ite structure for the TOP event can now be found, using rule 1, to be:

\[ \text{TOP} = \text{ite}(B, 1, \text{ite}(C, 1, 0)) \cdot \text{ite}(A, 1, 0) \]
\[ = \text{ite}(B, 1 \cdot \text{ite}(A, 1, 0), \text{ite}(C, 1, 0) \cdot \text{ite}(A, 1, 0)) \]
\[ = \text{ite}(B, \text{ite}(A, 1, 0), \text{ite}(C, \text{ite}(A, 1, 0), 0)) \]

To construct the BDD the ite structure for the TOP event expression is iteratively investigated to determine the expression represented on the '1' and '0' outlet branches. The resulting BDD for figure 3.7 is shown in figure 3.8.

![BDD representation of figure 3.7](image)

**3.4 Qualitative Analysis of a BDD**

Rauzy defined that each path from the root node of a BDD to a terminal node containing the value '1' represented a solution of \( f \). Only the components on this path...
that exit their nodes on the ‘1’ outlet branch are included in the cut set. For the BDD illustrated in figure 3.8, then, the cut sets are:

\[
\begin{align*}
C_1 &= \{BA\} \\
C_2 &= \{CA\}
\end{align*}
\]

These are also the minimal cut sets for the system. However the BDD does not always produce a list of minimal cut sets. To obtain the minimal cut sets either the resulting list of cut sets can be reduced using Boolean algebra rules or the BDD can undergo a minimisation process. For the scope of this thesis the minimisation process of the BDD is not required but is explained in detail in (ref 26).

### 3.5 Quantitative Analysis of a BDD

The probability of the top event occurrence, \( Q_{\text{TOP}} \), can be expressed by the BDD as the sum of probabilities of the disjoint paths through the BDD. The disjoint paths through the BDD are the paths that lead from the root node to any terminal node containing the value ‘1’. For quantitative purposes the components exited on the ‘0’ outlet branch are included. For component \( i \) that lies on a ‘0’ outlet branch, the probability of travelling down such a branch is given by \( \overline{Q}_i \), which is equal to \( (1-Q_i) \).

The disjoint paths for figure 3.8 are:

1) BA
2) BCA

The top event probability for figure 3.8 is therefore calculated by:

\[
\text{TOP} = Q_B Q_A + \overline{Q}_B Q_C Q_A
\]

The BDD can also be used to determine other measures of a system’s performance, such as the frequency of failure of the system and the contribution to failure for individual components in the system (ref. 27). Initiator/enabler theory can also be included in the calculation procedures (ref. 28).

### 3.6 Summary and Conclusion

The use of BDDs to improve the efficiency of identifying the minimal cut sets of a fault tree and calculating the exact top event probability has been proven for large
complex systems (ref. 26). Because of this, the method can be seen as superior to the conventional methods such as FTA. One pass through the BDD not only produces the exact top event probability but can also provide the information required to calculate the structural importance of each individual component. The system failure logic is still present in the fault tree structure however the quantification process is achieved more efficiently by employing the BDD approach.

As with all of the existing analysis techniques the BDD approach has some limitations. The main drawback with this method is that incorrect ordering of the basic events can result in an excessively large diagram, which can prove difficult to analyse reducing the efficiency of the method. Over the last decade various investigations have been initiated to attempt to find a general ordering heuristic that will produce a minimal BDD for any fault tree structure (ref. 29). However, for large systems, even with poor ordering schemes, the BDD approach produces a more accurate analysis than that available via the more traditional methods.

The two main dependent failure events investigated in this thesis are that of sequential and standby failures. As with the FTA method, the BDD method can provide a limited representation of a system containing sequential failures and can model hot standby failure events. Warm and Cold Standby failures, however, cannot be represented or analysed using the BDD approach.
4.1 Introduction

The analysis techniques outlined in Chapters 2 and 3 are effective at dealing with systems that contain independent failure events. Although these techniques can be used to analyse systems containing sequential failures, the accuracy of the analysis relies on the ‘strength’ of the dependencies. One way of avoiding such errors is to use a method of analysis that can model dependent failures. As outlined in section 1.5.2.1 Markov models do not require the assumption of independence and can therefore be used to analyse systems containing dependencies such as standby failures, common cause failures and sequential failures.

This chapter aims to describe the foundation and application of the Markov analysis method in a reliability environment.

4.2 The Markov Process

In the 1950’s one class of stochastic process that was used in the industrial sector was the Markov Process. From a reliability perspective a Markov process can be seen as a mathematical model that is useful in the study of complex systems, where a complex system may be seen as one which is subject to certain inspection-repair processes and can therefore be in a number of different states other than the simple ones of working or failed. The Markov technique describes a system in time and space using the laws of probability and can be used to analyse the dependability of repairable systems (ref. 12).

The basic concept of the Markov process consists of two elements, states and transitions between states. The Markov process models the system as a state-transition diagram where the states are represented as circles and the transitions are illustrated as arcs travelling out of and into states. The system being analysed is in
one of these states at all times and from time to time moves to a different state. An analogy can be used to illustrate the concept of a Markov state-transition diagram:

Imagine a group of towns, connected by various roads, and a man in a car who is free to travel from town to town. The towns in this example are equivalent to the states in the Markov model and the car travelling from one town to another corresponds to the system making a transition. This system is illustrated as a state-transition diagram in figure 4.1.

![Figure 4.1 A State-Transition Diagram](image)

The time the man stays in each town is known as the state holding time (ref. 8). All towns may not be directly reached from every state. The towns to which driving is possible correspond to the set of outgoing transitions each state has that specify which other towns are directly reachable from the given town. In figure 4.1 there exists a town which is the required destination of the driver, i.e. town 4. Once the driver has arrived in town 4 he will not leave. This state is known as an absorbing state and is equivalent to system failure in a reliability analysis model.

One of the underlying assumptions with the Markov method is that in any small time period only one transition can occur. In addition to this for the basic Markov approach to be applicable the system must be characterised by a lack of memory, that is, the future states of the system are independent of all past states, except the
immediately preceeding one. This is more commonly cited as the Markov Property (ref. 30), which was built into the Markov process in order to simplify quantification of the state-transition diagram. Without the Markov property the probability of arriving at state K by time t would depend on the conditional probabilities associated with the sequence of states through which the stochastic process passed on its way to state K. The Markov property allows simplification by assuming that the probability of arriving in K by time t is dependent only on the state immediately preceeding state K, therefore:

\[ P[X_k = K | X_{k-1} = a, X_{k-2} = b; \cdots; X_{k-n} = p] = P[X_k = K | X_{k-1} = a] \]

The final assumption associated with the Markov analysis method is that system behaviour must be the same at any time (ref. 6). This implies that transition i is the same at all times in the past and future, i.e. stationary. For any system containing dependent failure events the Markov modelling technique can be used to analyse the system providing the system possess a lack of memory, has identifiable states and stationary transition probabilities.

4.3 Construction of the State Transition Diagram

The first stage of a Markov analysis is to draw the state transition diagram. The construction of the Markov diagram begins with the identification of the initial state, or states, of the system at time \( t=0 \). The initial state/s are generally working states where each component is assumed to be functioning as desired. Therefore for a system comprising two components, A and B, that can be either working or failed the initial state would be, A working, B working. The Markov diagram is developed further by identifying all possible transitions out of the initial state, where each transition i represents the probability of failure for component i, \( Q_i \) for discrete time models, or the rate of failure for component i, \( \lambda_i \) for continuous time models. This process results in the identification of new states. For each new state developed all possible transitions are identified, with \( \mu_i \) representing the repair rate of component i. The state-transition diagram is deemed complete when no further transitions out of any newly developed states can be identified. As the Markov method is used to model dependencies, a system containing dependent failure events will be used to illustrate the construction process.
Consider a two component standby system which, in order to function correctly, requires one component to be working on-line. Given the failure of the primary component the component in standby is brought on-line. For this system it is assumed that a component in standby fails with a rate that is equal to half the failure rate of the component when on-line. Due to this assumption this example system is classed as a warm standby system (ref. 6).

The initial state of the system is that one component is working and one component is in standby. This is represented on the Markov diagram by two initial states, S1 and S2:

\[
\begin{array}{c}
A:W \\
B:S \\
S1
\end{array} \quad \begin{array}{c}
A:S \\
B:W \\
S2
\end{array}
\]

where 'W' symbolises a working component and 'S' represents a component in standby.

The Markov diagram is developed from these two initial states by identifying all possible transitions out of each state. For S1 two possible transitions can occur:

1) The failure of the on-line component A. This results in the standby component B becoming the on-line component while component A is being repaired. The system therefore moves into the state (A:F, B:W), where 'F' represents component failure, with the rate $\lambda_A$.

2) The failure of the standby component B. The system therefore moves into the state (A:W, B:F), with the rate $\frac{1}{2} \lambda_B$.

Similarly, the transitions out of state S2 can be identified. The Markov diagram now becomes:
The state-transition diagram is completed following the identification of all transitions out of states S3 and S4, and any other new state developed, and is shown in figure 4.2.

Figure 4.2 Markov Diagram for a 2 Component Warm Standby System

4.4 Different Types of Markov Models

There exists four different Markov models depending on the state space of the system, $S$, and the time governing a transition, $T$. As both $S$ and $T$ can either be discrete or continuous the following Markov models exist:

1) Discrete State - Discrete Time: The state space is discrete, i.e. there exists a fixed number of identifiable states, and transitions occur at fixed unit intervals. Transitions are represented by a fixed probability.
2) Discrete State - Continuous Time: The state space is discrete and the time between transitions is a random variable. The transitions are represented by a rate.

3) Continuous State - Discrete Time: The state space is infinite and transitions occur at fixed time intervals.

4) Continuous State - Continuous Time: The state space is infinite and the transition can occur at any time.

For reliability problems the Markov models are limited to those having a countable number of states, i.e. models 1 and 2

### 4.4.1 Discrete State- Discrete Time Markov Model

In a discrete time process it is necessary to specify the probabilistic nature of the transitions assuming that the time between the transitions is constant (ref. 30). The earlier analogy can be used to describe a Markov model that is discrete in time and space. In order to simplify the problem only two towns will be considered. The states are numbered S1, representing town 1, and S2, representing town 2. Let the probability that the man moves out of town 1 to be $\frac{1}{2}$ and the probability of moving out of town 2 to be equal to $\frac{3}{4}$. The state-transition diagram for this system is depicted in figure 4.3.

![Figure 4.3 Discrete State- Discrete Time Markov Model](image-url)
Reliability problems are usually of the type that are discrete in state but continuous in time and for the scope of this thesis only this type of model will be discussed in detail. An in-depth account of the analysis procedures used for the discrete state discrete time model can be found in (ref. 30, 31, 32).

4.4.2 Discrete State- Continuous Time Markov Model

For reliability and availability studies of systems with dependent failures the most commonly applied Markov model is discrete in space and continuous in time. This type of model implies that the transitions between states can occur at any time. As only one transition can occur in any small time period transitions now occur between short time intervals \((t, t+dt)\) instead of occurring between time \(n\) and \(n + 1\) as in the discrete case (ref. 33). The time between transitions is called the state holding time, which for the discrete state continuous time model, is exponentially distributed. The exponential distribution holds a prominent position in reliability calculations because it describes so well the behaviour of components and systems in their useful life period. The transitions for the continuous time model are hence constant rates. A single component can be used to describe a Markov model that is discrete in space and continuous in time. For a single component there exists two states, one representing the component in a working condition and one representing component failure. The Markov diagram for a non-repairable component is shown in figure 4.4, where the probability of going from a working state to a failed state in a small time interval is equal to \(\lambda dt\).

![Figure 4.4 Markov Representation of a Non-Repairable Component.](image-url)
For a repairable component the Markov diagram is modified as state 2, S2, is no longer an absorbing state. The probability of repairing a component in a small time interval \( dt \) is represented by \( \mu dt \) in figure 4.5.

![Markov Representation of a Repairable Component](image)

Figure 4.5 Markov Representation of a Repairable Component

4.5 Quantitative Analysis of a Markov Diagram

Depending on the type of system being analysed either the reliability of the system, \( R_s(t) \), or the availability, \( A_s(t) \), requires derivation. Reliability was defined in chapter 2 as being the probability that the system has not failed at any time from time zero up until time \( t \) and is usually the measure of interest for non-repairable systems. The Markov representation of a non-repairable system contains an ‘absorbing state’ which represents system failure. The unreliability of the system is equal to the probability of residing in the absorbing state. For repairable systems the measure of interest is the system’s availability at time \( t \), where availability can be defined as the probability that the system is operating at time \( t \). Since the system cannot be in more than one state at a time, i.e. each state is mutually exclusive, it follows that the sum of the probabilities of being in any subset of the Markov model’s states is also a valid probability. Therefore to obtain the availability of the system at time \( t \) the probabilities of residing in a working state are summed together. Alternatively the system’s unavailability, \( Q_{sys}(t) \), can be obtained by the summation of the probabilities of being in a failed state at time \( t \). The identification of the working or failed states is problem dependent.
These definitions indicate that following the evaluation of a Markov model the measure of interest is a probability. In order to analyse a system using a Markov approach the probability of residing in state i at time t must be determined.

**4.5.1 Obtaining the Probability of Residing in State i at time t**

The key element in finding a procedure for determining the probability of being in an individual state at time t is to concentrate on the change in probability with respect to time for each state (ref. 8). To illustrate the procedure of determining the probability of residing in state i at time t the Markov representation of a single repairable component, given in figure 4.5, can be used.

The probability that the system is working at time \( t + dt \) is denoted by \( P_{SI}(t + dt) \). There exists two mutually exclusive ways in which the component can be working at time \( t + dt \):

i) The component can be working at time \( t \), i.e. residing in state S1, with no failure occurrence in the time interval \( dt \).

   OR

ii) The component can be failed at time \( t \), i.e. residing in state S2, and repaired in the interval \( dt \).

The probability of residing in state S1 at time \( t + dt \) is therefore expressed as:

\[
P_{SI}(t + dt) = P_{SI}(t) \cdot (1 - \lambda dt) + P_{S2}(t) \cdot (\mu dt)
\]

which equates to:

\[
\frac{P_{SI}(t + dt) - P_{SI}(t)}{dt} = -\lambda P_{SI}(t) + \mu P_{S2}(t)
\]

As \( dt \to 0 \):

\[
\frac{dP_{SI}(t)}{dt} = -\lambda P_{SI}(t) + \mu P_{S2}(t)
\]

Therefore,

\[
\frac{dP_{SI}(t)}{dt} = -\lambda P_{SI}(t) + \mu P_{S2}(t)
\]
Intuitively, by inspection of equation (4.2), it can be seen that the change in probability for state S1 is simply the difference between the frequency of coming into S1 from state S2 and the frequency of going out of S1 to S2 in the model, i.e.:

\[ P_1(t) = [\text{inflow to state } i] - [\text{outflow from state } i] \]

\[ = \sum_j \left[ \text{rate of transition to state } i \text{ from other states } j \right] \times [\text{probability of being in state } j] \]

\[ - \sum_j \left[ \text{rate of transition from state } i \text{ to other states } j \right] \times [\text{probability of being in state } i] \]

It can be noted that due to the Markov property the probability of residing in state i will always be a 1st order differential equation (ref. 34).

Similarly a 1st order differential equation can be developed to represent the probability of residing in state S2, the failed state, at time \( t + dt \).

\( P_{S2}(t + dt) \) can be achieved in two mutually exclusive ways:

i) The component can be failed at time \( t \), i.e. residing in state S2, with no repair in the time interval \( dt \).

OR

ii) The component can be working at time \( t \), i.e. residing in state S1, and fail in the interval \( dt \).

The probability of residing in state S2 at time \( t + dt \) is therefore expressed as:

\[ P_{S2}(t + dt) = P_{S2}(t) \cdot (1 - \mu dt) + P_{S1}(t) \cdot (\lambda dt) \]  \hspace{1cm} (4.3)

which equates to:

\[ \frac{P_{S2}(t + dt) - P_{S2}(t)}{dt} = \lambda P_{S1}(t) - \mu P_{S2}(t) \]

Therefore,

\[ \frac{dP_{S2}(t)}{dt} = \lambda P_{S1}(t) - \mu P_{S2}(t) \]  \hspace{1cm} (4.4)
Hence for a system with 2 states, the Markov diagram is represented at time $t$ by a system of 2 simultaneous first order differential equations. Generally then, a Markov diagram with $n$ states can be expressed at time $t$ by a system of $n$ simultaneous 1st order differential equations, one for each state (ref. 34).

The differential equations given in equations (4.2) and (4.4) can be written in matrix form, i.e:

\[
\begin{pmatrix}
\dot{P}_{s1}(t) \\
\dot{P}_{s2}(t)
\end{pmatrix} =
\begin{pmatrix}
P_{s1}(t) \\
P_{s2}(t)
\end{pmatrix}
\begin{bmatrix}
-\lambda & \lambda \\
\mu & -\mu
\end{bmatrix}
\]

therefore $\dot{P} = PA$ where the square matrix $A$ is referred to as the differential matrix (ref. 31) or the transition rate matrix (ref. 6). The transition rate matrix can be formulated directly from the Markov diagram where the element $A(i,j)$ represents the transition rate from state $i$ to state $j$ and $A(i,i)$ represents the transition rate out of state $i$, which is always negative. The rows in the $A$ matrix, therefore, sum to zero. The matrix representation is generally used for Markov analysis as the transition rate matrix can be easily developed from the Markov diagram, avoiding the need to identify each states differential equation. The solution of $\dot{P} = PA$ yields a vector of state probabilities at the specified time $t$.

Solution of the differential equations is dependent on whether a steady-state solution or a transient solution is required.

**4.5.2 Steady-State Solution for a Markov Diagram**

A steady-state solution describes the limiting probability of residing in state $i$ after a large passage of time. The steady-state or limiting probabilities of an ergodic Markov process, i.e. a process where the limiting state probabilities are not dependent on the initial conditions, will be non-zero. Limiting state probability can be found for availability systems, however they have little significance in reliability systems as the probability of residing in the absorbing state is unity.

For the single repairable component system, the steady-state probabilities for state $S1$ and $S2$ are obtained by solving the equations:
As equations (4.5) and (4.6) are not linearly independent an additional equation is required in order to solve for $P_{S1}(t)$ and $P_{S2}(t)$. As the system must be either working or failed at any time $t$ the additional equation used is:

$$P_{S1}(t) + P_{S2}(t) = 1 \quad (4.7)$$

Using equation (4.7), equation (4.5) becomes:

$$0 = -\lambda P_{S1}(\infty) + \mu(1 - P_{S1}(\infty))$$

$$\therefore P_{S1}(\infty) = \frac{\mu}{\lambda + \mu}$$

Similarly,

$$P_{S2}(\infty) = \frac{\lambda}{\lambda + \mu}$$

For a system containing one component this process of finding the steady-state probabilities is trivial, which would not be the case for larger systems. It is general practice to solve the Markov diagram, for steady-state, using $P = PA$.

For a steady-state solution $P_i(t) = 0$ hence $PA = 0$. As the equations are not linearly independent the additional equation $\sum_{i=1}^{n} P_i(t) = 1$ is substituted into $PA$, which results in $PA$ becoming a system of linear simultaneous equations which can be solved using a number of different techniques. One of the most popular of the many methods available to solve a system of linear simultaneous equations is the Gauss Elimination method.

To illustrate the process of obtaining a steady-state solution a two component cold standby system can be utilised. For a cold standby system the components which are in standby are assumed to be perfectly reliable, i.e. have zero failure rates. Once brought on-line component $i$ fails with the rate $\lambda_i$. For this system it is assumed that
the system will function as long as one component is working on-line. The initial states in the Markov diagram are therefore (A:W, B:S) and (A:S, B:W). The Markov diagram is developed by identifying all possible transitions out of each newly created state. The Markov diagram for the 2 component cold standby system is shown in figure 4.6.

The transition rate matrix for the Markov diagram in figure 4.6 is equal to:

\[
[A] = \begin{bmatrix}
-(\lambda_A) & 0 & \lambda_A & 0 & 0 \\
0 & -(\lambda_B) & 0 & \lambda_B & 0 \\
0 & \mu_A & -(\mu_A + \lambda_B) & 0 & \lambda_B \\
\mu_B & 0 & 0 & -(\mu_B + \lambda_A) & \lambda_A \\
0 & 0 & \mu_B & \mu_A & -(\mu_A + \mu_B)
\end{bmatrix}
\]

For a steady-state solution:

\[
\begin{bmatrix}
P_1 \\
P_2 \\
P_3 \\
P_4 \\
P_5
\end{bmatrix} = \begin{bmatrix}
0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

In order to be able to perform Gauss elimination, to obtain a steady-state solution for each state, the \( A \) matrix requires modification. The matrix must be transposed and the additional equation, \( P_1 + P_2 + P_3 + P_4 + P_5 = 1 \), included. The system is now represented by the following set of linear simultaneous equations:
which can be solved using Gauss elimination to give the following steady-state probabilities:

\[
\begin{align*}
P_1(\infty) &= \frac{\lambda_B \mu_A \mu_B (\lambda_B + \mu_A + \mu_B)}{K} \\
P_2(\infty) &= \frac{\lambda_A \mu_A \mu_B (\lambda_A + \mu_A + \mu_B)}{K} \\
P_3(\infty) &= \frac{\lambda_B \lambda_A \mu_B (\lambda_A + \mu_A + \mu_B)}{K} \\
P_4(\infty) &= \frac{\lambda_B \lambda_A \mu_A (\lambda_B + \mu_A + \mu_B)}{K} \\
P_5(\infty) &= \frac{\lambda_B \lambda_A \lambda_B \lambda_A + \lambda_B \mu_B + \lambda_A \mu_A}{K}
\end{align*}
\]

where

\[
K = \mu_A^2 \mu_B \lambda_A + \mu_A \mu_B \lambda_A^2 + \mu_A \mu_B^2 \lambda_A + \mu_A \mu_B \lambda_B^2 + \mu_A \lambda_A \lambda_B^2 + \lambda_A^2 \lambda_B^2 + \lambda_B^2 \mu_B \lambda_A + \mu_B^2 \lambda_B \lambda_A
\]

As the system functions correctly providing one component is working on-line, the system fails when no components are working on-line, i.e. state 5. The steady-state unavailability for this 2 component cold standby system is therefore equal to:

\[
\frac{\lambda_B \lambda_A (\lambda_B \lambda_A + \lambda_B \mu_B + \lambda_A \mu_A)}{K}
\]

Even for this small system the algebraic solution was determined using a computer. This highlights one of the difficulties encountered when solving a Markov model, even for a steady-state solution. Unless the system is trivial with a small number of states in the Markov representation, a computer will be required to solve the simultaneous equations.
In addition to the steady-state probability for each state, and hence the system, a useful measure is the mean time duration in each state. Following the identification of this parameter the analyst can identify:

- **Mean Time Between Failures (MTBF)**
- **Mean Time To Repair (MTTR)**
- **Mean Up Time (MUT)**

where

\[ \text{MUT} = \text{MTBF} - \text{MTTR} \quad (4.8) \]

The mean time duration that a system resides in a particular state can be obtained using the transition rate matrix, \( A \). Consider state \( i \) and let \( T_i \) equal the time of residence of \( i \), assuming that entering \( i \) occurs at time 0. The probability of entering \( j \) between \( t \) and \( t + dt \) is found by utilising the fact that the system was in state \( i \) at time 0 and moved to \( j \) in \( dt \). The probability that no transition occurs from state \( i \) to any state \( k \), where \( k \) is not equal to \( j \), prior to \( t \) is given by \( e^{-a_{ij}t} \) and the probability that a transition occurs from \( i \) to \( j \) in \( dt \) is given by \( a_{ij} e^{-a_{ij}t} dt \). Therefore the probability of entering \( j \) between \( t \) and \( dt \) is represented by:

\[
\left( \prod_{k=1, k \neq j}^{p} e^{-a_{ik}t} \right) \cdot a_{ij} e^{-a_{ij}t} dt
\]

\[
= a_{ij} e^{-a_{ij}t} dt
\]

As the system could leave from any state \( i \) to enter any state \( j \) the time duration in state \( i \) can be found to be equal to:

\[ T_i = -a_{ii} e^{a_{ii}t} dt \quad (4.9) \]

From equation (4.9) the probability density function \( f_{T_i}(t) \) can be deduced:

\[ f_{T_i}(t) = -a_{ii} e^{a_{ii}t} \quad (4.10) \]

The residence time in state \( i \) therefore follows an exponential distribution and the mean duration in state \( i \) is given by:
\[
\bar{T}_i = \frac{1}{-a_n} = \frac{1}{\text{Rate of departure from state } i}
\] (4.11)

Using this definition and denoting \(\bar{T}_i\) as the mean residence time in other states before returning to state \(i\), it can be shown that the asymptotic frequency of encountering state \(i\), \(f_n\), is given by equation (4.12) (ref. 12).

\[
f_n = -a_n \cdot Q_1(\infty)
\] (4.12)

Using equations (4.11) and (4.12) the MUT, MTBF and MTTR can be obtained for any system. For example taking figure 4.2, the Markov representation of a 2 component warm standby system, there exists only one failed state, S5, therefore:

\[
\begin{align*}
\text{MTTR} &= \bar{T}_5 = \frac{1}{\mu_A + \mu_B} \\
\text{MTBF} &= f_n = \frac{1}{-a_{55} \cdot P_5(\infty)}
\end{align*}
\]

and MUT is calculated using equation (4.8).

The Markov process which contains an absorbing state is not an ergodic process and hence the steady-state probability offers little information regarding the system’s performance behaviour. What can be found by analysing a system containing an absorbing state is the Mean time to first failure (MTTFF). Billington and Singh (ref. 39) stated that MTTFF could be calculated using:

\[
\text{MTTFF} = [1-Q]^{-1}, \quad \text{where } Q \text{ represents the probability matrix truncated by removal of all rows and columns that correspond to an absorbing state.}
\]

For a single non-repairable component the Markov diagram has the form depicted in figure 4.4 with the probability matrix, \(P\), equal to \[
\begin{bmatrix}
1-\lambda & \lambda \\
0 & 1
\end{bmatrix}
\]

Truncating \(P\) due to the absorbing state, state 2, results in \(Q\). Hence \(Q = [1-\lambda]\). The mean time to first failure is therefore equal to:

\[
\text{MTTFF} = [1-(1-\lambda)]^{-1} = \frac{1}{\lambda}
\]
Alternatively the MTTFF can be found by matrix manipulation (ref. 12). Let $A_k'$ denote the reduced matrix with only working state transition present then MTTFF is given by:

$$
MTTFF = \frac{1}{\prod_{k} |A_k|}
$$

4.5.3 Transient Solution for a Markov Diagram

A transient solution describes the probability of residing in a certain state at time $t$, and is useful for analysing systems that are only required to function for a short time period, i.e. a safety system which is brought on-line in order to alleviate a hazardous event. In order to obtain a general solution which represents the probability of being in a certain state as a function of time, the matrix of differential equations, equation (4.13), must be solved.

$$
\dot{P}(t) = (P(t))A
$$

(4.13)

When the initial state of the system, $[P(0)]$, is known it is possible for a true solution to be found for equation (4.13). As with the steady-state example there exists a variety of techniques that can be used to solve a system of coupled linear differential equations. The usual methods employed, in relation to a Markov diagram, can be divided into two categories:

(i) Full Analytical Resolution - through Laplace transformation and eigenvalue approaches.

(ii) Direct Numerical Resolution - through numerical integration of O.D.E with step by step methods.

Full Analytical Resolution

For the first category a commonly cited technique is the eigenvalue approach which involves the identification of the eigenvalues, and subsequently the eigenvectors, of the system (ref. 5, 35). This method, however, has been shown to be problematic for very reliable systems and inaccurate for Markov diagrams with a large number of
states (ref. 12). Further problems with the full analytical resolution methods were highlighted following the comparison between two indirect exponential methods, a Laplace transformation method and an integration method (ref. 36). The results of the comparison indicated that the O.D.E integration method used was the most efficient method, with good accuracy and computing time. The remaining methods, although accurate, required an excessive amount of computation due to the characteristics of the methods. Developments in the analytical methods have been achieved to a certain extent (ref. 37), but numerical integration techniques are presently favoured.

**Numerical Integration Resolution**

Numerical integration techniques provide the probability of residing in state i at time t by progressing from the initial situation, \[ P(0) \], in very small time steps. One of the most popular methods available to solve first order differential equations is the Runge-Kutta method (ref. 6). The main problem surrounding the solution of equation (4.13), using integration techniques, is that the transition rate matrix may be sparse and contain a degree of 'stiffness', i.e. failure and repair rates differ greatly in magnitude. Tombuyses and Derought postulated that in order to avoid analytical errors a variable stepsize integration technique should be employed, such as an implicit trapezoidal rule (ref. 38).

It should be noted that obtaining a transient solution for any Markov diagram containing more than a few hundred states is problematic and in such cases simulation will generally be employed in order to produce an accurate solution.

**4.6 Conclusion**

Any system that is characterised by a lack of memory, has an identifiable number of states and contains constant transition rates can be modelled using the Markov analysis techniques outlined in this Chapter. The Markov analysis method has been shown to be particularly useful for modelling systems which contain dependent failure events. It can therefore be used to accurately analyse sequential and standby failures. For any system modelled a number of reliability measure can be calculated including system's unavailability and the mean residence time in each state.
The method, however, has limitations. The main disadvantage lies in the size of the Markov representation for a system. Even for a relatively small system the Markov diagram can become large and complex, which can lead to construction problems. Attempts have been made to alleviate this problem by 'merging' similar states (ref. 34, 40), i.e. states containing 2 failed and 1 working components. This reduction technique, though, can only be used for systems which contain identical failure and repair rates for each component and is therefore not always applicable.

Analysis of a Markov diagram is efficient for small systems but requires complex solution techniques when the Markov diagram is large. A further disadvantage is that the diagram holds no logical description of how the system functions or how the model was developed and for this reason is not as appealing to engineers as the FTA method.

Although the Markov method has limitations it nevertheless can be used to accurately and efficiently describe and analyse small systems containing dependent failure events.
CHAPTER 5

CAUSE-CONSEQUENCE DIAGRAM METHOD

5.1 Introduction

In many industrial systems, where safety is of the utmost importance, it is necessary that expedient tools for accident analysis are available and employed at the design stage. Such tools must be able to handle large systems in a systematic way and be helpful at displaying the factors that are of vital importance for the safety of the system. The previous four chapters have outlined several methods which attempt to achieve this for systems characterised by different features. This chapter describes a method that contains capabilities superior to FTA, BDD and Markov methods for sequentially operating systems.

While investigating a nuclear power plant Nielsen (ref. 14) noticed that a given accident can be characterised by a ‘cause’, a sequence of events where the time between their occurrence can be an important parameter and finally by a consequence. This type of system could be characterised as one with various shut-down mechanisms that are initiated given the presence of some initiating event, i.e. a pressure limit is exceeded. In order to identify all relevant accidents for a such a system Nielsen stated that the tool used must be able to determine the possible causes of the accident event and identify the possible consequences given that one or more of the accident limiting provisions could fail.

Previously discussed techniques, such as FTA and Markov analysis, are incapable of identifying both the possible causes of an undesirable event AND all the possible consequences resulting from it. A technique, however, has been developed that possesses the ability to identify the causes of an undesired event and from this event develop all possible system consequences. The technique is known as the Cause-Consequence Diagram method. The Cause-Consequence Diagram method was developed at RISO National laboratories, Denmark, in the 1970’s to specifically aid in
the reliability and risk analysis of nuclear power plants in Scandinavian countries (ref. 12). The method was created to assist in the cause-consequence accident analysis of the nuclear plants, which involved identification of the potential modes of failure of individual components and then relating these causes to the ultimate consequences for the system (ref. 41). The method can be seen as superior to Event Tree Analysis, which is also capable of identifying all consequences of a given critical event, as it models at component level and therefore is functionality driven and not subsystem driven. In addition to this the cause-consequence diagram method can account for time delays which is not a feature available in the ETA method.

This chapter aims to describe the development of the cause-consequence diagram method and its uses in a reliability environment.

5.2 The Cause-Consequence Diagram Method

In 1971 Nielsen proposed a method for an expedient presentation of the logical connections between a "spectrum" of accident causes and a "spectrum" of relevant consequences (ref. 14). Nielsen stated that as well as being a tool for illustrating the consequences of particular failures the method could also serve as a basis from which the probability of occurrence of the individual consequences could be evaluated. The consequences evaluated include both the system terminating as intended and an incorrect termination. As all consequence sequences are investigated the method can assist in the identification of system consequences that might have not been contemplated at the design stage.

The main principle of the technique is based on the occurrence of a critical event, i.e. an event that disturbs the balance of the process plant. The identification of the critical event is problem dependent and choosing the correct place to start is important as there may be many possible initial events, not all of which have serious consequences. Identification of a true critical event is imperative for large systems as a great amount of work can be wasted by analysing an event that does not directly effect the process. Focus should therefore only be made on functional failures of process components that directly effect the plant balance. Once a critical event has been
identified all relevant causes of the critical event and potential consequences are developed using two conventional reliability analysis methods. This situation is represented in figure 5.1.

![Figure 5.1 Simple Representation of a Cause-Consequence Diagram Structure](image)

The two reliability analysis tools used in the development of the cause-consequence diagram method are the fault tree analysis method, described in chapter 2, and the event tree analysis method, outlined in section 1.5.3. The FTA method was used by Nielsen in two situations to describe the causes of an undesired event. Firstly the technique was used to describe the causes of the critical event. The second function for the fault tree method was to describe the failure causes of the accident-limiting systems (emergency shut-down systems). The event tree method was used as the link between the causes of the critical event and the various consequences that could result. The ETA method was used to identify the various paths that the system could take, following the critical event, depending on whether certain subsystems/components functioned correctly or not.

The relationship between the two reliability methods is shown in figure 5.2.
5.2.1 Symbols for Construction

The symbols applied by Nielsen for the construction of a cause-consequence diagram are depicted in figure 5.3. It can be seen that the symbols used for the ‘cause’ part of diagram are simply those used for a fault tree structure. For the ‘consequence’ part, though, new symbols were developed. The main symbol used in the construction of the consequence diagram is the decision box (figure 5.3). The decision box created by Nielsen is an identical representation of the ‘YES-NO’ branches seen on an event tree structure. The connection point between the consequence and cause diagrams is the NO outlet path of these decision boxes as the failure causes of a system, represented by a decision box, are developed using FTA. A further important symbol, in terms of identifying the correct consequence, is the delay box. The delay symbol is used in systems where timing of event failures is relevant. In the analysis of such systems the time that passes between influential sequential failures is important as the knowledge of this may help the analyst differentiate between different system outcomes. In addition to this the time delay provides a means of taking into account random failures which could occur in a time interval following the occurrence of the critical event.
<table>
<thead>
<tr>
<th>SYMBOLS FOR CAUSE DIAGRAM</th>
<th>FUNCTION DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>![AND Gate Symbol]</td>
<td><strong>AND GATE:</strong></td>
</tr>
<tr>
<td></td>
<td>Allows causality to pass up through the tree if at any time all inputs to the gate occur.</td>
</tr>
<tr>
<td>![OR Gate Symbol]</td>
<td><strong>OR GATE:</strong></td>
</tr>
<tr>
<td></td>
<td>Allows causality to pass up through the tree if at any time at least one input to the gate occurs.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SYMBOLS FOR CONSEQUENCE DIAGRAM</th>
<th>FUNCTION DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Direction Of Events Symbol]</td>
<td><strong>Direction Of Events:</strong> Symbolises the direction from one event to the next event in a particular sequence.</td>
</tr>
<tr>
<td>![Decision Box]</td>
<td><strong>Decision Box:</strong> Represents the functionality of a component/subsystem. Mutually exclusive events.</td>
</tr>
<tr>
<td>![Delay Symbol]</td>
<td><strong>Delay Symbol:</strong> Represents a time interval</td>
</tr>
<tr>
<td>![Consequence Box]</td>
<td><strong>Consequence Box:</strong> Holds information concerning the accident/consequence description</td>
</tr>
<tr>
<td>![Comment Box]</td>
<td><strong>Comment Box:</strong> Holds definition of critical event.</td>
</tr>
</tbody>
</table>

*Figure 5.3 Symbols used in Cause-Consequence Diagram*

The overall structure of the cause-consequence diagram method proposed by Nielsen in 1971 is depicted in figure 5.4.
5.2.3 Rules of Construction for a Cause-Consequence Diagram

Rules for the construction of a cause-consequence diagram were detailed by Nielsen in two sections, those for the cause part of the diagram and those for the consequence part of the diagram. For the cause part it should be noted that many of the rules postulated were similar to those used in the construction of a fault tree structure. The
rules outlined by Nielsen for the construction of the cause diagram can be summarised as a three step procedure:

1) **Identification of the Top Event**: The construction of the cause diagram begins with the definition of the undesired event, i.e. the system failure of interest.

2) **Cause Diagram Development**: Using a deductive process the causes of the undesired event are discovered and connected by means of logical gates. The procedure is repeated until all events have been fully developed, i.e. the basic events are reached.

3) **Validation of the Diagram**: For each gate used all inputs must be both necessary and sufficient to produce the output event. If this is not the case then the diagram will not be valid for probability analysis.

Similarly a set of rules were devised for the construction of the consequence diagram which show a certain degree of similarity to those used for the construction of an event tree:

1) **Definition of a Critical Event**: The starting point is an examination of all relevant operating conditions with reference to a definition of a critical event, i.e. an event that initiates a certain sequence of events to occur.

2) **Consequence Diagram development**: A logical clearing-up of connections between events and conditions is carried out in order to determine possible consequences. For each sub-system/component only two states are normally taken into account, correct and incorrect functioning. Given a critical event certain subsystems/components will be activated. The consequence diagram is developed by evaluating the state of the system depending on whether these subsystems/components function correctly or not.

3) **Graphical Symbols**: The difference between the event tree structure and the consequence diagram of the cause-consequence structure is the graphical symbols used. Using these symbols forces the analyst to follow up and present all possible sequences.
Nielsen’s work was further developed in the late 1970’s by Taylor, a researcher at RISO. The construction process for a cause-consequence diagram was detailed in Taylor’s work on creating an effective interlock design for a chemical plant (ref. 42). Taylor stated that following the identification of a critical event the consequences of that event should be traced along the plant component interconnections. At each step, as a new component is encountered, all of the possible further events are evaluated depending on the state of the component, i.e. working, failed, open or closed. The result is an event tree, the consequence part of the cause-consequence diagram.

In addition to the rules for the construction of the cause-consequence diagram Nielsen outlined a three step procedure for the correct accident analysis of a nuclear system and in 1974 extended it to a seventeen step procedure. The cause-consequence diagram, or cause-consequence chart, is an integral part of the cause-consequence analysis. It should be noted that a cause-consequence analysis represents the entire process of identifying the relevant accident causes and their consequences and not simply the analysis.

### 5.3 Construction Example

To illustrate the construction of a cause-consequence diagram, using the rules outlined in section 5.2, a simplified domestic hot water system depicted in figure 5.5 can be used. In this system the gas valve is operated by a temperature measuring device. The gas valve operates the burner in a full-on or full-off mode with a constantly lit pilot light providing the ignition source. Water temperature control is achieved by the controller opening the main gas valve when the temperature drops below the preset limit. The burner ignites and heats the water. When the upper temperature limit is reached the gas valve is closed. As a safety precaution a relief valve is included in the system configuration which opens when the pressure reaches 100psi. In the situation where the temperature limit is exceeded the tank pressure will rise and could result in tank rupture. It is assumed that the tank will rupture if the pressure exceeds 100psi. The component functions and failure modes are listed in table 5.1.
The critical event for the domestic water system is that the pressure exceeds 100psi causing the tank to rupture. Having identified the critical event the next stage in the construction process is to identify all possible causes of the event. A pressure increase to 100psi and above can only be achieved if the water is heated beyond its normal maximum temperature. There exists two situations which can cause this event. Firstly the gas valve can fail to close when instructed to by the controller and secondly the gas valve is not closed due to a failure within the temperature monitoring loop. The ‘cause’ diagram depicted in figure 5.6 shows the causes of the pressure exceeding 100psi.
Pressure exceeds 100psi

No signal to shut valve

Gas valve fails open

SH
CH

Figure 5.6 The Cause Diagram for the critical event: 'Pressure Exceeds 100psi'

Following the identification of the critical event and its causes, the final stage in the process of constructing a cause-consequence diagram is to develop the relevant consequences of the critical event on the system. For the domestic water system there exists only one safety feature, the relief valve. The consequences of the pressure exceeding 100 psi therefore depends on the functionality of the relief valve. A decision box is created which represents the correct functioning of the relief valve, i.e. relief valve opens. Following the NO outlet branch from the decision box indicates that the relief valve has failed closed. The effect of the relief valve failing to open is that the tank will rupture. Following the YES outlet branch, however, indicates that the relief valve has functioned correctly and results in a decreased pressure and no tank rupture. The decision box, representing the functionality of the relief valve, and its consequences make up the consequence part of the cause-consequence diagram (Figure 5.7).

Figure 5.7 The Consequence Diagram for the critical event 'Pressure Exceeds 100psi'
The cause-consequence diagram representing the domestic hot water system is hence developed by the coupling of the cause diagram, figure 5.6, and the consequence diagram, figure 5.7, at the critical event comment box. Figure 5.8 shows the cause-consequence diagram for the domestic hot water system.

![Figure 5.8 Cause-Consequence Diagram for the domestic hot water system](image)

5.4 Quantitative Analysis of a Cause-Consequence Diagram

Besides being used in connection with cause and consequence identification, the cause-consequence diagram offers systematic support for probabilistic modeling (ref. 14). In 1971 Nielsen used a standby pump system to illustrate the analysis of the cause-consequence diagram method. The probability that a total loss of flow from the pump was calculated using integrals of the probability distribution functions. The cause diagram for the critical event was solved using FTA, as were the cause trees used to represent the NO outlet branches of the decision boxes. The inclusion of the delay symbol meant that some sequences included integration of the probability distribution function where a failure could occur in a certain time interval. In order to gain an expression to represent the event 'No Flow' two calculations were made.
Firstly the probability of each sequence leading to the outcome ‘No Flow’ was determined using the relevant time delays and integrals of the probability distribution functions. Following this all sequences that led to the outcome, ‘No flow’, had their respective probabilities of occurrence summed together to yield the total probability of the event. The analysis of this dependent system is possible due to the inclusion of the delay symbol and the fact that the cause-consequence diagram can model events that occur in a particular sequence. Although the main advantage of the cause-consequence diagram technique can be seen as its ability to model dependent systems, it can also be used as an alternative method to the FTA technique for independent systems.

5.4.1 Quantification of a System Containing Independent Failure Events

The procedure for analysis of an independent system begins with the assignment of probabilities/frequencies to each outlet branch stemming from a decision box. Following this the probability of any one sequence is obtained by multiplication of the probabilities associated with each decision box in that sequence. The probability of any particular consequence is then obtained by the summation of the probability of each sequence that terminates in that consequence. This quantification process can be seen to be identical to the analysis procedure outlined in chapter 3 for the BDD method. To illustrate the quantification process of a system containing independent failures the cause-consequence diagram constructed in section 5.3 and shown in figure 5.8 can be used. The probability of failure is equal to the probability of ending in the consequence ‘TR’. The probability of the sequence path leading to the consequence TR can be shown to be equal to:

\[ P(\text{TR}) = P(\text{Critical Event}) \cdot Q_{RVc} \]

where the probability of the critical event is given by:

\[ P(\text{Critical Event}) = 1 - (1 - Q_{SH})(1 - Q_{CH})(1 - Q_{GVO}) \]

The probability of tank rupture via the cause-consequence analysis can be shown to be identical to that obtained from a FTA. The cause-consequence diagram method offers nothing more than an alternative representation of the system which contains independent failures. Hickling postulated that neither the cause-consequence analysis
method described for an independent system or the FTA method could model time
dependent accident sequences (ref. 43). Hickling suggested, however, that the cause-
consequence diagram method had features which could be used in order to model such
time-dependent systems.

5.4.2 Quantification of a System containing Dependent Failure Events

A detailed analysis procedure for a system containing dependent failures was outlined
in a paper that investigated the system failure for a 2-unit standby system with
imperfect switching (ref. 41). The system comprised three parts, an operative unit, a
standby unit and a switching unit, which is used to bring the standby unit into action
following the failure of the operative unit. The cause-consequence diagram, with no
repair, is depicted in figure 5.9. In order to find the probability that the system failed
during the time interval 0 to T the probability of sequence 1, 2 and 3 had to be
determined. It can be noted that the critical event for the cause-consequence diagram
shown in figure 5.9 does not have a cause diagram to describe its failure causes. This
is because the critical event, A fails in time t₁, can only be caused by A failing.

The probability that sequence 1 occurs is given by: *The probability that A fails in time
t₁ and the operator fails to switch to the standby unit B:*

\[ P_{s1}(T) = k.F_A(T) \]

where \( k \) is the probability that the operator fails to switch and \( F_A(T) \) is the cumulative
failure distribution for the component A, where A can fail any time between 0 and T
to cause system failure during the time interval 0 to T.

The probability of sequence 2 is given by: *The probability that A fails in time t₁, the
operator switches to the standby component B but B has failed while in standby.*

\[ P_{s2}(T) = \overline{k} \int_{t_0}^{T} f_A(t_1).S_B(t_1)dt_1 \]

where \( \overline{k} \) is the probability that the operator performs the switching action successfully
and is equal to (1-k).

The term \( f_A(t) \) represents the probability density function for time to failure of
component A and \( S_B(t) \) is the cumulative failure distribution for the standby unit B.
Similarly the probability of sequence 3 is identified. The sequence occurs if component A fails in time $t_1$, the operator switches to the available standby, component B, which then fails prior to time $T$.

$$P_{33}(T) = k \int_0^T f_A(t_1).(1 - S_B(t_1)).F_B(T - t_1)dt_1$$

It can be noted that the expression given by Nielsen and Runge for sequence 3 models a dependent failure as the order in which the events occur is relevant. Component B must be available before it can fail on-line. This dependency was modeled by manipulation of the cumulative density function, $F_B(t)$, where $F_B(T-t_1)$ indicates that component B fails at some point between time $t_1$ and $T$. This calculation procedure, however, cannot be derived from inspection of the cause-consequence diagram for the standby system. This highlights one of the fundamental weaknesses with the cause-consequence diagram method as an analyst would not know how to quantify the system from the diagram alone. In this case the quantification procedure was developed independently of the diagram.
In addition to the non-repairable case Nielsen & Runge investigated the repairable case. The analysis of the system was complex and only an approximation to the probability of failure was given. It was stated that one of the problems with the analysis procedure used was that the procedure did not account for the importance of relevant time delays in the system. For example the same probability was given to the action of switching regardless of whether it took 1 minute or 3 hours to switch. A switching action that takes an extended period of time is equivalent to no switching action at all. Such features were not incorporated in Nielsen’s quantification of the switching system.

In 1975 Nielsen, Platz and Runge used the cause-consequence diagram method to analyse a redundant protection system (ref. 44). The accident analysis performed asked the question: "Does a given protection system perform the required function under the relevant conditions?". A cause-consequence diagram was used to provide an answer. The protection system analysed was a core spray system in a nuclear boiling-water reactor. The system was used to prevent the fuel core from overheating given a loss of primary coolant. The system consisted of 2 identical core spray systems, A and B, each of 100% capacity. Therefore failure occurred when both core spray systems had failed to function. The configuration of the system can be seen in figure 5.10. The blacked out valves indicate normally closed motorised valves and the valves labelled ‘c’ are non-return valves.

In each system a pump, (P1,P2) takes suction from the suppression pool and coolant there flows to a nozzle through a series of valves. The individual cooling system, A, can be considered as 2 subsystems, A1 and A2, where A1 contains: F2, V1, P1, C1, V2, V6, V4, V3 and A2 contains: V5 and C2. Similarly B1 contains: F1, V7, P2, C3, V8, V9, V10 and B2 contains: V11 and C4.
Two main failure modes were identified that would cause the overall failure of either of the two core spray systems:

i) The system is unavailable when a demand is present.

ii) The system is available but fails before the required operating time $t_0$ is over.

The probability of system failure was given as the probability that the system was unavailable given a demand for a core spray OR that the core spray is established but stops before the required operating time is over. The cause-consequence diagram was constructed from the critical event loss of coolant and is given in figure 5.11. For this system it was assumed that the subsystem A2 and B2 could not fail once activated,
due to the fact that the components contained within these are passive. The cause-
consequence diagram was hence constructed by inspecting the availability of A2 and
B2 first.

The quantification of $P_u$, the unavailability of the system given a demand, was
obtained via the summation of the probability of 4 sequence paths terminating at the
outcome ‘NA’. As no dependencies existed between any of the subsystems, A1, A2,
B1 or B2, $P_u$ was obtained from the cause-consequence diagram as:

$$P_u = P_{A2}(t)P_{B2}(t) + P_{A1}(t)\bar{P}_{B2}(t) \cdot P_{B1}(t) + \bar{P}_{A2}(t) \cdot P_{B2}(t) \cdot P_{A1}(t)$$

$$+ P_{B1}(t) \cdot P_{A1}(t) \bar{P}_{B2}(t) \cdot \bar{P}_{A2}(t)$$

where $P_i(t)$ represents the probability that subsystem $i$ cannot be activated at time $t$ and
$\bar{P}_i(t)$ implies the complement of $P_i(t)$.

As a measure for the unavailability of the total cooling system, A and B, the average
probability of simultaneous failure of A and B within $T$ was determined, therefore:

$$P_{u_{av}} = \frac{1}{T} \int_0^T P_a(t) \cdot P_b(t) \, dt = \frac{1}{T} \int_0^T P_u \, dt$$

The probability that given a demand the core spray system was available but failed
before time $t_0$, $P_0$, required a time-dependent quantification. Nielsen identified 5
sequences that terminated in the consequence: ‘Failure of core spray before the
required operating time’. $P_0$ was obtained via the summation of the 5 sequence paths
probability which lead to the outcome ‘S’ on the cause-consequence diagram. The
quantification technique used was identical to that used in the switching example
given in (ref. 42). For example the probability of sequence c, given a demand at time
t, was shown to equal:

$$\text{Probability of Sequence } C = \bar{P}_{A2}(t)\bar{P}_{B2}(t)\bar{P}_{B1}(t)P_{A1}(t) \cdot \int_0^{t_0} f_{B1}(t_1)Q_{A1}(t_1) \, dt_1$$

where sequence C is represented by A2, B2 and B1 being available given a demand,
with subsystem A1 not available and B1 fails before A1 is repaired. The term
$f_{B1}(t_1) \, dt_1$ represents the probability that subsystem B fails in the time interval $t + t_1$
and $Q_{A1}(t)$ is the cumulative function of time to repair for subsystem A1 therefore
$\bar{Q}_{A1}(t_1)$ represents the cumulative probability that A is not repaired in the time $t+t_1$. 

101
Nielsen concluded that the cause-consequence diagram was a tool that could provide efficient qualitative and quantitative analysis. However, as with the switching example, the quantification of the cause-consequence diagram was completed independently of the diagram and hence an analyst would not know how to quantify the system using the diagram alone.

Figure 5.11 Cause-Consequence Diagram for the Core Spray System.
An alternative quantification technique was postulated by Hickling for analysing time-dependent failures. Hickling stated that the cause-consequence diagram had the ability to model processes that extended over a period of time by using feedback loops, i.e. exit paths from decision boxes are permitted to connect to decision boxes which have already been ‘visited’ in a sequence. A system where the order of failure is important, i.e. a sequentially operating system, can be modelled using such feedback loops. For illustration Hickling presented a plant which comprised of two units: a containment system and a leak detection/isolation system. He postulated that if the containment system failed prior to the failure of the leak detection/isolation system then the isolation system would shut down the plant. However, if the isolation system failed first then until the fault is detected and repaired any failure of the containment system would cause plant failure. The cause-consequence diagram, containing feedback loops, for this system is shown in figure 5.12. By inspection of figure 5.12 it can be seen that Hickling uses the YES outlet branch to symbolise failure. For this type of construction process the fault trees representing the failure cause of the decision box would be attached to the YES outlet branch instead of the NO outlet branch as with Nielsen's work.

Inclusion of the feedback loops implies that the input to a decision box in one period of time depends on the output in the previous period of time, i.e. time-dependent modelling. Hickling stated for such systems quantification could not be achieved by multiplication of the outlet branch probabilities in a given sequence, as before, as component failures are no longer independent. Instead each decision box has an associated failure rate and the expected number of times by time t that the system enters a certain state can be calculated.

Hickling stated that the expected number of times, by time t, that the system enters a dangerous state, $E_D(t)$, is found by determining the rate at which the state is entered, $R_D(t)$, integrated over the time period 0 to t. The failure rate for the containment and isolation systems were defined as $\lambda_c(t)$ and $\lambda_i(t)$ respectively and the rate at which the dangerous state was entered was given as:

$$R_D(t) = P_N(t) \lambda_d(t)$$
and therefore

\[ E_D(t) = \int_0^t P_N(t) \lambda_d(t) dt \]

As with the quantification procedure described for the switching standby system, the quantification of this system could not be deduced from the cause-consequence diagram structure.

Figure 5.12 Cause-Consequence Diagram for Containment-Isolation Plant System.

Hickling also highlighted a less complex analysis where the failure and repair rates were deemed constant and the measure of interest became the probability of reaching an outcome, i.e. Escape to atmosphere. Using this approach the probability of ending in a dangerous state was given by:

\[ E_D(\infty) = E_N(\infty) \cdot \frac{\lambda_f}{\lambda_c + \mu_c} \]
More usefully, following solution of the coupled equations, the probability of ending in the state, escape to the atmosphere, was found to be:

\[ E_{EA(\infty)} = \frac{\lambda_i}{\lambda_e + \lambda_i + \mu_i} \]

This analysis approach holds similarities to that of the Markov analysis methods.

5.5 Further Applications of the Cause-Consequence Diagram Method

Since the development of the cause-consequence diagram method a number of studies have used it to perform risk and reliability assessments. In 1977 the method was used to analyse an instrumental air system (ref. 45). The cause-consequence analysis highlighted that there was a weak link in the ‘dryer section’ of the system and hence the method not only performed a quantitative analysis but also identified an advantageous design modification. In the same year the method was used to analyse a interlock design (ref. 42). Interlocks are switch-like devices intended to prevent relatively simple procedural errors. By using the cause-consequence diagram method it was shown that the inclusion of interlocks produced a safer more reliable system.

More recently the technique was used for the analysis of an automatic burner system (ref. 13). The burner formed part of an experimental rig at the British Gas research station in the midlands. The research showed that a cause-consequence diagram was capable of dealing with ‘complex events’, i.e. events that may occur in different states in the same sequence. The quantification procedure used, however, was not detailed in the report. Since the late 1980’s no further investigation into the cause-consequence diagram technique has occurred.

5.6 Conclusion

The user of a completed plant will largely judge its reliability performance on the costs resulting from faults in terms of loss of production, damage to the plant and injury to staff (ref. 42). It is therefore important to develop systematic methods for cause-consequence accident analysis, relating potential modes of failure to the ultimate consequences for the system. The cause-consequence diagram method outlined in this chapter can be seen to be such a method. One of the advantageous features of this method is that all possible event sequences are identified and the
analyst is forced to study all eventualities following a given critical event. Although this can be achieved via the use of the event tree analysis tool, the cause-consequence method can model complex events which sometimes render the event tree technique inefficient. The main feature of this model is therefore that it can successfully model dependent failure events.

Several authors have used the technique as the main analysis tool for a safety assessment. The inclusion of a time delay symbol allows time-dependent failures to be modelled. Two alternative approaches for analysing time-dependent failures have been outlined in this chapter. The documentation, however, of the quantification of the cause-consequence diagram is limited and a generalised analysis method or even rigorous definitions of the meaning of the symbols to enable diagram quantification is yet to be developed. In both papers outlined in this chapter the quantification was performed independently to the cause-consequence diagram structure. The cause-consequence diagram, in its present form, can therefore be seen as a tool which provides a thorough documentation of the entire failure logic of the system.

The cause-consequence diagram method has great potential for analysing dependent failure events and due to the construction being based on sequence the technique could be used effectively to represent a system that contains sequential failures. The technique has also been shown to be capable of representing systems containing standby failure events. Developments, however, in the use of the diagram to perform quantification analysis is required if this technique is to compete with the traditional methods such as FTA and Markov analysis.
CHAPTER 6
ANALYSIS OF SYSTEMS WITH DEPENDENT FAILURE EVENTS

6.1 Introduction

For the reliability analysis of industrial systems the FTA technique is generally employed. As discussed in section 1.5.1.1 one of the reasons for its popularity is that the method represents the failure logic of the system in a tree structure, which provides a good documentation of the way the failure logic was developed. In addition to this the technique is extremely efficient and accurate when analysing systems containing independent failure events. When independence is not satisfied alternative approaches, such as Markov methods, are required if an accurate result is to be obtained. Markov methods can be utilised to analyse sequential and standby failures, however, they do not represent the system failure logic as intuitively as a fault tree and are therefore not as appealing to reliability engineers. A further disadvantage of the Markov techniques, as outlined in chapter 4, is that the size of the state-transition diagram, and therefore the number of equations requiring solution, increases rapidly as the number of components in the analysis increases.

To enable the analysis of systems with dependencies and retain the advantages of both the FTA and Markov techniques an investigation into the development of a combined model was initiated. The development of such a model was achieved, in part, by Dugan & Gulati (ref. 47). Dugan & Gulati embedded a Markov analysis option within a fault tree framework thus retaining the structure of the fault tree whilst enhancing the accuracy of the result.

Extending the work initiated by Dugan & Gulati two types of dependent failures have been investigated. The two dependent failure types are sequential failures, where the order of failure is relevant, and standby failures, where the failure probability is conditional on whether the component is in standby or on-line.
This chapter describes the development procedure of a new assessment method that combines the most advantageous features of both the fault tree method and the Markov method. In this way systems containing both independent and dependent components can be analysed accurately and efficiently and thus the superiority of the new method can be highlighted. The developments discussed in this Chapter are summarised in (ref. 46).

6.1.2 Fault Tree Analysis Program

Prior to the development of a model to analyse fault trees containing dependent failure events a fault tree program was created to analyse independent fault trees. The general algorithm depicted in figure 6.1 outlines the procedure of the FTA program.

![Figure 6.1 Summary of the FTA program](image-url)
Initially the fault tree structure is stored in a *.ats data file. The general layout of an *
ats file is:

<table>
<thead>
<tr>
<th>NAME</th>
<th>GATE TYPE</th>
<th>KVG/NVG</th>
<th>No. OF GATE INPUTS</th>
<th>No. OF BASIC EVENT INPUTS</th>
<th>INPUTS BY NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Characters Long</td>
<td>4 Characters Long</td>
<td>3 Characters Long</td>
<td>2 Integers Long</td>
<td>2 Integers Long</td>
<td>max of 9 inputs 8 characters long with 2 spaces between each</td>
</tr>
</tbody>
</table>

The ‘NAME’ column corresponds to the label given to a gate or basic event. The gate type represents the type of gate under inspection, where only OR, AND and VOTE gates were considered. The fourth column is used to represent a VOTE gate where KVG indicates the number of gate inputs that need to fail out of the possible NVG inputs to cause the VOTE gate to occur. The number of gate inputs and basic event inputs are shown in the 5th and 6th columns respectively with their names given in the final column.

To illustrate the creation of the *.ats file the simple fault tree shown in figure 6.2 can be used.

![Figure 6.2 Simple Fault Tree Structure](image)

The *.ats file to represent this fault tree is:

```
TOP OR 0 01 1G1 C
G1 AND 0 00 2A B
```
Following the execution of the program the *.ats file is read in one line at a time. Various arrays are created in order to store the fault tree structure in an integer form which is easy to manipulate. The arrays created are listed in table 6.1.

<table>
<thead>
<tr>
<th>Name of Array</th>
<th>Function of Array</th>
</tr>
</thead>
<tbody>
<tr>
<td>NAMES (1d Character array)</td>
<td>Stores the character label of the gate/basic event</td>
</tr>
<tr>
<td>OUTEVENT (1d Integer array)</td>
<td>Stores the integer code for all gates. The integer code corresponds to the gate position in NAMES.</td>
</tr>
<tr>
<td>INEVENT (2d Integer array)</td>
<td>Stores all inputs to a gate. The integer code corresponds to the events position in NAMES.</td>
</tr>
<tr>
<td>GTYPE (1d Integer array)</td>
<td>Stores integer code for gate type. OR=1, AND =2, VOTE = 3</td>
</tr>
</tbody>
</table>

| Table 6.1 Arrays created to store a Fault Tree Structure. |

For the fault tree shown in figure 6.2 the following arrays would be created:

\[
\begin{align*}
\text{NAMES} &= \begin{pmatrix}
\text{TOP} \\
\text{G1} \\
\text{C} \\
\text{A} \\
\text{B}
\end{pmatrix} \\
\text{OUTEVENT} &= \begin{pmatrix}
1 \\
2
\end{pmatrix} \\
\text{INEVENT} &= \begin{pmatrix}
2 & 3 \\
4 & 5
\end{pmatrix} \\
\text{GTYPE} &= \begin{pmatrix}
1 \\
2
\end{pmatrix}
\end{align*}
\]

As detailed in section 2.4 the qualitative analysis of a fault tree involves the identification of the minimal cut sets of the system. The basic events of the fault tree structure are determined by comparison of OUTEVENT and INEVENT. Basic events are only present in INEVENT whereas all gates, apart from the top gate, are stored in both OUTEVENT and INEVENT. Following identification the basic events are stored in BEVENT, which is a 1-dimensional integer array. Based on the MOCUS algorithm given in (ref. 6) an algorithm was created to identify the minimal cut sets of a fault tree. The algorithm functions on the basis that if the fault tree consists of OR gates then the number of minimal cut sets will increase whereas if the structure
contains AND gates: the order of the minimal cut sets will increase. The program sets up a 2-dimensional array CUTSETS and stores the inputs to the top event in the array using the following rules:

1) If the gate is an 'OR' gate then the gate inputs are stored vertically in the array
2) If the gate is an 'AND' gate then the gate inputs are stored horizontally in the array.

The array is then scanned from the top row downwards and any gate encountered is developed using the rules above. Following the expansion of all gates into their respective basic event inputs, the cut sets of a system are represented in CUTSETS. The minimal cut sets are identified by removing all repeated basic events in the same cut set and then by removing non-minimal cut sets i.e. cut sets in which all components do not necessarily all need to fail to cause the top event.

In order to produce a quantitative analysis the probability of failure for each component is required. The probability of failure is entered, as with the fault tree structure, via a data file. The *.aqd file is created by the user and is of the form:

<table>
<thead>
<tr>
<th>NAME</th>
<th>Integer Code</th>
<th>Probability ((Q))</th>
<th>Failure Rate ((\lambda))</th>
<th>Repair Rate ((\mu))</th>
<th>Inspection Interval</th>
<th>Mean time to repair</th>
</tr>
</thead>
<tbody>
<tr>
<td>8 Characters</td>
<td>2 integers</td>
<td>15 double precision</td>
<td>15 double precision</td>
<td>15 double precision</td>
<td>8 double precision</td>
<td>8 double precision</td>
</tr>
</tbody>
</table>

The data file is scanned one line at a time and depending on the integer value in the second column, either the probability of failure is determined and stored in the element PBEV(i), which corresponds to the component BEVENT(i), or the failure and repair rate of the component i is stored in RATES(i,1) and RATES(i,2) respectively. Table 6.2 shows the integer values that can be used in the *.aqd file and the codes corresponding action.

Using the multiplication law from probability theory the unavailability, \(Q_{MCi}\), for minimal cut set i is obtained and stored in PROB(i). The top event probability is then calculated using the minimal cut set upper bound given in equation (2.27).
The qualitative and quantitative procedures can be illustrated using the fault tree shown in figure 6.2. As shown previously the *.ats file is converted into:

\[
\text{TOP} \begin{cases} \text{G1} \\ \text{C} \\ \text{A} \\ \text{B} \end{cases} \quad \text{OUTEVENT} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad \text{INEVENT} = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 5 \end{pmatrix} \quad \text{GTYPE} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \quad \text{BEVENT} = \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix}
\]

The array CUTSETS is identified as:

\[
\text{CUTSETS} = \begin{pmatrix} 3 \\ 4 \\ 5 \end{pmatrix} \quad \text{which is equivalent to} \quad \begin{pmatrix} \text{C} \\ \text{A} \\ \text{B} \end{pmatrix}
\]

The cut sets are of a minimal form, therefore no reduction is required. The *.aqd file is created as:

\[
\begin{align*}
\text{A} & \quad 0 \quad 0.004 \\
\text{B} & \quad 0 \quad 0.0001 \\
\text{C} & \quad 0 \quad 0.005
\end{align*}
\]
Hence,

\[
PBEV = \begin{pmatrix} 0.005 \\ 0.004 \\ 0.0001 \end{pmatrix} \quad \text{and} \quad PROB = \begin{pmatrix} 0.005 \\ 0.0000004 \end{pmatrix}
\]

Finally the top event unavailability is calculated as:

\[
Q_{TOP} = 1 - [(1 - PROB(1)) \cdot (1 - PROB(2))] = 0.005
\]

6.2 Sequential Failures

Having created a program that can manipulate a fault tree structure the first of the two dependent failures was investigated. The area of sequential failures is not novel and a fault tree symbol exists that describes such failures. The Priority-And (PAND) gate, shown in figure 6.3, will result in the output event if the input events occur in the order in which they enter the gate, i.e. left to right. The top event in figure 6.3 will therefore occur if A fails first and B fails next. If B fails first and then A fails, the top event will not occur.

![Figure 6.3 The Priority-And Gate for a Fault Tree](image)

In traditional fault tree analysis this particular gate would be quantified by replacing the PAND gate with a normal AND gate, on the basis that this overestimates the probability. In order to produce an accurate quantification of the PAND gate a Markov model can be used. Following the construction procedure outlined in section 4.3 the state-transition diagram for figure 6.3 can be developed and is depicted in figure 6.4.
The state with the simultaneous failure of A and B can be achieved through two different routes arriving at state 4 (S4) and state 5 (S5). The situation of A failing first, A: F1, and B failing second, B: F2, is represented by state 4. It is this state that is the output event to the PAND gate. For an industrial system it is realistic to assume that when all components have failed or when system failure has occurred, all failed components will be repaired and the system brought back on-line. This situation is not represented by the Markov diagram of the PAND gate illustrated in figure 6.4. In order to include this assumption the repair transition rate, out of state 4, the state representing system failure, and state 5, the state where all components have failed but the top event has not occurred, required derivation. The repair rate for component i, if considered constant, is equivalent to the inverse of the mean time to repair (MTTR) for component i, i.e:

\[ \mu_i = \frac{1}{\text{MTTR}_i} \quad \text{or} \quad \text{MTTR}_i = \frac{1}{\mu_i} \]

Similarly, the repair rate for state i is given by the inverse of the mean time to repair state i. For state 4 and 5 in figure 6.4, the MTTR depends entirely on the type of maintenance team that is assumed to be available for the repair. Given the scenario that only one repair man is available the MTTR for state 4 and state 5 is represented
by the summation of the MTTR for component A and the MTTR for component B, i.e.

$$MTTR_{S4} = MTTR_A + MTTR_B$$

$$= \frac{1}{\mu_A} + \frac{1}{\mu_B}$$

$$= \frac{\mu_A + \mu_B}{\mu_A \mu_B}$$

The repair rate for state 4 and state 5, for a system with 1 repair man, is hence given by the inverse of the MTTR, i.e.

$$\mu_4 = \frac{\mu_A \mu_B}{\mu_A + \mu_B}, \mu_5 = \frac{\mu_A \mu_B}{\mu_A + \mu_B}$$

Given the scenario that two repair men are available the MTTR for state 4 and state 5 is represented by the maximum MTTR of component A or B, i.e.

$$MTTR_4 = \max \{MTTR_A, MTTR_B\}$$

$$= \max \{\frac{1}{\mu_A}, \frac{1}{\mu_B}\}$$

The repair rate for state 4 and 5, for a system with two repair men, is given by the inverse of the MTTR for state 4 and 5 respectively, i.e.

$$\mu_4 = \min \{\mu_A, \mu_B\}, \mu_5 = \min \{\mu_A, \mu_B\}$$

A general formula can be derived to represent the situation where all failed components are repaired in one time period to produce a transition from a state where all components are failed, or a state which represents system failure, to a state where all components are working. For a system with only one repair man, the repair transition from a state representing all components in a failed mode, or a state representing system failure, is given by equation (6.1), where n is equal to the number of components in state i and j corresponds to the failed component in state i. For a system with n repair men, the repair transition from a state representing all
components in a failed mode, or a state representing system failure, is given by equation (6.2).

\[
\mu_i = \frac{\prod_{j=1}^{n} \mu_j}{\sum_{j=1}^{n} \mu_j}
\]

(6.1)

\[
\mu_i = \text{Min}\{\mu_1, \ldots, \mu_n\}
\]

(6.2)

The Markov representation for the PAND gate with 2-input events, A and B, is hence transformed from figure 6.4 to figure 6.5, where \( \mu_4 \) and \( \mu_5 \) are determined using equation (6.1) or (6.2) depending on the type of maintenance team assumed for the system.

6.2.1 Construction of the Markov Diagram for a PAND Gate

Given a PAND gate with n inputs a general procedure had to be developed for the correct construction of the Markov diagram, which could be automated and therefore transparent to the user. Initially an investigation into the total number of states in a
Markov diagram for a PAND gate with \( n \) inputs was completed. Table 6.3 shows the results obtained by investigating 2 to 6 basic event inputs into a PAND gate.

<table>
<thead>
<tr>
<th>Number of Basic Event inputs to PAND gate</th>
<th>Total Number of states in the Markov diagram</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>3</td>
<td>16</td>
</tr>
<tr>
<td>4</td>
<td>65</td>
</tr>
<tr>
<td>5</td>
<td>326</td>
</tr>
<tr>
<td>6</td>
<td>1957</td>
</tr>
</tbody>
</table>

Table 6.3 Number of States in a Markov Diagram for a PAND gate with 2 to 6 input events

An algorithm was required that, given the number of basic event inputs to a PAND gate, could produce the correct state-transition diagram. The development of any Markov diagram is a recursive process of identifying the number of failures that can occur from a newly created state. It was noted, from the 5 manually produced Markov diagrams, that the total number of states could be calculated from knowing the number of inputs to the PAND gate. Starting from the initial state, the working state, only one component failure was considered at a time and the result was the development of the number of new states at the next level. For each new level developed the number of failed components present in any one state increases by one. Therefore for level \( i \) the number of failed components in any state present in level \( i \) is equal to \( i-1 \). A general formula was created, equation 6.3, to determine the number of states in a level due to the states in the previous level of the Markov diagram, where the initial state resides in level 1 of the diagram. The development process stops when all the components in a state are failed, which can be shown to occur at level \( n+1 \).

\[
\text{Number of states} = \left( \text{Number of states at level } i-1 \right) \times \left( \text{Number of working components in any at level } i \right) \times \left( \text{one state in level } i-1 \right) \tag{6.3}
\]
The determination of the number of states, at each level, in the Markov diagram is represented in figure 6.6. The total number of states, \( N_{\text{total}} \), is given by equation (6.4).

\[
N_{\text{total}} = \sum_{i=1}^{n+1} \text{Number of states at level } i \tag{6.4}
\]

To illustrate the process of determining the total number of states the PAND gate shown in figure 6.7 can be utilised.

Figure 6.6 Development of the number of states in a Markov Diagram at each level

Figure 6.7 Example PAND gate with 4 inputs
The number of states at level 1 is equal to 1 as initially all components are assumed to be functioning correctly. There are 4 possible components that can fail, hence the number of states at level 2 is equal to 4, using equation (6.3). Similarly the number of states at level 3, 4, and 5 can be calculated:

\[
\begin{align*}
\text{Number of states at Level 3} & = 4 \times 3 = 12 \\
\text{Number of states at Level 4} & = 12 \times 2 = 24 \\
\text{Number of states at Level 5} & = 24 \times 1 = 24
\end{align*}
\]

Therefore the total number of states, using equation (6.4), can be calculated as:

\[
N_{\text{total}} = 1 + 4 + 12 + 24 + 24 = 65, \text{ as shown in table 6.3.}
\]

The failure sequence of the components in each state is identified using a similar process to that used for the determination of the total number of states in the diagram. An array, STATE, is created which has the dimensions $N_{\text{total}}$ by $N_{\text{basic}}$, where $N_{\text{basic}}$ is equal to the number of basic event inputs to the PAND gate. Each element in $\text{STATE}(i,j)$ is initialised to equal 0, which symbolises that a component is working. Using the initial state the states in level 2 are created by changing, one at a time, each working component found in the initial state into a failed component. The ‘0’ for each component is changed to a ‘1’ to symbolise that the component has failed first and a new state is developed. For the fault tree in figure 6.7 the $\text{STATE}$ array representing level 1 and 2 of the Markov diagram can be shown to be of the form:

\[
\text{STATE} = \begin{pmatrix}
0 & 0 & 0 & 0 \\
1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 \\
0 & 0 & 1 & 0 \\
0 & 0 & 0 & 1
\end{pmatrix}
\]

which is equivalent to:
The number of states, previously identified for level 2, is then used in the
determination of the states in level 3. Each state in level 2 is taken one at a time and
all working components failed, one at a time, to produce the states in level 3. All
working components are changed from a ‘0’ to a ‘2’ to symbolise that the component
has failed second. This process is continued and the n+1th level is created by
changing the working components in the states present in level n from ‘0’ to ‘n’ to
represent that those components have failed last in the sequence.

As STATE is being created the transition matrix for the Markov diagram is also
developed. Whenever a component in any state is changed from a working
component to a failed component a link is created between the two states and the
failure rate of the relevant component is stored in the array TRANS(i,j). The ith row
represents the state where the component was working and jth element represents the
new state where the component is failed. The failure rate for component i is entered
via the *.aqd file and stored in the array RATES(i,1), where the first column indicates
the failure rate. The repair rate of component i is placed in RATES(i,2), where the
second column indicates the repair rate of a component i.

In order to complete the Markov diagram the appropriate repair rates also have to be
entered into TRANS. The repair rates are not determined when developing STATE
but are found after the Markov diagram construction is completed. Each state is taken
in turn and all Nbasic components investigated. For a working state that does not
contain all Nbasic components in a failed mode, if a component is identified as being
failed then it is repaired regardless of what order it failed in. The remaining
components in the same state are then studied as the order of failure may now change
Rule 6.1 was developed to ensure that following the repair of a failed component the order of failure for the remaining basic events was modified correctly:

**Rule 6.1:** For a working state with less than \( N_{basic} \) failed components:

If any component is repaired then all other components, if their order of failure is greater than that of the component being repaired, must be reduced by 1. Alternatively if their order of failure is less than that of the repaired component then their order of failure is unchanged.

For example say the state under inspection had the form:

\[
(1 \ 2 \ 3 \ 0)
\]

If the first component was repaired then a transition would exist, governed by the repair rate of the first component, between the state above and the state:

\[
(0 \ 1 \ 2 \ 0)
\]

For a state which represents system failure or a state containing \( N_{basic} \) failed components the system is brought back on-line by repairing all failed components. Rule 6.2 was developed to ensure that the system was repaired correctly.

**Rule 6.2:** For a failed state or a state with \( N_{basic} \) failed components.

If state \( i \) is a failed state or all components in state \( i \) are failed, then all components are re-set to a working mode, regardless of their order of failure.

For example say the state under inspection had the form:

\[
(1 \ 2 \ 3 \ 4)
\]

As all components have failed each one is repaired and a transition exists, governed by either equation (6.1) or (6.2) depending on the maintenance assumption for the system, between the state above and the initial state: \((0 \ 0 \ 0 \ 0)\)
The code identifies the state that, following the repair of a component, the system would travel into and stores it in the array REPAIR. The array STATE is then searched and the state identical to that stored in REPAIR is identified and the appropriate repair rate is placed in TRANS.

In order to solve the state-transition diagram the diagonal elements in TRANS must be set equal to the negative of the sum of the remaining elements in the corresponding row, as postulated in section 4.5.1.

To illustrate the process of determining the Markov diagram and the state-transition matrix the two-input PAND gate shown in figure 6.3 can be used. As with the independent fault tree analysis outlined in section 6.1.2, the fault tree structure is entered via a *.ats file. The gate type label for a PAND gate in the *.ats file is set equal to PAND and the numerical code used by the program to represent a PAND gate is 4. For figure 6.3 the *.ats file has the following form:

```
TOP PAND 00 2A B
```

The number of states at level 1 is equal to 1 and using equation (6.3) the number of states at level 2 and 3 can be shown to be equal to two respectively. Therefore the array level is equal to:

\[
\text{LEVEL} = \begin{pmatrix}
1 \\
2 \\
2
\end{pmatrix}
\]

and \( \text{Ntotal} \) equals 5.

Initially all elements in the first row of the array STATE are set equal to 0 and using the first state, STATE is created and equals:

\[
\text{STATE} = \begin{pmatrix}
0 & 0 \\
1 & 0 \\
0 & 1 \\
1 & 2 \\
2 & 1
\end{pmatrix}
\]
where component A is represented by the 1st column and B by the second. The transition rate matrix is also filled with the appropriate failure rates:

\[
\begin{pmatrix}
0 & \lambda_A & \lambda_n & 0 & 0 \\
0 & 0 & 0 & \lambda_n & 0 \\
0 & 0 & 0 & 0 & \lambda_A \\
0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

Each row in STATE is then inspected and those that contain failed components are repaired, using rule 6.1 or 6.2, and the appropriate repair rate is stored in TRANS. For example taking the second row in state:

\[(1, 0)\]

It can be seen that the first element represents a failed component. Repair of the failed component leads to the state:

\[(0, 0)\]

which is identical to the state represented by the 1st row of STATE. Therefore the repair rate of component A is stored in TRANS(2,1). This procedure is repeated for all states and the repair rate from state 4 and state 5 is determined using equation (6.2), as 2 repair men are assumed to be available. The transition matrix becomes:

\[
\begin{pmatrix}
0 & \lambda_A & \lambda_n & 0 & 0 \\
\mu_A & 0 & 0 & \lambda_n & 0 \\
\mu_B & 0 & 0 & 0 & \lambda_A \\
\min\{\mu_A, \mu_B\} & 0 & 0 & 0 & 0 \\
\min\{\mu_A, \mu_B\} & 0 & 0 & 0 & 0 \\
\end{pmatrix}
\]

The transition rate matrix is completed by setting all the diagonal elements of the matrix equal to the negative of the sum of the remaining elements in that row. Hence TRANS becomes:
The Markov diagram for this example is shown in figure 6.8.

![Markov Diagram for the STATE array describing a two-input PAND gate](image)

**6.2.2 Quantification of a Markov Diagram for a PAND Gate**

The various techniques that can be utilised to quantify a Markov diagram were discussed in-depth in Chapter 4. In section 4.5.1 it was shown that the linear differential equations could be written in matrix form as:

\[
\begin{bmatrix}
\dot{P}_1(t), \dot{P}_2(t), \ldots, \dot{P}_n(t) \\
\end{bmatrix} = \begin{bmatrix} P_1(t), P_2(t), \ldots, P_n(t) \end{bmatrix} \cdot \begin{bmatrix} A \end{bmatrix}
\]

where \( A \) is the state-transition rate matrix, \( n \) is equal to the total number of states in the Markov diagram and \( \dot{P} \rightarrow 0 \) as \( t \rightarrow \infty \). As outlined in section 4.5.1, there exists several different techniques that can be used to produce a solution for a Markov diagram depending on whether a steady-state solution or transient solution is required. The numerical technique of Gauss elimination was chosen as the tool to be used to analyse the system of linear simultaneous equations, which results when a steady-state
solution is required. Prior to the application of the Gauss elimination procedure the
transition rate matrix requires modification. The matrix must be transposed in order
to represent the differential equations, all of which become simultaneous equations as
\[ \dot{P} \to 0 \text{ as } t \to \infty. \]
In addition to transposing the transition rate matrix a further
equation is required before a solution can be determined, as outlined in section 4.5.2.
The sum of the probabilities of residing in each of the states in the Markov diagram by
definition must equal unity. The additional equation is therefore:

\[ P_1 + P_2 + \ldots + P_n = 1 \]

As outlined in section 4.5.3, several different techniques can be used to produce a
time-dependent solution. Such techniques range from the use of eigenvectors to
numerical analysis methods such as Runge-Kutta. For the purpose of my investigation
a NAG routine was chosen that could solve a system of stiff first-order ordinary
differential equations using a variable-order, variable-step method implementing the
Backward Differentiation Formula (BDF). This method was chosen predominately
for its application to systems which possess a degree of stiffness, i.e. where the repair
and failure rate of components differ greatly in magnitude.

Following the completion of either the Gauss elimination program or the NAG
routine, the array SOLN(i) is created which either contains the steady-state
probabilities of being in state i or the probability of residing in state i at time t. The
final step in the quantification procedure involves the identification of the failed states
in the Markov diagram. For the sequential failure problem the failed state depends
entirely on the order of inputs to the PAND gate. To identify the failed state in the
Markov diagram each state is inspected and the state found which represents the
components failing in the same order as they entered the PAND gate. Following the
identification of the failed state, i, the steady-state probability of system unavailability
or the probability of system failure at time t, is given by SOLN(i).

The steady-state quantification procedure can be illustrated by using the Markov
diagram developed for the two-input PAND gate shown in figure 6.8. Let component
A and B have failure rates $1 \times 10^{-5}$, $2 \times 10^{-4}$ and repair rates 0.06, 0.05 respectively. The transition rate matrix then becomes:

$$
\begin{bmatrix}
-0.00021 & 0.00001 & 0.0002 & 0 & 0 \\
0.06 & -0.0602 & 0 & 0.0002 & 0 \\
0.05 & 0 & -0.05001 & 0 & 0.00001 \\
0.05 & 0 & 0 & -0.05 & 0 \\
0.05 & 0 & 0 & 0 & -0.05
\end{bmatrix}
$$

which is modified to represent the system of linear simultaneous equations used to obtain a steady-state solution:

$$
\begin{bmatrix}
-0.00021 & 0.06 & 0.05 & 0.05 & 0.05 & 0 \\
0.00001 & -0.0602 & 0 & 0 & 0 & 0 \\
0.0002 & 0 & -0.05001 & 0 & 0 & 0 \\
0 & 0.0002 & 0 & -0.05 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 1
\end{bmatrix}
$$

Following the execution of the Gauss elimination program the array SOLN was developed and was equal to:

$$
\begin{bmatrix}
0.99585 \\
1.65 \times 10^{-4} \\
4 \times 10^{-3} \\
6.6 \times 10^{-7} \\
8 \times 10^{-7}
\end{bmatrix}
$$

There exists only one failed state for this example and the code identified that as state 4. The steady-state probability for the two-input PAND gate was therefore equal to:

$$Q_{SYS(\infty)} = SOLN(4) = 6.6 \times 10^{-7}$$

The general procedure for analysing a PAND gate with n inputs is depicted in figure 6.9.
Enter fault tree structure in *.ats file and failure data in *.aqd file

Determine the number of states at each level using Nbasic and Eqn. (6.3)

Determine failure characteristics of each component in each state using LEVEL. Set up TRANS

Complete TRANS by including repair transitions using Rule 6.1 & 6.2. Diagonal terms = (sum of all other terms in row)

Modify matrix ready for analysis. Transpose, add row of 1's, column of 0, for steady-state solution

Solve for steady-state using Gauss Elimination. Produce steady-state unavailability identifying the failed state, i, then Qsys = SOLN(i)

Figure 6.9 Procedure for analysing a PAND gate with n inputs.

6.2.3 Modification to the Sequential Analysis Procedure

By inspection of the Markov diagram developed for a PAND gate with 2 input events it was noted that there existed 5 states, one more state than the state-transition diagram for 2 components where the order of failure is irrelevant. Inspection of table 6.3 illustrates that by considering the order of failure a further increase in the size of the Markov diagram is apparent. The main disadvantage of using the Markov method for system analysis is that the size of the diagram increases rapidly as the number of components grow. This disadvantage is magnified further when the sequence of failure is considered. The Markov technique can be seen to be most efficient when dealing with a small number of dependent components, i.e. 2 basic event inputs to a PAND gate results in the size of the Markov diagram increasing by one state only.

It was therefore decided that an investigation should be completed into the modification of a PAND gate with more than two inputs. Initially an attempt was made to split the PAND gate into smaller fault tree structures which could then be analysed separately. It was decided that components entering a PAND gate would be analysed in pairs. This clause was included so that following the solution of the PAND gate a new ‘super’ event could be easily developed which described the failure
characteristics of the subtree containing the sequential gate. In order to develop a new super event, which would be an input to an dependent gate, the following reliability parameters required derivation:

- The probability of failure for the Top event, $Q_{TE}$
- The failure rate of the Top event, $\lambda_{TE}$
- The repair rate of the Top event, $\mu_{TE}$

Using a two-input PAND gate these parameters can be derived easily. As the 5 state Markov diagram contains only one failed state the repair rate for the system, $\mu_{TE}$, and the probability of failure, $Q_{TE}$, can be obtained directly from the analysis of the diagram. The repair rate for a system, as stated in section 6.2, is equivalent to the inverse of the mean time to repair (MTTR), i.e.

$$\mu_{S} = \frac{1}{MTTR_{S}}$$

For any Markov diagram the MTTR is equal to:

$$MTTR = \frac{-1}{\text{transitions out of failed state}}$$

As the 5 state Markov diagram for a PAND gate, with 2 inputs, contains only one failed state, i.e. state 4 in figure 6.8, the MTTR is given as:

$$MTTR = \frac{-1}{a_{44}}$$

where $a_{44}$ represents the transitions out of the failed state. The repair rate of the system is hence:

$$\mu_{TE} = -a_{44}$$

(6.5)
The probability of system failure for the 5 state Markov diagram is simply the probability of residing in state 4 at time $t$, i.e.

$$Q_{TE}(t) = Q_4(t) \quad (6.6)$$

and for steady state this is equal to:

$$Q_{TE} = Q_4(\infty) \quad (6.7)$$

The remaining parameter, $X_{TE}$, can be obtained from $\mu_{TE}$ and $Q_{TE}$ for a steady state solution. In chapter 2, a steady-state approximation for the system’s unavailability was given as:

$$Q = \frac{\lambda}{\mu + \lambda}$$

Hence an expression for $\lambda_{TE}$ is derived as:

$$\lambda_{TE} = \frac{Q_{TE} \cdot \mu_{TE}}{(1 - Q_{TE})} \quad (6.8)$$

For a transient solution, $\lambda_{TE}$ is obtained via the derivation of the system’s unconditional failure intensity, $w_{TE}$, from the Markov diagram. It can be shown that the unconditional failure intensity for a Markov diagram is obtained by multiplying the probability of being in a working state $i$, $Q_i$, by the transition rate which transports the system into a failed state, $\lambda_j$. Therefore,

$$w_{TE} = \sum_{i=1}^{n} Q_i \prod_{j} \lambda_j \quad (6.9)$$

where $n$ represents all working states that communicate with a failed state, $j = \text{component which fails to cause transition to failed state}$. The failure rate of the top event, $\lambda_{TE}$, is then found using equation (2.12).

A 2-input PAND gate can therefore be expressed as a ‘super’ event using equations (6.5)-(6.9), depending on whether a steady state or transient solution is required. To illustrate the procedure of taking input events to a PAND gate in pairs figure (6.10) can be used.
The fault tree structure is modified in order to produce a PAND gate with 2-input events. Figure 6.10 is changed to the form:

![Figure 6.10 PAND gate with three input events](image)

For figure 6.11 the PAND gate with inputs A and B is solved first. Following the steady-state solution of the 5 state Markov diagram representing the PAND gate with inputs A and B a new event, $A^*$, is developed with:

$$Q_{A^*} = Q_4(\infty), \quad \mu_{A^*} = -\alpha_{44}, \quad \text{and} \quad \lambda_{A^*} = \frac{Q_{A^*} \cdot \mu_{A^*}}{(1 - Q_{A^*})}$$

and figure 6.11 is reduced to the form depicted in figure 6.12.

![Figure 6.11 Conversion of a three-input PAND gate to produce a 2-input PAND gate](image)
The top event probability is then obtained by the analysis of another 5 state Markov diagram which represents the PAND gate with inputs A* and C. Figure 6.10 is solved via the solution of two 5 state Markov diagrams. If the fault tree structure had been solved as one module then a 16 state Markov diagram would have required analysis. The saving in computation for such a small example is minimal, however for larger systems a saving would be apparent. It can be shown that the solution of the two, 5 state Markov diagram is identical to the answer produced via the solution of the single 16 state Markov diagram. This is, however, only applicable if the events in the pairings do not exist elsewhere in the tree structure.

The code was updated to incorporate the modifications outlined above. Following the identification of a PAND gate the left-most pair is analysed first and the super event label placed back in the fault tree structure. This process is then continued until there exists only 2 basic event inputs into the PAND gate.

6.3 Combination of Independent and Dependent Failures

The analysis of an industrial system may require consideration of a mixture of independent and dependent failure events. Many systems comprise of a combination of the two, however the proportion of the system which contains dependent failures is usually small. As only a fraction of the fault tree is usually dynamic transforming the entire tree into a Markov diagram in order to derive an accurate result can been shown to be inefficient. However by utilising the results highlighted in section 6.3 any PAND gate can be solved using Markov analysis and be replaced in the fault tree structure as a super event. Providing that the inputs to the PAND gate do not occur
elsewhere in the tree structure, the submodule containing the dependent failures can be solved separately to the remainder of the tree.

The general procedure for analysing a fault tree structure containing both independent and dependent parts includes finding independent subtrees (ref. 47).

6.3.1 Identification of Independent Subtrees

The term 'Modularization' is used to describe the process of identifying independent subtrees. When incorporating modularization for research or software development purposes many authors have implemented Dutuit and Rauzy's linear-time algorithm (ref. 48), which has been shown to be both efficient and simple to implement. The basic principle of the algorithm can be stated as follows:

"Let \( v \) be an internal event, (a gate), and \( t_1 \) and \( t_2 \) respectively the first and second dates of visits of \( v \) in a depth-first left-most traversal of the fault tree. Then \( v \) is a module iff none of its descendants is visited before \( t_1 \) and after \( t_2 \) during the traversal”

By definition the root event (TOP) and terminal events (Basic Events) are always modules. The algorithm’s main features include two depth-first left-most (DFLM) traversals:

The first traversal is to identify the first, second and last time an internal event is visited and the relevant counters are set according to the following rules: In the first visit to a node the counter first_date is set, and in the second visit second_date is set. Further visits to the node increment the counter last_date. Additionally if the counter to a node is set then the inputs to the node are not visited and for basic events first_date and second_date are identical

The second traversal is used to identify the minimum of the first date and maximum of the last visiting date for each gate's inputs. For each gate node all inputs to the gate are traversed and min_date and max_date are set.
Following the two DFLM traversals each gate node holds five pieces of information and each basic event three. The gate ‘v’ is then concluded to be a module iff a) the collected minimum is greater than the first date of v and b) the collected maximum is less than the second date of v.

The fault tree depicted in figure 6.13 can be used to illustrate the modularization process described in the above section. Table 6.4 shows the count of the first traversal, the results of which are depicted in table 6.5. Table 6.6 shows the results of the second DFLM traversal. Inspection of table 6.6 highlights that the linear time algorithm has determined that the independent submodules are: {TOP, G1, G5, G7}. The gate G2 is not independent because the collected maximum (15) is not less than the second visit date for G2 (12). G3 is deemed dependent as the collected minimum (5) is not greater than the first visit date for G3 (13). Similarly G4 and G6 are identified as dependent submodules.

<table>
<thead>
<tr>
<th>Step 1</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>'v'</td>
<td>TOP</td>
<td>G1</td>
<td>G2</td>
<td>G4</td>
<td>D</td>
<td>C</td>
<td>G4</td>
<td>G5</td>
<td>B</td>
<td>H</td>
</tr>
<tr>
<td>Step 11</td>
<td>12</td>
<td>13</td>
<td>14</td>
<td>15</td>
<td>16</td>
<td>17</td>
<td>18</td>
<td>19</td>
<td>20</td>
<td></td>
</tr>
<tr>
<td>'v'</td>
<td>G5</td>
<td>G2</td>
<td>G3</td>
<td>G6</td>
<td>D</td>
<td>E</td>
<td>G6</td>
<td>G7</td>
<td>F</td>
<td>G</td>
</tr>
<tr>
<td>Step 21</td>
<td>22</td>
<td>23</td>
<td>24</td>
<td>25</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>'v'</td>
<td>G7</td>
<td>G3</td>
<td>G1</td>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 6.4 Counter for first DFLM Traversal

<table>
<thead>
<tr>
<th>top</th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
<th>G6</th>
<th>G7</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>F</th>
<th>G</th>
<th>H</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>13</td>
<td>4</td>
<td>8</td>
<td>14</td>
<td>18</td>
<td>24</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>16</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>2nd</td>
<td>25</td>
<td>23</td>
<td>12</td>
<td>22</td>
<td>7</td>
<td>11</td>
<td>17</td>
<td>21</td>
<td>24</td>
<td>9</td>
<td>6</td>
<td>5</td>
<td>16</td>
<td>19</td>
<td>20</td>
</tr>
<tr>
<td>last</td>
<td>25</td>
<td>23</td>
<td>12</td>
<td>22</td>
<td>7</td>
<td>11</td>
<td>17</td>
<td>21</td>
<td>24</td>
<td>9</td>
<td>6</td>
<td>15</td>
<td>16</td>
<td>19</td>
<td>20</td>
</tr>
</tbody>
</table>

Table 6.5 Results for figure 6.13 following first DFLM traversal

<table>
<thead>
<tr>
<th>Name</th>
<th>top</th>
<th>G1</th>
<th>G2</th>
<th>G3</th>
<th>G4</th>
<th>G5</th>
<th>G6</th>
<th>G7</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>5</td>
<td>9</td>
<td>5</td>
<td>19</td>
</tr>
<tr>
<td>max.</td>
<td>24</td>
<td>22</td>
<td>15</td>
<td>20</td>
<td>15</td>
<td>10</td>
<td>16</td>
<td>20</td>
</tr>
<tr>
<td>module?</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>NO</td>
<td>YES</td>
</tr>
</tbody>
</table>

Table 6.6 Results for figure 6.13 following second DFLM traversal
This example illustrates the efficiency of the algorithm to highlight independent submodules.

The highlighted submodules are then solved by using either FTA or Markov methods, depending on whether they contain independent or dependent failure events. Once the individual subtrees have been solved, the subtree is replaced by a 'super' event which contains the appropriate reliability parameters. As the fault tree structure can contain both independent and dependent parts, the 'super' events must contain the required reliability parameters for each part. Therefore for each super event the following parameters must be obtained:

- The probability of failure for the Top event, $Q_{TE}$
- The unconditional failure intensity for the Top event, $w_{TE}$
- The failure rate of the Top event, $\lambda_{TE}$
- The repair rate of the Top event, $\mu_{TE}$
Following the solution of a PAND gate equations (6.5-6.9) are utilised to calculate $Q_{TE}$, $\mu_{TE}$, $w_{TE}$ and $\lambda_{TE}$. The relationship between the frequency, rate and probability of failure was outlined in chapter 2 as:

\[ w_i = \lambda_i(1 - Q_i) \]

Hence given the steady-state probability of failure for the top event, equation (6.7), and the failure rate of the top event, equation (6.8), the steady-state unconditional failure intensity can be obtained via equation (6.10).

\[ w_{TE} = \lambda_{TE}(1 - Q_{TE}) \quad (6.10) \]

Following the solution of a subtree containing independent failures the probability of failure, $Q_{TE}$, is obtained using the MOCUS algorithm and the minimal cut set upper bound. In order to obtain an expression for the failure and repair rate of the Top event, the unconditional failure intensity of the top event is required, $w_{TE}$. As stated in section 2.6.5 the unconditional failure intensity for a system can be obtained via:

\[ w_{TE} = \sum_{i=1}^{n} G_i(q(t)) \cdot w_i(t) \quad (6.11) \]

where $n$ is the total number of components in the system and $G_i(q(t))$ is Birnbaum's criticality function.

Once $w_{TE}$ and $Q_{TE}$ have been obtained $\lambda_{TE}$ can be calculated using equation (2.12). In order to obtain an expression for the repair rate of the top event, $\mu_{TE}$, the unconditional repair intensity requires derivation. The unconditional repair intensity, $\nu_{TE}$, is the probability that the system is failed at time $t$ and is repaired in the small time interval $dt$. A formula had to be developed in order to obtain $\nu_{TE}$ from a fault tree structure. Birnbaum's criticality function was used in the development as: for component $i$ to cause system failure the system must be in a critical state for component $i$ and component $i$ must fail. i.e:

\[ G_i(Q(t)) \cdot Q_i(t) \]

This expression represents the probability of system failure due to component $i$, therefore the repair of the system due to component $i$ is given by:

\[ G_i(Q(t)) \cdot Q_i(t) \cdot \nu_i(t) \quad (6.12) \]

where $\nu_i(t)$ is obtained using equation (2.15).
From equation (6.12) a formula can be devised which calculates the unconditional repair intensity of a system with \( n \) events, \( \nu_{TE}(t) \):

\[
\nu_{TE} = \sum_{i=1}^{n} G_i(q(t)) \cdot q_i(t) \cdot \nu_i(t)
\]  

(6.13)

Therefore for a system containing 2 basic events, A and B, represented by the minimal cut sets \( \{A\}, \{B\} \), the unconditional repair intensity of the system due to component A is:

\[
\nu_{TE/A} = (1-Q_B)Q_A \cdot \nu_A
\]

and the overall system unconditional repair intensity is:

\[
\nu_{TE} = (1-Q_B)Q_A \cdot \nu_A + (1-Q_A)Q_B \cdot \nu_B
\]

Having obtained an expression for \( \nu_{TE} \) the repair rate for the top event can be calculated using equation (2.15).

To illustrate the modularization and analysis process for a fault tree containing both independent and dependent structures figure 6.14 can be utilised.

![Fault Tree Diagram](image)

Figure 6.14 Fault Tree containing Independent and Dependent Failure Events
The results of the modularization process for figure 6.14 are depicted in table 6.7 and 6.8.

<table>
<thead>
<tr>
<th>Table 6.7 Results for figure 6.14 following first DFLM traversal</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
</tr>
<tr>
<td>min</td>
</tr>
<tr>
<td>max.</td>
</tr>
<tr>
<td>module</td>
</tr>
</tbody>
</table>

By inspection of table 6.8 it can be seen that not all of the gates are independent subtrees. Gate 1, G1, is not an independent subtree as the collected maximum (14) is not less than the second visiting date (5) for G1. Also Gate 2, G2, is not an independent subtree as the collected minimum (4) is not greater than the first visiting date (6) for G2. The remaining gates are independent and G3 and G4 can be solved separately, prior to the solution of the top gate. By scanning the fault tree structure it can be noted that G3 is a PAND gate and hence dynamic in nature. The inputs to the gate are not basic events and as G4 is an independent subtree it is solved first, using FTA. The new basic event, D*, is created which has the reliability parameters:

\[
Q_{D^*} = Q_D - Q_E \\
\lambda_{D^*} = \frac{w_{D^*}}{1 - Q_{D^*}} \\
\mu_{D^*} = \frac{\lambda_{D^*}}{Q_{D^*}} \\
w_{D^*} = Q_D \cdot w_D \cdot Q_D \cdot w_E \\
v_{D^*} = Q_D \cdot Q_E \cdot v_D + Q_D \cdot Q_E \cdot v_E \\
V_{D^*} = Q_D \cdot Q_E \cdot v_D + Q_D \cdot Q_E \cdot v_E + Q_D \cdot Q_E \cdot V_E
\]

G3 now has 2 basic event inputs and can be converted into a 5 state Markov diagram and solved to produce the new basic event, F*, whose reliability parameters are determined using equations (6.5 - 6.9). Following the solution of the PAND gate the structure of the fault tree contains no further independent subtrees or dependencies and the entire fault tree can be solved using the MOCUS algorithm. The fault tree given in figure 6.14 was therefore solved via the solution of two static fault trees and one 5 state Markov diagram. This process is noticeably more efficient than converting
the entire structure into a Markov diagram, which with 6 basic events occurring sequentially, would have had 1957 states.

6.4 Summary
The algorithm depicted in figure 6.15 illustrates the process which was developed in order to solve a fault tree structure that could comprise of traditional fault tree gate symbols as well as the PAND gate.

Figure 6.15 Algorithm used to solve Fault Trees containing Sequential and Independent Failure Events.
A general procedure has been created which has the capability of modelling independent and sequential failures on a fault tree structure. Following the development of the fault tree diagram for a system the tree undergoes a modularization process. The identification method traces though the original fault tree, depth-first left-most, starting from the top event. Each gate is then checked to determine whether or not it is the largest possible independent subtree, which will be either independent or dependent in nature. Once these subtrees are identified they are solved using the appropriate technique and their solution integrated back into the tree. Several Markov and fault tree models may need to be solved to give the final result.

The advantage of this combined model is that not only can dependent failures be modelled efficiently and accurately, the failure logic of the system is retained due to the system being represented on a fault tree diagram. The process created, however, is not without faults. The main disadvantage of the new combined model is that if an event is present in both independent and dependent parts of the tree then modularization is limited. In addition to this the ‘pair analysis’ procedure used to solve a PAND gate with more than 2 inputs is also infeasible if any of the inputs to the PAND gate are repeated elsewhere in the tree structure. The result of this is that a larger portion of the fault tree would have to be converted into a Markov diagram, if an accurate answer is to be obtained. Conversion of large parts of the fault tree into a Markov diagram, as shown in this chapter, result in a large Markov diagram which can be computationally expensive to solve. However this is a theoretical limitation as it has not, as yet, been encountered in any real system.

6.5 Standby Failures

The first dependency studied was that of sequential failures and a new assessment procedure was developed for solving a fault tree structure containing PAND gates accurately. The second type of dependency that was investigated was that of standby failures. A standby component is included to increase the levels of redundancy within a system in an attempt to increase the reliability of the system. A primary component may be initiated given a demand or be continuously operating. In the event of the failure of the primary component, the standby component is brought on-line. In reliability problems three main types of standby components can be encountered:
Cold Standby:
The standby component is classified as 'cold' if it is considered perfectly reliable when in standby i.e. has a zero failure rate. When a cold standby component is brought on-line it operates with a non-zero failure rate and therefore the failure rate of the standby component is dependent on the failure of the primary component.

Warm Standby:
The standby component is classified as 'warm' when it has a lower failure rate when in standby than when on-line. The failure rate of the standby component is therefore dependent on the functionality of the primary component.

Hot Standby:
The standby component is classified as 'hot' when the failure rate of the component is identical regardless of whether the component is in standby or on-line. No conditional dependency therefore exists between the primary component and the standby component. For such standby components FTA can be used as the analysis tool.

The warm standby component is possibly the most realistic and for the remainder of this chapter the warm standby case will be discussed in detail. The cold standby case will be outlined and the differences between the two cases highlighted.

6.5.1 The New Standby Gate
Analysis of a system containing standby failure events also required a modelling technique that could handle dependencies. As with the sequential failure events, discussed in section 6.2, it was decided that the standby components should be represented on the fault tree structure yet analysed using Markov methods. To include the standby components in the fault tree structure a new gate had to be created. In order to determine failure of a system containing n components, each of which are either on-line or in standby, the number of components, k, that are required to function in order for the system to function must be known. The system will therefore function as long as k/n components are operating. The fault tree structure describes the failure logic of a system and for this reason the standby gate must also represent failure. For any system that requires k/n components to be functioning, the standby gate must
indicate that \( n-k+1 \) components are required to fail to cause system failure. This observation led to the development of the warm standby gate illustrated in figure 6.16.

![Figure 6.16 The New Warm Standby Gate for a Fault Tree Structure](image)

The standby gate has features similar to the normal voting gate and, as with the PAND gate, the inputs to the standby gate are brought in, in a left to right ordering. Therefore in a system containing 2 components, one of which is required to be functioning correctly, the left-most component input would be on-line with the remaining components assumed to be in standby. The initial state of the Markov diagram for such a system would be:

![Markov Diagram](image)

The Markov diagram is constructed by identifying all possible transitions out of all newly created states. Following the identification of the possible transitions out of state one (S1), i.e. failure of the working and standby components, the Markov diagram becomes:

![Markov Diagram](image)

where \( \lambda_{BS} \) represents the failure rate of component B in standby.
State two (S2) and three (S3) are then investigated and their possible transitions identified. For S2 if A is repaired it is assumed that A is placed into standby as B is working on-line. Similarly the repair of component B from S3 is placed back into standby as A is working on-line. All possible transitions out of newly developed states are identified and the Markov diagram for figure 6.16, with k=1, can be seen in figure 6.17. The Markov diagram includes the assumption that repair from a failed state, or a state where all components have failed, leads to the initial working state with the transition rate $\mu_5$ which is determined using equation (6.1) or (6.2), depending on the maintenance team assumed for the system.

![Markov Diagram for a Standby Gate with 2 components, where k=1.](image)

The state with the simultaneous failure of A and B is the output event to the standby gate, this can be seen as state 5 (S5) in figure 6.17.

Similarly a cold standby gate was developed, the structure of which is shown in figure 6.18.
The Markov diagram for a cold standby system with 2 components is developed in an identical manner to that of the Markov diagram for a warm standby system and is shown in figure 6.19. The only difference between the Markov diagram for a warm standby gate and a cold standby gate is that in a cold standby system the standby components cannot fail until they have been brought on-line, i.e. no transition exists between state 1 and 3, and state 4 and 2.

6.5.2 Construction of the Markov Diagram for a System with Standby Components

Prior to the inclusion of the standby gate into a fault tree representation of a system, an investigation into the development of the correct Markov diagram for a system
containing standby failures was initiated. As stated in section 1.5.2.1 the number of states, \((N_{\text{state}})\), in a Markov diagram for a system containing components that can either be working or failed, is equal to:

\[
N_{\text{state}} = 2^n
\]

where \(n\) is equal to the number of basic events in the system.

For a system containing components that can either be working, failed or in standby this formula cannot be used to determine \(N_{\text{state}}\). A new formula, therefore, had to be developed. In order to determine the form of the new formula an investigation into the size and pattern of the Markov diagram was completed for a standby gate with 3, 4 and 5 component inputs. Table 6.9 shows the results obtained from this investigation, where any state in level \(i\) contains \(i-1\) failed components.

<table>
<thead>
<tr>
<th>System: (k/n)</th>
<th>STATES ON LEVEL 1</th>
<th>STATES ON LEVEL 2</th>
<th>STATES ON LEVEL 3</th>
<th>STATES ON LEVEL 4</th>
<th>STATES ON LEVEL 5</th>
<th>STATES ON LEVEL 6</th>
<th>NSTATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>1/3</td>
<td>3</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td>13</td>
</tr>
<tr>
<td>2/3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>1</td>
<td></td>
<td></td>
<td>10</td>
</tr>
<tr>
<td>1/4</td>
<td>4</td>
<td>12</td>
<td>12</td>
<td>4</td>
<td>1</td>
<td></td>
<td>33</td>
</tr>
<tr>
<td>2/4</td>
<td>6</td>
<td>12</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td></td>
<td>29</td>
</tr>
<tr>
<td>3/4</td>
<td>4</td>
<td>4</td>
<td>6</td>
<td>4</td>
<td>1</td>
<td></td>
<td>19</td>
</tr>
<tr>
<td>1/5</td>
<td>5</td>
<td>20</td>
<td>30</td>
<td>20</td>
<td>5</td>
<td>1</td>
<td>81</td>
</tr>
<tr>
<td>2/5</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>1</td>
<td>86</td>
</tr>
<tr>
<td>3/5</td>
<td>10</td>
<td>20</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>1</td>
<td>56</td>
</tr>
<tr>
<td>4/5</td>
<td>5</td>
<td>5</td>
<td>10</td>
<td>10</td>
<td>5</td>
<td>1</td>
<td>36</td>
</tr>
</tbody>
</table>

Table 6.9 Results from investigation into Markov Diagrams for Standby Systems

An algorithm was required that given the number of basic event inputs to the standby gate, \(n\), and the number of components that need to function in order to work the system, \(k\), could produce the correct state-transition diagram. As with the sequential gate it was discovered that the number of states at each level of the Markov diagram could be identified. Instead of manipulating the number of components that could fail in any one state, as with the PAND gate, permutations were used to identify the
number of states in each level of the diagram. A permutation is an arrangement of n events in a certain order. For a system with components that can either be working, failed or in standby, there exists three different group types. By splitting the component status into different groups a formula can be used to calculate the number of states in any given level due to the number of components in each group. When n things can be divided into c classes such that things belonging to the same class are alike, while things belonging to different classes are different, then the number of permutations of these things taken all at a time is:

\[
\frac{n!}{n_1! \cdot n_2! \cdot n_3! \cdot \ldots \cdot n_c!} \quad (n_1 + n_2 + n_3 + \ldots + n_c = 1)
\]

where \( n_j \) is the number of things in the \( j^{th} \) class.

A general formula was created based on the permutation equation to determine the number of states in each level for the new standby gate.

Number of states at level \( i \) = \[
\frac{n!}{n_1! \cdot n_2! \cdot n_3!}
\] \hspace{1cm} (6.14)

\( n_1 = \) number of working components in a state in level \( i \)
\( n_2 = \) number of failed components in a state in level \( i \)
\( n_3 = \) number of standby components in a state in level \( i \)

The determination of the total number of states, \( N_{total} \), is given, as with the PAND gate, by equation (6.4).

To illustrate the process of determining the number of states at each level and the total number of states in a Markov diagram for a warm standby gate, figure 6.20 can be used.
For this system \( k=2 \) and \( n=3 \), therefore the system fails if \( 2/3 \) inputs are in a failed state. The number of states at level 1 can be found using equation (6.14) and the fact that the class of working components contains two components, i.e. \( n_1 = 2 \), the class of failed components contains no entries, i.e. \( n_2 = 0 \) and the standby class contains 1 component, i.e. \( n_3 = 1 \). Hence the number of states in level 1 of the Markov diagram for figure 6.20 is equal to:

\[
\text{Number of states in level 1} = \frac{3!}{2!0!1!} = 3
\]

For the number of states in level 2 it must be noted that a working component has failed from level 1 and therefore a standby component is brought on-line. Hence,

\[
\begin{align*}
    n_1 &= 2 \\
    n_2 &= 1 \\
    n_3 &= 0
\end{align*}
\]

and

\[
\text{Number of states in level 2} = \frac{3!}{2!1!0!} = 3
\]

Similarly the number of states in level 3 and 4 can be shown to be equal to:

\[
\begin{align*}
    \text{Number of states in level 3} &= 3 \\
    \text{Number of states in level 4} &= 1
\end{align*}
\]

\( N_{\text{total}} \) is equal to 10 and all the data produced can be seen to be identical to that given in table 6.9.
The failure characteristics of the components in each state are identified using an identical process to that used for the PAND gate. Initially STATE is created with all N\text{total} by N\text{basic} elements set equal to ‘-1’. The label ‘-1’ is used to indicate a component that is in standby. The different combinations of k from n are then identified and the states in level 1 created. For example take a warm standby system containing 3 components, one of which is required to function in order for the system to work. In this system k = 1 and the initial states in level 1 are created by identifying the different combinations of 1 working component from 3 standby components. i.e.

\[
\begin{align*}
(A:W, B:S, C:S) &= (0 \ -1 \ -1) \\
(A:S, B:W, C:S) &= (-1 \ 0 \ -1) \\
(A:S, B:S, C:W) &= (-1 \ -1 \ 0)
\end{align*}
\]

Using the initial states, the states in level 2 are created by changing, one at a time, each working or standby component found in an initial state into a failed component. The ‘0’ for each working component is changed to a ‘1’ to symbolise component failure. The remainder of the components are then examined in an attempt to identify a standby component. It is assumed that the first identified standby component will be brought on-line and hence the ‘-1’ is changed to a ‘0’ and a new state is developed. For the warm standby case, in the event of the failure of a standby component the ‘-1’ is changed to a ‘1’ and a new state is developed. For example taking the 3 initial states that were created above for the 3 component warm standby system, with k = 1, the states in level 2 are created as follows:

\[
\begin{align*}
(A:W, B:S, C:S) &= (0 \ -1 \ -1), A \ fails &= (A:F, B:W, C:S) = (1 \ 0 \ -1) \\
(A:W, B:S, C:S) &= (0 \ -1 \ -1), B \ fails &= (A:W, B:F, C:S) = (0 \ 1 \ -1) \\
(A:W, B:S, C:S) &= (0 \ -1 \ -1), C \ fails &= (A:W, B:S, C:F) = (0 \ -1 \ 1) \\
(A:S, B:W, C:S) &= (-1 \ 0 \ -1), A \ fails &= (A:F, B:W, C:S) = (1 \ 0 \ -1) \\
(A:S, B:W, C:S) &= (-1 \ 0 \ -1), B \ fails &= (A:W, B:F, C:S) = (0 \ 1 \ -1) \\
(A:S, B:W, C:S) &= (-1 \ 0 \ -1), C \ fails &= (A:S, B:W, C:F) = (1 \ 0 \ -1) \\
(A:S, B:S, C:W) &= (-1, -1, 0), A \ Fails &= (A:F, B:S, C:W) = (1 \ -1 \ 0) \\
(A:S, B:S, C:W) &= (-1, -1, 0), B \ Fails &= (A:S, B:F, C:W) = (-1 \ 1 \ 0) \\
(A:S, B:S, C:W) &= (-1, -1, 0), C \ Fails &= (A:W, B:S, C:F) = (0 \ -1 \ 1)
\end{align*}
\]
In the instance that a new state developed is identical to a state previously created, the new state is not included in the Markov diagram as it already exists. For the fault tree in figure 6.20 the STATE array representing level 1 and 2 of the Markov diagram would be of the form:

\[
\begin{pmatrix}
0 & 0 & -1 \\
0 & -1 & 0 \\
-1 & 0 & 0 \\
1 & 0 & 0 \\
0 & 1 & 0 \\
0 & 0 & 1
\end{pmatrix}
\]

In order to determine the failure characteristics of the components in the states in level 3, the states previously identified for level 2 are used. Each state in level 2 is taken, one at a time, and all working components failed, one at a time, and any available standby components brought on-line to produce the states in level 3. The process of failing a working component and activating a standby component, or failing a standby component, is continued until the (n+1)th level where all components will be in a failed state.

As STATE is being created the links between appropriate states are also set in TRANS. Whenever a component in a particular state is changed from a working or standby component to a failed component a link is created between the two states and the failure rate of the component stored in TRANS(i,j). As with the sequential problem the failure and repair data is entered via the *.aqd file. The failure rate for component i, which is on-line, is stored in RATES(i,1) and the failure rate for component j, which is in standby, is stored in RATES(j,3). In addition to the identification of the failure links between states, the repair paths are also determined as the Markov diagram is developed. The following rules were devised to ensure that the correct repair links were identified and the appropriate rates stored in TRANS.

1) **For a warm standby gate only**: If component c is a standby component which fails then communication between the state where component c was in standby, i, and
the state where component c is failed, j, is two-way, i.e.

\[
\lambda_{cs} \quad \mu_c
\]

\[\text{TRANS}(i,j) = \text{RATES}(c,3) \]
\[\text{TRANS}(j,i) = \text{RATES}(c,2)\]

2) **For both warm and cold standby gates:** If component c is a working component which fails then communication between the state where component c was working, i, and the state where component c is failed, j, is dependent on whether state i contains any standby components:

   i) If state i contains standby components then the communication between state i and j will be one-way, i.e.

\[
\lambda_c
\]

\[\text{TRANS}(i,j) = \text{RATES}(c,1)\]

The repair transition from state j is determined by comparing state j with all other existing states and identifying state m where \(\text{STATE}(m,i) = \text{STATE}(j,i)\) for all components except c and \(\text{STATE}(m,c) - \text{STATE}(j,c) = -2\) i.e.

\[
\lambda_c
\]

\[\text{TRANS}(i,j) = \text{RATES}(c,1) \]
\[\text{TRANS}(j,m) = \text{RATES}(c,2)\]
For example say that state $i$ was equal to $(A: W, B: S, C: S)$, i.e.

$$state\ i = (0\ -1\ -1)$$

and the component in the first column, $A$, was failed to produce state $j$ where

$(A: F, B: W, C: S)$:

$$state\ j = (1\ 0\ -1)$$

Repair of component $A$ would now lead to the state where $(A: S, B: W, C: S)$ i.e.

$$state\ m = (-1\ 0\ -1)$$

which is not equal to state $i$ but equal to state $m$, where $STATE(m,A)-STATE(j,A) = -2$.

ii) If state $i$ does not contain any standby components then the communication between state $i$ and $j$ is two-way, i.e. $TRANS(i,j) = RATES(c,1)$, $TRANS(j,i) = RATES(c,2)$ if state $j$ is not a failed state or does not contain $n$ failed components.

iii) If state $i$ does not contain any standby components, and state $j$ is a failed state or contains $n$ failed components then the communication between state $i$ and $j$ is one-way, i.e $TRANS(i,j) = RATES(i,1)$. A repair transition exists from state $j$ to state $m$, where state $m$ is the initial working state, with the rate $\mu_j$ determined by equation (6.1) or (6.2).

To illustrate the process of determining the Markov diagram and the state transition matrix, the 2-input warm standby gate shown in figure 6.21 can be utilised.

![Figure 6.21 2-Input Warm Standby Gate, k=1](image-url)
As with the PAND gate, outlined in section 6.2, the fault tree structure is entered via a *
.ats file. The gate type labels for a warm and cold standby gate are set equal to
WSBY and CSBY and the numerical code used by the program to indicate a warm or
cold standby gate is 5 and 6 respectively. In addition to this the number of failed
components, n-k+1, that are required to fail to cause the standby gate occurrence is
entered in the KVG column of the data file, with the number of inputs, n, entered in
the NVG column. For figure 6.21 the *.ats file has the form:

\[
\text{TOP WSBY2 20 2A B}
\]

The number of states at level 1 is equal to 2, using equation (6.14) and the number of
states at level 2 and 3 can be shown to equal 2 and 1 respectively. Therefore:

\[
\begin{bmatrix}
2 \\
2 \\
1 \\
\end{bmatrix}
\]

and \( N_{\text{total}} = 5 \).

Initially all elements in \( \text{STATE} \) are set equal to -1. The first states are created by
determining the different combinations of \( k \) working components from \( n \) standby
component, hence the two initial states are:

\[
(A: \text{W}, B: \text{S}) = (0, -1) \\
(A: \text{S}, B: \text{W}) = (-1, 0)
\]

Using the newly developed states in each level \( \text{STATE} \) is created and equals:

\[
\begin{bmatrix}
0 & -1 \\
-1 & 0 \\
1 & 0 \\
0 & 1 \\
1 & 1 \\
\end{bmatrix}
\]

The transition-rate matrix is created simultaneously with the development of \( \text{STATE} \).
For example the state (1 0) is developed from state (0 -1) due to the failure of
component A and therefore \( \text{TRANS}(1,3) \) is set equal to \( \lambda_A \). The state (-1 0) is
identified as the state which would be attained by the system following the repair of A
and \( \text{TRANS}(3,2) \) is set equal to \( \mu_A \). The failed state for this system can be identified
as state 5, hence a repair transition exists from state 5 to state 1, the initial working state. The complete transition rate matrix is:

\[
\text{TRANS} = \begin{pmatrix}
0 & 0 & \lambda_A & \lambda_{RS} & 0 \\
0 & 0 & \lambda_{AS} & \lambda_h & 0 \\
0 & \mu_A & 0 & 0 & \lambda_{RS} \\
\mu_h & 0 & 0 & 0 & \lambda_A \\
\mu_s & 0 & 0 & 0 & 0
\end{pmatrix}
\]

The transition rate matrix is completed by setting all the diagonal elements in the matrix equal to the negative of the sum of the remaining elements in that row. Hence,

\[
\text{TRANS} = \begin{pmatrix}
-(\lambda_A + \lambda_{RS}) & 0 & \lambda_A & \lambda_{RS} & 0 \\
0 & -(\lambda_{AS} + \lambda_h) & \lambda_{AS} & \lambda_h & 0 \\
0 & \mu_A & -(\mu_A + \lambda_h) & 0 & \lambda_h \\
\mu_h & 0 & 0 & -(\lambda_A + \mu_h) & \lambda_A \\
\mu_s & 0 & 0 & 0 & -(\mu_s)
\end{pmatrix}
\]

The Markov diagram for this example can be shown to be identical to that given in figure 6.17, with the probability of system failure equal to the probability of residing in state 5.

6.5.3 Quantification of the Markov Diagram for a System with Standby Components

Steady-state quantification of the Markov diagram representing the new standby gates is achieved as described in section 6.2.2. The array SOLN is produced following the completion of the Gauss elimination routine and each state in the Markov diagram is examined. A failed state is identified as any state which contains less than k working components. Once identified as a failed state the probability of residing in that state is stored in FAILEDSTAT. Once all states have been investigated the steady-state top event probability is equal to FAILEDSTAT.
Having developed an algorithm that could produce the correct Markov diagram and a steady-state solution for the new standby gate an industrial system, which contained standby components and required a transient solution, was investigated.

6.6 Application of the Combined Model to an Industrial System with Standby Failures

The system used to illustrate the new warm standby gate was part of a deluge system found on an offshore platform, the configuration of which is shown in figure 6.22. The system consists of four individual pump streams where two pump streams are powered by an electric source and two are powered by diesel. This type of system is used in the event of a fire on the platform and is therefore dormant until a demand is made on the system. The functionality of each component and their respective failure modes and system consequence is given in table 6.10.

![Figure 6.22 Deluge Pump System](image-url)
<table>
<thead>
<tr>
<th>Component</th>
<th>Function</th>
<th>Failure Mode</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Valve</td>
<td>To test flow of water</td>
<td>Failed open: Decreases amount of water delivered to ringmain</td>
</tr>
<tr>
<td>Pressure Relief Valve</td>
<td>To relieve overpressure in pipeline</td>
<td>Failed open: Decreases amount of water delivered to ringmain</td>
</tr>
<tr>
<td>Isolation Valve</td>
<td>To isolate value for maintenance purpose</td>
<td>Failed Closed: No water delivered to ringmain</td>
</tr>
<tr>
<td>Filter</td>
<td>To filter water</td>
<td>Blocked: Decreases amount of water delivered to ringmain</td>
</tr>
<tr>
<td>Pump</td>
<td>To pump water to ringmain</td>
<td>Failed Broken: No water delivered to ringmain</td>
</tr>
<tr>
<td>Electric Power Supply</td>
<td>To power pumps</td>
<td>Failed no power: No water delivered to ringmain</td>
</tr>
<tr>
<td>Diesel Power Supply</td>
<td>To power pumps</td>
<td>Failed no power: No water delivered to ringmain</td>
</tr>
</tbody>
</table>

Table 6.10 Functionality and Failure Modes for components in the Deluge Pump System

In this particular example the pump system will fail if at any time less than two pumps are functioning, i.e. k=2. Given a demand for deluge the electric pumps will be brought on-line and the diesel pumps will be placed in standby. Only in the event of either of the electric pumps failing will the diesel pumps be brought into operation. Given a demand then, the electric power will start the electric pumps and providing the isolation values are open, the pressure relief valve and test valve are closed and the filter clear, water will be delivered successfully to the ringmain.

6.6.1 Fault Tree Construction for the Deluge System

The main concern for the deluge system is that a sufficient amount of water is delivered to the ringmain. The top event for the fault tree was therefore identified as ‘Deluge system fails to deliver a sufficient amount of water’. The top event will occur if less than two pump streams are functioning. A warm standby gate for the top event was implemented to indicate the number of channels that are required to fail to cause system failure. For the deluge system k=2 and n=4, the system will therefore fail if at any time three or more pump streams are in a failed state. The top level representation for the deluge system can be seen in figure 6.23.
The fault tree branches for each individual pump stream were developed by identifying all possible causes for pump stream failure. Inspection of the first electric pump stream highlighted that the following failures would cause the pump stream to deliver an insufficient amount of water to the ringmain:

- Pressure relief valve failed open: PRV1
- Test Valve failed open: TV1
- Either isolation valve failed closed: ISOL21, ISOL11
- Filter Blocked: F1
- Pump failed to start: EP1FS
- Pump failed whilst running: EP1FR
- Electric Power failed: EP

The fault tree structure for electric pump stream 1 was developed using these failure observations and is depicted in figure 6.24.

The fault tree structures representing the remaining three pump streams were identified in an identical manner and the fault tree representation of the deluge system developed (figure 6.25).
ELECTRIC PUMP STREAM 1 FAILS

G1

No Pressure at isol outlet

G2

Test valve fails open

G3

Isol valve failed closed

G4

Isol11

G5

Pump Fails to start

Pump Fails whilst running

G6

Filter blocked

Isol2 valve fails closed

Figure 6.24 Fault Tree for 'Electric Pump Stream 1 Fails'
By inspection of the fault tree structure representing 'Electric pump stream 1 fails' it can be seen that the gate governing the failure of the pump, G5, is an exclusive-OR gate which indicates that the input events are mutually exclusive, i.e. cannot occur at the same time. Unlike the standby example illustrated in section 6.5, the individual pump streams consist of components that have the potential to be in standby, dependent components, and those that are either working or failed, independent components. Conversion to the Markov transition diagram for the dependent elements was hence complicated by the combination of these dependent and independent components.

6.6.2 Analysis of the Deluge Pump System

It was noted that each pump stream had two failure modes. Firstly given a demand the pump stream could be unavailable and secondly, once available, the pump stream could fail before the required time $t_0$ had elapsed. Realistically, given a demand the deluge system would need to function for a period of about 12 hours, hence $t_0$ was set equal to 12 hours. The causes of the stream failing to start are represented by the independent elements of the fault tree structure for the stream, i.e. the pressure relief valve open, the isolation valves closed, the test valve open, the pump not starting and the filter blocked. It is realistic to assume that if these independent events are available at the start of the demand then they will not fail in $t_0$, as they are passive components. The causes of the second failure mode, pump stream failing once running, are represented by the dynamic elements of the tree, i.e. the pumps failing once running and the power supply failing. The power supplies in this example are common to more than one pump stream and were assumed to be perfectly reliable at the time a demand was made on the system.

An algorithm was required to identify and separate the two failure modes so that Markov analysis could be used for the dependent elements and FTA could be used for the independent events. The modularization algorithm used for the sequential failures identified the following gates as being independent: \{GTOP, G5, G6, G11, G12, G17, G18, G23, G24\}. The result, however, was not strictly true as the inputs to G5, G11, G17 and G23 contain the standby components, which
are dependent on each other. An alternative algorithm was hence required which could identify the standby events and any other events which had a direct relationship with the standby events. It was defined that the new standby gate would indicate that at some point below the gate in the fault tree structure a standby component would be encountered. The standby components in the fault tree structure were identified by a ‘$’ symbol at the beginning of their label. The $ was used to differentiate between these dependent component failures and those of the passive components such as the pipeline, whose failure was not dependent upon the primary streams. The power supplies were also deemed as dependent failures, as they are repeated events which can only fail during $t_0$, and removed for inclusion in the Markov model.

A new algorithm was developed based on these identification symbols. Whenever a standby gate was encountered by the program a search was initiated for the standby component on each input branch to the standby gate. Following the identification of the standby component $i$, the component was removed from the fault tree structure and stored in the array STAN$(i)$. In addition to this any other component $i$ which was identified as being dependent was stored in the array DEP$(i)$. For the deluge pump system, the removal of the standby events and the repeated events, resulted in the four pump streams becoming independent in nature. The independent subtrees were then solved using FTA and the probability of each stream failing to start was obtained. The removal of the dependent events and repeated events, and the solution of the independent subtrees, resulted in the standby gate being the last gate to solve. As the standby gate indicates dependent failures the gate was solved using Markov methods.

6.6.3 Construction of the Markov Diagram for the Deluge Pump System

The total number of states in the Markov diagram for the deluge system could not be determined using the procedure highlighted in section 6.5.2. due to the fact that repair was considered unrealistic in a 12 hour period. The construction of the Markov diagram for the deluge system was however based on the same principles as that of the procedure given in section 6.5.2. The predominate difference was that, for the deluge system, there existed only one initial state in level 1 and that was the state that symbolised that the system was working. For the Markov diagram, only the
components stored in STAN were included in each state. The initial state of the Markov diagram therefore represented both electric pumps working and both diesel pumps in standby, i.e.

\[
\text{STATE} = (0 \ 0 \ -1 \ -1) \quad \text{OR}
\]

The states in the second level of the diagram were developed from the initial state by changing, one at a time, each working or standby component found in the initial state into a failed component. As defined in section 6.5.2: If a working component is changed then the ‘0’ is changed to a ‘1’ and the first identified standby component brought on-line, i.e. ‘-1’ changed to a ‘0’, and if a standby component is changed then the ‘-1’ was changed to a ‘1’. Therefore taking the initial state for the deluge system the states in level 2 were produced by:

1) Changing EP1 from a working component(0) to a failed component(1) and DP1 from standby component(-1) to a working component (0).

2) Changing EP2 from a working component(0) to a failed component(1) and DP1 from standby component(-1) to a working component (0).

3) DP1 from standby component(-1) to a working component (0).

4) DP2 from standby component(-1) to a working component (0).

The array STATE, for level 1 and 2 of the Markov representation of the deluge pump system, was therefore identified as:
The states in the third level were developed from the second level and so on. In the instance that a new state developed was identical to a state previously created, the new state was not included in STATE. The Markov diagram for the deluge system was developed and is shown in figure 6.26.
The array TRANS was set up alongside the determination of STATE, however only failure links were identified and stored, as repair in a 12 hour period was deemed impossible. The TRANS matrix had the following form:

\[
\begin{bmatrix}
0 & \lambda_{yp1} & \lambda_{yp2} & \lambda_{yp15} & \lambda_{yp25} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \lambda_{yp2} & \lambda_{yp1} & \lambda_{yp15} & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & \lambda_{yp1} & 0 & 0 & \lambda_{yp1} & \lambda_{yp2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & \lambda_{yp1} & 0 & 0 & \lambda_{yp2} & \lambda_{yp15} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_{yp1} & \lambda_{yp2} & 0 & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_{yp2} & \lambda_{yp1} & 0 & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_{yp1} & \lambda_{yp2} & 0 & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_{yp1} & \lambda_{yp2} & 0 & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_{yp1} & \lambda_{yp2} & 0 & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_{yp1} & \lambda_{yp2} & 0 \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_{yp1} & \lambda_{yp2} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & \lambda_{yp1} \\
0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0
\end{bmatrix}
\]

The development of the Markov diagram thus far had not considered the effect on the individual pump streams due to the failure of the power supplies. The power supply components are not standby components but are repeated events which affect the individual pump streams during operation and therefore must be included in the Markov diagram and the state-transition matrix. The electric and diesel power supplies can fail at any time once the system is running. Each state was therefore investigated and the effect of either power supply failing was identified. From the original fault tree structure of the deluge pump system it was noted that DEP(1), ep, was present in the first two branches stemming from the standby gate and therefore affected the first two components in the Markov diagram. Similarly it was noted that DEP(2), dp, affected the 3rd and 4th components in the Markov diagram. In the event of the electric power supply failing, then, the system would move into a state where neither of the electric pumps would be functioning. Similarly, failure of the diesel power supply would cause the system to move into a state where neither of the diesel pumps would be available or functioning.
The initial state of the Markov diagram, for the deluge system, can be used to illustrate the inclusion of the power supplies in the diagram. In State 1 (S1) both electric pumps are working and both diesel pumps are in standby. Failure of the electric power supply would result in both electric pumps failing to function and both diesel pumps to be brought on-line, which is state 6 (S6) in figure 6.26. Alternatively, failure of the diesel power would result in state 11 (S11), where both diesel pumps are unavailable and both electric pumps are working.

To incorporate the failure transitions, due to the failure of the electric and diesel power supplies, each state i was investigated. Each component, c, in state i that was affected by the electric power supply was changed. If c was a working component then it was changed from a '0' to a '1' and the first identified standby component was brought on-line, i.e. '-1' changed to a '0'. If c was a standby component then it was changed from a '-1' to a '1'. All components affected by the electric power were changed and the result was a new state which represented the change in the system’s functionality due to the failure of the power supply. The new state produced was stored in NEWSTAT(m). STATE was then scanned and the state identical to NEWSTAT(m) identified as state j. A transition therefore exists between state i and state j, due to the failure of the electric power supply, i.e. TRANS(i,j) = RATES(EP,1). The procedure was then repeated for the diesel power supply.

It should be noted that certain states may communicate with the same state twice. For example the state (EP1:F, EP2:W, DP1:W, DP2:S) communicates with the state (EP1:F, EP2:F, DP1:W, DP2:W) twice. One transition represents the failure of the second electric pump, EP2, and the other transition represents the failure of the electric power supply, EP.

Following the identification of all transitions due to the failure of the electric and diesel power supplies the transition-rate matrix changed to:

\[
\begin{array}{cccc}
\end{array}
\]
6.6.4 Quantification of the Markov Diagram for the Deluge System

As with the standby gate analysed for a steady-state solution in section 6.5.3, the probability of system failure is given by the summation of the probabilities of residing in a failed state. For the deluge system, though, a time-dependent quantification was required as the probability of system failure is dependent on whether the system is available given a demand and whether it works continuously up to time $t_0$. The NAG routine outlined in section 6.2.2 was used to perform the analysis.

In order to perform a time-dependent analysis, the probability of starting in each state, $[P_1(0), P_2(0), \ldots, P_n(0)]$, requires determination. For the deluge pump system the initial states are created due to the availability of the four pump streams. State 1 (S1) in figure 6.25 is an initial state where all four pump streams are available. The probability of starting in S1 is given by the probability that pump stream 1 (PS1) is available AND pump stream 2 (PS2) is available AND pump stream 3 (PS3) is available AND pump stream 4 (PS4) is available. The availability of each pump stream, given a demand, is represented by the independent subtrees solved previously using FTA. Therefore the probability of starting in S1, given a demand, is equal to:

$$P_1(0) = (1-P_{PS1})(1-P_{PS2})(1-P_{PS3})(1-P_{PS4})$$

The probability of starting in any one of the 16 states in the Markov diagram of the deluge system were identified in a similar manner and are shown in table 6.11.
Table 6.11 Initial Probability for each state in figure 6.26

Following the application of the time-dependent analysis routine, the probability of residing in state i was stored in SOLN(i). The final step in the quantification procedure involved the identification of the failed states in the Markov diagram. In order to identify the failed states in the Markov diagram each state was scanned and if there existed n-k+1 or more failed components in the state then the state was considered a failed state and the probability of residing in that state was stored in the variable FAILEDSTAT. Following the inspection of all states the failure probability for the system was given by FAILEDSTAT. For the deluge pump system the failed states were identified by the program as being S12-S16. The failure probability of the deluge pump system, given a demand, was therefore obtained by FAILEDSTAT, which represented the summation of the probabilities of residing in states 12-16 at time t₀.

Using the failure data given in tables 6.12 and 6.13, and the fault tree structure developed in section 6.7.1, the *.ats and *.aqd data files for the deluge pump system were created (Appendix I). The failure probability of the deluge system was calculated using the combined model, via the solution of four independent fault trees and one 16 state Markov diagram, and shown to equal:

\[ Q_{SYS} = 5.126 \times 10^{-2}. \]
### Table 6.12 Failure Rate and Probability of Failure for Pumps and their Power Supplies

<table>
<thead>
<tr>
<th>Component</th>
<th>Failure Rate/Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pump fails to start</td>
<td>$Q = 9.5 \times 10^{-3}$</td>
</tr>
<tr>
<td>Pump Online</td>
<td>$\lambda = 7.0 \times 10^{-4}$</td>
</tr>
<tr>
<td>Pump in Standby</td>
<td>$\lambda = 3.5 \times 10^{-4}$</td>
</tr>
<tr>
<td>Electric Power</td>
<td>$\lambda = 2.0 \times 10^{-3}$</td>
</tr>
<tr>
<td>Diesel Power</td>
<td>$\lambda = 1.0 \times 10^{-4}$</td>
</tr>
</tbody>
</table>

The general procedure for the construction and analysis of a standby system containing independent and dependent component failures is depicted in figure 6.27.
6.7 Combination of Standby Gates with other Fault Tree Symbols

Analysing the entire offshore platform would have comprised of analysing different subsystems including the deluge pump system. Without the use of the new standby gate the entire system would have to be analysed on one Markov diagram, if an accurate solution was required. In order to use the combined model, i.e. solve independent subtrees using either FTA or Markov analysis, the standby gate must be solved as an independent subtree and returned as a new super event. As with the PAND gate, the independent subtrees are identified using the modularization algorithm outlined in section 6.4.1 Following the solution of a standby gate a super event is created which is represented by $Q_{TE}, \lambda_{TE}, \mu_{TE}$ and $w_{TE}$, which in the case of the deluge system would be time-dependent. The failure probability, $Q_{TE}$, is obtained directly from the Markov analysis. The repair rate of the super event cannot be obtained directly from the Markov diagram unless there exists only one failed state, in which case:

$$\mu_{TE} = -a_i, \quad \text{where } i \text{ is the failed state.}$$

In the situation where the Markov diagram contains more than one failed state, as with the deluge system, the unconditional failure intensity, $w_{TE}$, and the unconditional repair intensity, $v_{TE}$, are obtained from the diagram. $w_{TE}$ is calculated using equation (6.9) and the unconditional repair intensity, $v_{TE}$, is obtained by multiplying the probability of being in a failed state $i$, $Q_i$, by the transition rate which transports the system into a working state, $\mu_j$. Therefore,

$$v_{TE} = \sum_{i=1}^{n} Q_i \prod_{j} \mu_j \quad \text{where } n \text{ represents all failed states that communicate with a working state, } j \text{ = component which is repaired to cause transition to working state} \quad (6.15)$$

From $Q_{TE}, w_{TE}$ and $v_{TE}$ the remaining reliability parameters, $\mu_{TE}$ and $\lambda_{TE}$, are found using equations (2.15) and (2.12) respectively.
6.8 Summary

A new standby gate was developed in order to represent standby failures on the traditional fault tree diagram. A general procedure was created to convert the standby gate with n inputs into a Markov diagram and solve for steady-state. In industrial systems it is extremely rare that a system is made up of only standby components and for this reason an algorithm had to be developed which could solve a system containing both standby and independent failures. A general procedure was created which has the capability of modelling independent and standby failures on a fault tree structure. Following the development of the fault tree structure the tree was scanned and all standby failures and dynamic failures, which were directly linked to the standby events, were removed. The removal of the dependent failures resulted in the fault tree branches, stemming from the standby gate, becoming independent. These subtrees were solved using FTA and the remaining dependencies modelled on a Markov diagram.

A deluge pump system used on an offshore platform was modelled accurately using the new combined model. The configuration of the system can be noted to be typical of a system containing standby components and hence the procedure used to model the deluge system could be implemented as the analysis tool for other systems containing standby components. The main advantage with the combined model is that there is no need to model the entire system using a Markov diagram to obtain an accurate answer. The modelling process, however, is complicated if repair is considered feasible. The main problem with considering repair for the deluge system would have been that in the Markov diagram the system may be in a particular state due to the failure of the passive components and not the dynamic components. For example the deluge system could be in the state (EP1:F, EP2:F, DP1:F, DP2:W) due to the failure of electric pump 1 and 2 whilst running and the test valve in the first diesel pump stream being unavailable at time zero. The Markov diagram construction procedure outlined in this chapter would repair the standby diesel pump and create a transition to the state (EP1:F, EP2:F, DP1:W, DP2:W). This representation would be incorrect as the first diesel pump stream could only be brought back on-line if the test valve, tv3, was repaired. In order to include the correct repair transitions on the

168
Markov diagram the causes of being in a state would require identification and then the repair transitions could be determined. If a modelling technique was required that included repair a further investigation into the standby gate combined with independent failures would have to be initiated.

6.9 Conclusions

Two different types of dependent failures have been investigated and a new modelling procedure developed. The basis of the algorithm is to identify and separate independent and dependent failures. If separation can be achieved then the independent failures are modelled using FTA and the dependent failures are modelled using Markov methods. The sequential failures are modelled using the PAND gate and following the conversion of the gate to a Markov diagram the dependent failures can be replaced back in the fault tree structure as a super event which contains all relevant reliability parameters. The standby failures are modelled using the standby gate and the $ symbol, and following the solution of all independent failures the system can be converted into a Markov diagram based only on the number of dependent components in the system. The standby gate can also be integrated on the fault tree structure with other subsystems. As with the PAND gate, solution of the standby gate is achieved separately and replaced back into the fault tree structure as a new super event.

The development of the PAND gate and the standby gate have made it possible to include the dependent failures on a traditional fault tree diagram. Retention of the fault tree diagram is advantageous as the failure logic of the system is readily documented and the diagram itself is easy to construct from an engineering perspective. It can be concluded then, that by using the combined model an accurate result can be obtained for systems containing sequential and standby failures. In addition to this a solution is obtained efficiently by using a combination of FTA and Markov methods.
CHAPTER 7

OPTIMISATION OF A DEPENDENT SYSTEM

7.1 Introduction

There are many considerations which affect the decision on the implementation of a particular design. The first six chapters of this thesis have concentrated on one of these, the analysis of the system and the determination of its probability of failure. Although it is essential to design a reliable and safe system, especially if system failure can mean a catastrophic event, other parameters are also considered such as cost. Ideally a manufacturer would aspire to produce the most economical yet adequately reliable design.

Having developed an analysis tool that can effectively deal with dependent systems, research into the identification of the 'best' design possible for a system can be initiated. The combined model can be utilised at the analysis stage of the design process, which comprises: preliminary design, analysis, appraisal and redesign. Once the reliability has been obtained the design can be judged for its acceptability and a decision can be made on whether to implement the design or not.

This chapter describes the development of an optimisation method that can be used to identify the 'best'; design for a particular system. The fundamental findings postulated in this chapter are summarised in (ref. 49, 50).

7.2 Representation of Design Alternatives

In order to be able to determine the 'best' design a number of design options require consideration. Ideally all possible designs should be determined and analysed. This implies that a large number of trees would need to be constructed and evaluated before a decision upon the best design could be formed. The process of constructing and analysing all potential designs would be extremely time consuming, potentially error prone and totally impractical for any real system design problem.
An advantage of retaining the fault tree structure in the combined model, apart from its logical attractiveness, is that all the potential design options can be represented on one tree by using **house events** (ref. 51,52). House Events (Figure 7.1) are used to model the existence or non-existence of an event by assigning Boolean values of True (1) or False (0).

These events can effectively turn branches of the fault tree structure on and off. Figure 7.2 can be used to illustrate the use of house events where a valve used in a system can be selected from two types. In the situation where type 1 is chosen HV1 would be set equal to true, i.e. 1, and HV2 set to false, i.e. 0, illustrated in figure 7.2. Alternatively if type 2 was chosen then HV1 would be set equal to false and HV2 set to true.

The failure logic for the set of design variations can now be represented on a single fault tree structure by using house events. The implementation of house events
reduces the number of different fault trees that are required to be constructed to just one. For each different design, however, the relevant house events require modification and prior to analysis the fault tree structure needs to be reduced to a form which contains only gates and basic events. The flowchart depicted in figure 7.3 shows how any encountered house events will be treated in order to produce the reduced fault tree structure.

ACTION 1 = Delete house event input to gate. If this only leaves one input event delete AND gate, change input event to be an input to the gate at next level up.

ACTION 2 = Gate output is true. Proceed up through tree structure until an AND gate is encountered. Delete this input branch up to AND gate. If this only leaves one input event delete AND gate, change input event to be an input to the gate at next level up.

ACTION 3 = Gate output is false. Proceed up through tree structure until an OR gate is encountered. Delete input branch up to the OR gate. If this is the only input to the OR gate change to an input event to the gate at the next level up.

ACTION 4 = Delete House Event to gate. If this only leaves one input to the OR gate change to an input event to the gate at the next level up.

Figure 7.3 Algorithm for removal of House Events following a design specification

To illustrate the reduction process the example fault tree in figure 7.2 can be utilised. The house event representing the selection of valve 1, HV1, is equal to 1 and is an input to an AND gate. Following figure 7.3 the house event is deleted and as this leaves only one input, the AND gate is also deleted. Similarly, the house event representing the selection of valve 2, HV2, is equal to 0 and is an input to an AND
gate. The Gate is false and the fault tree structure beneath, and including the gate, is deleted up to the next encountered OR gate. The reduced fault tree structure takes the form required, where the valve failure is due to the selected valve, i.e:

![Fault Tree Diagram]

Each design is represented by a set of unique variables which determine the status of the house events such that the reduction of the fault tree structure represents the failure causes of the design under inspection. Failure causes of any other design can be represented by changing the respective values attributed to the individual house events. The time required to manually enter all new data for each design, combined with the analysis computation time, would prove to be excessive if the number of design options was large. To overcome this problem it was decided to embed the analysis within a structured optimisation procedure. The design variables are then automatically set by the optimisation procedure and the appropriate house events set. The reduced fault tree would then be analysed and the reliability/availability of the design determined. This process would continue until the optimisation scheme converged to an optimal design.

Andrews (ref. 52) described an approach by which the optimal performance of a safety system in terms of its availability was determined using the FTA method. With the advances in FTA an updated, more efficient, optimal design search was achieved three years later (ref. 51). Andrews and Pattison solved the fault trees by converting the structure into a Binary Decision Diagrams (ref. 53) where the house events were treated by ascribing probabilities of 0 and 1. A recommendation for the best design was produced via an optimisation process. The predominant difference between this part of the investigation and previous research was that the tool, used to perform the analysis of the system, would be the newly developed combined model implying that
the analysis would now be more appropriate for systems with dependencies than previous studies.

### 7.3 Optimisation Techniques

As outlined in the introduction, the problem is to determine the best design where a number of design options exist. In order to obtain an optimal design, without evaluation of all possible designs, an optimisation procedure can be utilised. Many types of optimisation techniques exist and which one to apply is dependent on the characteristics of the problem. The function we wish to minimise/maximise, \( f(x) \), is often referred to as the **Objective Function** and represents the assessment criteria for the function. The fundamental problem with optimising a safety system design is that the objective function cannot be explicitly defined. This is due to the fact that the objective function, which is related to the structure function of the system, changes alongside any design variable modification. A detailed account of the current optimisation techniques and those applicable to reliability problems can be found in the doctorate thesis by Pattison (ref. 54). Three main areas were investigated namely: Linear Programming, Gradient based optimisation techniques and Random Search methods. In summary the following conclusions were postulated:

1) Linear programming optimisation techniques are inappropriate for application to the safety system optimisation problem. The main reasoning for this stems from the need to have an objective function to give a strict mathematical formulation of the design problem.

2) Gradient methods are inappropriate for the safety system design optimisation problem as the objective function, \( f(x) \), and its derivatives, \( Vf(x) \), are required to exist and be of a continuous nature.

3) Random search methods rely on function evaluation only and generally pay little attention to the structure of the objective function. These techniques can be applied to the safety system design problem, however such methods may require more computation in obtaining the optimal.
From this research it was concluded that most effective class of optimisation technique for this type of problem is that of random search. Pattison used a Genetic Algorithm to find the optimal design for a High-Integrity Protection System (ref. 54). In addition to this various other authors have used a Genetic Algorithm successfully in a reliability context (ref. 50,51,55). Painton and Campbell showed that a Genetic algorithm performed favorably when compared to the results obtained from an enumeration of a configuration space for a PC system (ref. 55). In this paper although the problem was not complex it was found that other techniques, namely that of a hill-climbing procedure, could not obtain a global optimal solution.

The technique chosen to perform the optimisation of a system with dependent failure events was therefore the Genetic Algorithm.

7.4 Genetic Algorithms

A genetic algorithm is a type of evolutionary computation method. Such methods employ a search strategy that solves complex problems by simulating evolution via a computer algorithm. A common principle governing evolutionary computation methods is that each algorithm begins the search from a population of trial solutions. Each of the trial solutions is then assessed by a measure of performance and used in combination with a selection process to determine which solutions continue into the next generation. New solutions can be created via the alteration of the existing solutions. Each type of evolutionary computation method functions by concentrating on a different aspect of the evolution process. Genetic algorithms concentrate on the genetic link between existing and newly formed structures.

7.4.1 Background

Genetic Algorithms (GA) were invented by John Holland in the 1970's to mimic some of the processes observed in natural evolution (ref. 56). In evolution each species strives for survival against complicated and changing environments. According to Darwin those organisms better adapted to the environment survive and reproduce. Natural selection suggests that the origin and diversification of species results from the gradual accumulation of individual modifications.
As each species changes to adapt to a new environment so the genetic make-up contained within a chromosome is modified. New generations are formed by reproduction and new chromosomes are generated due to the crossover of each parent’s genetic information. Each chromosome is comprised of a number of genes, which encode a particular feature of the individual, and occasionally these genes may be mutated which results in a new chromosome structure. Each gene can adopt several different values in order to describe its characteristics. The value of any particular gene is referred to as an allele.

The work completed by Holland was based on two methods: the encoding of complex structures by simple representation (bit strings) and the process of improving such structures by simple transformations. Holland began work on algorithms that manipulated strings of binary digits. The strings can be viewed as chromosomes and the individual bits of the string as genes. Holland awarded each chromosome a fitness value depending on its strength in the surrounding environment. An evolution process was then developed by performing a number of generations from an initial population based on the value of each chromosome at each generation. The selection at each generation was biased due to the chromosome value so that those with the best evaluation tended to be reproduced more often than those that possess a poor fitness. The reproduction strings were then subjected to the genetic operators of crossover and mutation and a new generation produced.

In summary the algorithms developed by Holland combine survival of the fittest among string structures with a stochastic structured information exchange to form a process similar to that of evolution. The application of GA to an optimisation setting was developed, based on Holland’s work, by DeJong in 1975 (ref. 60). Following these developments GAs have been implemented in a wide range of practical optimisation problems.

Table 7.1 contains a description of the main terms used in GAs and the remainder of this chapter.
<table>
<thead>
<tr>
<th>TERM</th>
<th>DESCRIPTION</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHROMOSOME(string)</td>
<td>A binary solution string for the problem</td>
</tr>
<tr>
<td>GENE</td>
<td>A variable contributing to a solution string, represented as a bit</td>
</tr>
<tr>
<td>ALLELE</td>
<td>Value of a gene</td>
</tr>
<tr>
<td>POPULATION</td>
<td>A set of solution strings</td>
</tr>
<tr>
<td>FITNESS</td>
<td>Performance measure given to a solution string</td>
</tr>
<tr>
<td>REPRODUCTION</td>
<td>Process of selecting chromosomes/strings based on their fitness value</td>
</tr>
<tr>
<td>CROSSOVER</td>
<td>Process of mating of two chromosomes/strings</td>
</tr>
<tr>
<td>MUTATION</td>
<td>Process of mutating a gene</td>
</tr>
</tbody>
</table>

Table 7.1 Terms used in GAs

### 7.4.2 Methodology

GAs can vary in complexity depending on which genetic operators are used. For the purpose of this thesis the simple genetic algorithm, described by Goldberg (ref. 58), was utilised. The general procedure of the simple GA involves the identification of an initial population followed by successive applications of the reproduction, crossover and mutation operators. Prior to the application of the simple GA certain parameters require identification. These are:

- The representation scheme to be used for the problem.
- A fitness measure for each string in a population.
- Definition of the parameters and variables that will control the algorithm
- Definition of a performance measure for each run of the GA
- Definition of a set of criteria for termination of a run.

To provide a full understanding of the procedure used to perform a GA each of the parameters identified above will be discussed.

#### 7.4.2.1 The Representation Scheme

Initially GAs require the natural parameter set of the optimisation problem to be coded as a finite-length string over some finite alphabet. For example say the function \( f(x) = x^2 \) was optimised over the interval \([0,31]\), initially the parameter \( x \) would be required
to be coded into a finite-length string. The length of the string must include the maximum value the parameter x can hold, i.e. 31. A string of length 5-bits is required as the coded string uses the binary digits 1 and 0, where

\[
\begin{align*}
11111_2 &= 31 \\
00000_2 &= 0
\end{align*}
\]

The representation scheme used for a GA is therefore a binary string. Application of the GA to a safety system design optimisation problem requires the system’s design variables to be coded into a binary string. For a system with several different design variables the binary string comprises of contiguous structure of smaller binary strings which correspond to the separate variables.

### 7.4.2.2 Fitness Measure

An initial population is set up randomly by placing 0’s and 1’s in each bit position of each individual string. Each string in the initial population is then decoded and awarded a fitness value depending on the design specifications. The fitness value is a measure of the performance of the string in its environment and the GA uses it to produce the next generation. In a reliability context the fitness value could be based on the reliability of the system. The fitness value is obtained via the evaluation of the fault tree. As the reliability value is generally of an inappropriate form for use in the algorithm the fitness value is obtained via some mapping scheme. The scheme used is dependent on the problem being solved.

### 7.4.2.3 Controlling Parameters

The parameters that control the simple GA are the genetic operators which perform reproduction, crossover and mutation.

**The Reproduction Operator**

Reproduction is a process in which individual strings are selected due to their fitness. Depending on whether the problem is to minimise or maximise, the strings will be reproduced according to the lowest or highest value respectively. The reproduction
operator can be implemented in the algorithm in a number of different ways, but the most commonly used is that of the Biased Roulette Wheel. The process works by allocating a proportion of a roulette wheel to each string according to its fitness value. The proportion, $P_i$, that any particular string allocates is determined by dividing the strings fitness, $f_i$, by the total fitness of the population, i.e:

$$P_i = \frac{f_i}{\sum_{j=1}^{n} f_j} \quad (7.1)$$

To illustrate the reproduction process the four strings depicted in table 7.2 can be utilised. The fitness value for each string is calculated and the roulette wheel is developed.

<table>
<thead>
<tr>
<th>STRING</th>
<th>FITNESS VALUE, $f_i$</th>
<th>PROPORTION OF WHEEL, $P_i$</th>
<th>INTERVAL BOUNDARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>1) 01111100</td>
<td>52.5</td>
<td>0.554</td>
<td>0.554</td>
</tr>
<tr>
<td>2) 01010101</td>
<td>11.5</td>
<td>0.121</td>
<td>0.675</td>
</tr>
<tr>
<td>3) 11110000</td>
<td>25.9</td>
<td>0.273</td>
<td>0.948</td>
</tr>
<tr>
<td>4) 00001100</td>
<td>4.8</td>
<td>0.052</td>
<td>1</td>
</tr>
<tr>
<td>TOTAL</td>
<td></td>
<td></td>
<td>94.7</td>
</tr>
</tbody>
</table>

Table 7.2: Initial Populations Fitness Values and Roulette Wheel Proportions

By using the biased roulette wheel fitter strings have a higher representation in succeeding generations. For this particular example string 1 would have the best chance of being represented in the proceeding generation as it occupies the interval 0 to 0.554 and hence has a 55.4% chance of reproduction.

The selected strings are then entered into the mating pool and subjected to the second operator, crossover.

**The Crossover Operator**

The main function of this operator is to combine good features of the strings created during reproduction in the hope of producing a fitter string. The modified strings, developed by crossover, may allow new points in the search space to be identified and evaluated. The newly reproduced strings are mated in pairs and although various
different crossover techniques exist the traditional one-point method will be implemented. As its title suggests the crossover is initiated about a single point. For each pair of strings a random number is generated, if this number lies between zero and the crossover rate, typically say 0.7, then the pair are crossed. The position of crossover is also generated randomly. An integer position k along the string length less one [1,L-1] is generated, the pair is then crossed between k+1 and L inclusively.

To illustrate the crossover procedure the two eight-bit strings, S1 and S2, can be utilised.

\[
S_1 = 01111100 \\
S_2 = 01010101
\]

During crossover, the crossover point is generated, in this example four, and the two new strings, S1' and S2', are created by swapping over the parts of each string from the bit positions 0 to 3.

\[
S_1' = 01110101 \\
S_2' = 01011100
\]

The combined emphasis of reproduction and the structured, though stochastic, information exchange of crossover give GAs much of their power (ref. 58). The crossed strings are then subjected to the third and final operator, mutation.

**Mutation Operator**

Mutation has been shown to be a means for reintroducing diversity into the population (ref. 56). Mutation has the capability of throwing the search into a new area of the search space and when used with reproduction and crossover it acts as an insurance policy against loss of important information.

The mutation operator works by changing individual bits in a string. For each bit in a string a random number is generated, if this figure is less than or equal to the mutation rate then the bit under inspection is changed. As the strings are represented in binary, if the bit under inspection is a 0 it is changed to a 1 and vice versa.
The new population is hence developed and the process of reproduction, crossover and mutation is continued until the required number of generations has been performed or until convergence has taken place in the population.

Another parameter that requires identification is the size of the initial population. The size can vary and is generally determined by the number of different system designs. Large populations provide a greater search of the solution space. It should be noted, however, that increasing the size also increases the computation effort required to perform the GA.

7.4.2.4 Performance Measures and Termination Criteria for a GA

The most popular termination criteria is to state the maximum number of generations evolved. In addition to this the GA can be terminated when a certain degree of convergence has been portrayed. There exists no general rule for convergence criteria as it is problem dependent. Some authors believe application of convergence criteria to be problematic as fitness of a population may remain static for a number of populations before a superior individual is found (ref. 57). One way of monitoring the behaviour of the GA is to record the 'best-so-far' string. In addition to this the average fitness or unreliability can be used as a measure of convergence for the GA.

7.5 Application to an Industrial System

Having outlined the general procedure for the use of a GA, it can now be applied to an industrial example, by the use of the combined model and house events, to find the best design option.

7.5.1 Deluge Pump System

The industrial system used was the offshore platform deluge pump system previously modelled using the combined model in chapter 6, section 6.7 (Figure 6.22). To reiterate, the function of the deluge pump system is to pump a sufficient quantity of water to the ringmain given a demand. For the purpose of this analysis it is assumed that the deluge system is required to function continuously for a period of 12 hours. For this system different types of pump can be implemented, each with varying capacities. The pump system will fail if at any time the capacity of the pumps is less
than 100%. The system comprises electric pumps and standby diesel pumps. Only in the event of a failure of an electric pump will a diesel pump be brought into action. Each electric pump is powered by a common electric power supply and similarly the diesel pumps are powered by a common diesel source. The isolation and test valves are used for maintenance purposes only and the pressure relief valve is used to decrease the pressure in the pipeline. Given a demand for deluge the electric power supply starts the electric pumps and the isolation valves should be open, the filter clear, the pressure relief valve closed and the test valve closed.

Even with a relatively small number of different components in a system the total number of different design options can be large. The deluge system optimisation problem has been restricted to have a relatively small number of alternatives in order to produce an exhaustive search solution to compare to that of the GA. The available design options are listed in table 7.3 and there exists 520 possible designs.

<table>
<thead>
<tr>
<th>Design Options</th>
<th>Design Variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>How many pumps, of the same type, are needed (1, 2, 3, 4)?</td>
<td>NE</td>
</tr>
<tr>
<td>What Type of Pump to use (A, B, C, D, E)?</td>
<td>PO</td>
</tr>
<tr>
<td>Maintenance Test Interval for System (1-26 Weeks)</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 7.3 Design Options for Deluge Pump System

It was assumed that the number of diesel pumps would be identical to the number of electric pumps. Therefore the maximum number of pumps the deluge system could contain would be eight, and the least two. For each variation of pump design the cost and pumping capacity changed. Individual pump data is illustrated in table 7.4 and the remaining component data was given in Chapter 6, Table 6.13.

<table>
<thead>
<tr>
<th>Component</th>
<th>Capacity</th>
<th>Failure Rate Per Hour</th>
<th>Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>On-line</td>
<td>Standby</td>
<td></td>
</tr>
<tr>
<td>Pump A</td>
<td>100%</td>
<td>3.5x10^-4</td>
<td>4500</td>
</tr>
<tr>
<td>Pump B</td>
<td>100%</td>
<td>1.0x10^-3</td>
<td>4700</td>
</tr>
<tr>
<td>Pump C</td>
<td>50%</td>
<td>7.0x10^-4</td>
<td>3200</td>
</tr>
<tr>
<td>Pump D</td>
<td>50%</td>
<td>5.0x10^-4</td>
<td>3300</td>
</tr>
<tr>
<td>Pump E</td>
<td>33.33%</td>
<td>1.0x10^-3</td>
<td>2600</td>
</tr>
</tbody>
</table>

Table 7.4 Pump Data
7.5.2 Component Unavailability

The causes of each pump stream failing is either that it fails to start or that it fails once running. Failure to start is governed by the passive components present on the line, i.e. the filter, isolation valves, pressure relief valves and test valves. Ideally when a demand is present these static components should be available. For these static components failure will only be detected during a maintenance inspection. Such failures are classed as unrevealed and their respective probability of failure is calculated using an approximation of equation (2.20), given in Chapter 2, which is shown in equation (7.2).

\[ Q_i = \lambda_i \left( \tau \cdot \frac{\theta}{2} \right) \]  

(7.2)

where:

- \( \lambda_i \) = the failure rate of component i
- \( \tau \) = the mean time to repair
- \( \theta \) = the maintenance test interval

The probability of each different pump type failing to start was assumed and is shown in table 7.5.

<table>
<thead>
<tr>
<th>Component</th>
<th>Probability of failure, ( Q_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pump (Type A) fails to start</td>
<td>0.01</td>
</tr>
<tr>
<td>Pump (Type B) fails to start</td>
<td>0.012</td>
</tr>
<tr>
<td>Pump (Type C) fails to start</td>
<td>0.0095</td>
</tr>
<tr>
<td>Pump (Type D) fails to start</td>
<td>0.011</td>
</tr>
<tr>
<td>Pump (Type E) fails to start</td>
<td>0.012</td>
</tr>
</tbody>
</table>

Table 7.5 Probability that Pump i Fails to Start

The failure of the power supply and the pumps failing whilst running are revealed failures, i.e. the failures of such components will be immediately apparent, and the probability of failure is therefore calculated using equation (2.19) given in chapter 2. As these are dependent failures, however, the combined model performs a Markov analysis of these components and hence each component requires a failure rate. As
the deluge system is only required to function for a 12 hour period each component is
demed as non-repairable.

7.6 The Optimisation Problem

Due to the differing capacities of the pumps not all of the 520 designs were feasible.
For example those designs which contain pump type E require three pumps to be
working and therefore a minimum of 4 must be present in the design in order to
deliver a sufficient water supply. Removing all infeasible designs reduced the total
number of possible designs to 494. For this particular design problem the limitations
were set as:

1) The total system cost must be less than or equal to 18,000 units. This relates to the
cost of the pumps only.

2) The average maintenance effort spent each week must not exceed 1 hour. This
relates to the pumps only and each pump is said to take 2 hours to test at each
inspection.

The design problem was specified as:

'To maximise the reliability of the system within the resource constraints'

Reliability is the term used to describe the behaviour of the system because with the
deluge system it is required to work from the time of the demand continuously over a
time period specified, in this case 12 hours. Including the cost and maintenance
constraints limited the number of possible designs to 122 and turned the optimisation
problem into a constrained one.

7.6.1 Fault Tree Representation of the Deluge Pump System

A single fault tree structure was constructed using house events to represent all
possible designs alternatives and is illustrated in Appendix II. Each pump stream has
a house event attached to the first gate to model whether that particular pump stream
had been chosen in the present design. The house event symbolises that the pump
stream will fail to pump if either the pump stream has not been chosen in the design, HPi, or it has been chosen, HPi, but fails to function. This scenario is represented in figure 7.4 where the house event, HP2, for the pump stream ‘electric pump 2’ is set equal to 1 which represents that the pump stream has not been chosen and HP2 is therefore equal to 0.

![Figure 7.4 House Event for number of Pump Streams in a Design](image)

In a similar fashion house events were also used with the five different pump type options and were set according to the pump type chosen, i.e. the value of PO. Each pump stream had each different pump type included however only one house event was set equal to true depending on whether PO = 1, 2, 3, 4 or 5. The configuration for pump type A is shown in figure 7.5.
7.6.2 Binary String Coding

In order to utilise the simple GA described in section 7.4 each individual design required coding into a binary string. For each of the three design variables a certain portion of the string was assigned which was capable of holding the maximum value possible for each variable. The number of pumps range from 1 to 4 and hence was assigned 2 bits of the binary string.

There exists five different pump type options for this problem and therefore the portion of the string governing the pump type had to accommodate a maximum of five. The pump type variable was assigned 3 bits of the coded string. A problem became apparent as a 3-bit string has a range from 0 to 7 and the number of different pumps varied from 1 to 5. It was decided that the range should be mapped and for the bit value \( x \), where \( x \) is the value of the bit string, the formula shown in equation (7.3) was devised.

\[
Pump \ Type \ i = \left( \frac{4x}{7} + 1 \right)
\]

(7.3)

where the value obtained is rounded to the nearest integer.
The third design variable, the inspection interval, has a maximum value of 26 weeks and was therefore assigned 5-bits of the binary string. Again the maximum bit value was greater than the range required for the variable. As with the pump type, a formula was devised, equation (7.4)

\[
\text{Inspection Interval in weeks} = \left( \frac{25x}{31} + 1 \right)
\]

(7.4)

where the value obtained is rounded to the nearest integer.

The total length of the string for each design therefore comprised of 10-bits (Figure 7.6), and hence the last ten bits of a 16-bit string was used to store each design.

![Figure 7.6 Binary String Composition for Deluge System Design](image)

### 7.6.3 Decoding a Design String

Each of the design variables were decoded from a particular design string by using MASK strings and the binary operator IAND, which is an intrinsic FORTRAN function. For each different design variable a mask string was developed covering only the bits of the string which correlated to the variable under inspection. For example the mask string correlating to the inspection interval was created as:

\[
0000000000111111
\]

where each bit in the string representing the range of the inspection interval was set equal to the binary digit 1 and all remaining bits were set equal to 0. Similarly the mask strings for the number and type of pumps were created and all three mask strings are shown in table 7.6.

<table>
<thead>
<tr>
<th>MASK1 (0)</th>
<th>0000000000111111</th>
</tr>
</thead>
<tbody>
<tr>
<td>MASK2 (PO)</td>
<td>0000000111000000</td>
</tr>
<tr>
<td>MASK3 (NE)</td>
<td>0000001100000000</td>
</tr>
</tbody>
</table>

Table 7.6 Mask Strings for Each Design Variable
Each design string was decoded by logically multiplying, using the IAND operator, each mask string with the design string to yield the variable settings for every new design. To illustrate the decoding process the following string can be utilised:

\[ 000000101101110 \]

To determine the inspection interval the string is multiplied, bit by bit, by its mask string:

\[
(000000101101110), (0000000000011111) = 0000000000011110
\]

The bit value for this string is 14 and using the mapping equation (7.4) the inspection interval for the given design string is equal to:

\[
\text{Integer Part} \left( \frac{25 \cdot 14}{31} + 1 \right) = 12 \text{ weeks}
\]

Similarly the pump type can be shown to equal Type C, using MASK2 and equation (7.3), and the number of pumps to equal 4 in total using MASK3.

A *.hqd datafile was created with each house event HPi initially set equal to one. After each string in a population is decoded the *.hqd file is updated and the appropriate house events are set. For example the string decoded above would produce the changes to the *.hqd data file shown in figure 7.7.

Once the appropriate house events have been set the fault tree structure is reduced using the rules outlined in section 7.2 such that the fault tree comprises of gates and basic events alone. The decoded inspection interval is passed into the combined model and used to calculate the probability of failure for each of the relevant components in the design.
7.6.4 Objective Function Value of a String

In order to be able to perform the optimisation search each string required a fitness value in order to initiate the reproduction process. The determination of each individual string’s fitness comprised of an analysis to gain a value to represent the reliability of the system. For any safety system, such as the deluge system, a high reliability would be expected. Fitness values all based on high reliability levels do not differentiate between designs when used to guide the reproduction into the next generation. A string with the reliability equal to 0.99 would be reproduced with a very similar likelihood as a string with reliability 0.95. To resolve this a mapping scheme was created whereby 0.99 was rewarded significantly more than 0.95. To distinguish between designs, the reliability improvement was rewarded by a non-linear function shown in figure 7.8. To determine a formula to represent the function shown in figure 7.8 MAPLE was used to fit a function to the following arbitrary four points:

When $x = 0.9$ $y = 0.0$
When $x = 0.95$ $y = 1.0$
When $x = 0.965$ $y = 5.0$
When $x = 0.98$ $y = 20.0$
Any system whose reliability was lower than 90% was rejected.

Using MAPLE the relationship between the systems reliability, $x$, and string fitness was found to be:

$$
\text{fit-value} := \begin{cases} 
324 - 720x + 400x^2 & \text{when } x < 0.95 \\
13601 - 28671x + 15111x^2 & \text{when } 0.95 < x < 0.965 \\
30984 - 64698x + 33778x^2 & \text{otherwise}
\end{cases} \quad (7.5)
$$

Hence given any system’s reliability value a corresponding fitness value could be awarded using equation (7.5). For any system design which violated either of the design specification constraints, the fitness value awarded to that particular string was set to zero which eliminated it from influencing the next generation.

Having developed a means of evaluating the fitness of a string the GA could now be implemented.

### 7.6.5 Convergence Criteria for the GA

Prior to the execution of the GA a termination point had to be stated. It was assumed that for the GA to be terminated prior to the exhaustion of the maximum number of generations set there should exist an indication of convergence by the average fitness
given at each generation. Figure 7.9 depicts the convergence pattern expected and highlights that as the GA converges to an optimal the difference between the average fitness of successive generations decreases. The greatest difference in average fitness values between the 20th and 30th generation in figure 7.9 was found to be 0.1. The GA was therefore assumed to have converged if the difference between the maximum average fitness value and the minimum average fitness value, spanning over ten generations, was less than 0.5.

![Figure 7.9 Convergence shown by Average Fitness](image)

In addition to the convergence criteria based on the average fitness the GA was also terminated if either the maximum number of generations were performed or if all the strings in a generation represented the same design, i.e. were all identical. Therefore there existed three different types of convergence criteria:

**TYPE I:** GA terminated due to the difference between the maximum average fitness and minimum average fitness \( \leq 0.5 \)

**TYPE II:** GA terminated as all strings in a given population are identical

**TYPE III:** Maximum number of generations performed.
7.7 Exhaustive Search

In order to assess the performance of the optimisation of the deluge system using a GA it was decided that all designs should be evaluated and the optimal design identified. Once highlighted the optimal string could then be compared to that given by the GA. As the number of design options for the deluge system was small the exhaustive search was completed in the order of minutes. The fitness value for each design string was calculated using equation (7.5) and the reliability of each design was obtained using the combined model. A summary of the exhaustive search is shown in table 7.7. Inspection of table 7.7 highlights that the optimal string was found to be 4 pumps of Type A, with an inspection interval of 1344 hours (8 weeks). The cost of the system is 18,000 units and the average maintenance each week on the pumps is 1 hour. It can be noted that this is on the boundary of the constrained problem.

<table>
<thead>
<tr>
<th>TYPE</th>
<th>NUMBER OF PUMPS</th>
<th>INSPECTION INTERVAL</th>
<th>FITNESS</th>
<th>Reliability</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>2</td>
<td>672</td>
<td>53.1024</td>
<td>0.996135</td>
</tr>
<tr>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>62.906811</td>
<td>0.999741</td>
</tr>
<tr>
<td>B</td>
<td>2</td>
<td>672</td>
<td>53.66274</td>
<td>0.996350</td>
</tr>
<tr>
<td>C</td>
<td>4</td>
<td>1344</td>
<td>45.930757</td>
<td>0.993266</td>
</tr>
<tr>
<td>D</td>
<td>4</td>
<td>1344</td>
<td>46.11837</td>
<td>0.993344</td>
</tr>
<tr>
<td>E</td>
<td>4</td>
<td>1344</td>
<td>0.34223</td>
<td>0.929250</td>
</tr>
<tr>
<td>E</td>
<td>6</td>
<td>2016</td>
<td>35.5003</td>
<td>0.988621</td>
</tr>
</tbody>
</table>

Table 7.7 Summary of Exhaustive Search (Optimal String Highlighted)

7.8 Genetic Algorithm Implementation

Two different types of optimisation were performed on the deluge system: A feasible region search and a penalty function approach. For each method an initial population had to be created. The initial population, size 20, was generated randomly. A random number was produced for each of the 10-bits in each string and a 0 or 1 placed in the string space depending on the value of the random number. Out of the 16-bit string only the positions 0 to 9 had to be initialised.

7.8.1 The Feasible Region Search

Each string created in the initial population was monitored to ensure that all 20 strings represented a feasible design. Difficulties arise due to infeasible points being rejected
as reduced information concerning the space is gained. To overcome the fact that not all the strings in the initial population may be valid, a clause was incorporated into the program. A new subroutine 'Valid' was created in order to check each newly developed string in the initial population. Each string was passed through to subroutine Valid where it was decoded and the constraints evaluated. If either of the constraints were violated then another string was created and the process continued until all 20 initial strings were valid. By introducing this clause the initial population was set up and contained 20 viable designs.

The program was set to run for a maximum of 200 generations or until the process converged. As postulated in section 7.6.5 the convergence criteria was set such that the GA stopped if all the strings in a population were the same or if the magnitude of difference between the minimum and maximum average fitness for ten generations was less than or equal to 0.5. Prior to the execution of the GA the crossover and mutation rates had to be set. It was decided that the crossover rate should be relatively high and the mutation rate relatively low. The crossover rate was set high in order to increase the chance of information exchange which in turn results in more of the solution space being explored and a reduction in the chance of settling for a false optimum (ref. 59). Similarly the mutation rate was set low so that diversity was occasionally reintroduced into the search. A high mutation rate would lose all the structure of the GA and randomise the search. For an initial population of 20 strings, each containing 10-bits of information, on average 1-bit per population would be changed by setting a mutation rate of 0.005. The algorithm therefore used a mutation rate of 0.005 and the crossover rate was set as 0.75.

At each generation the five fittest strings were stored and compared to the previously nominated best five. Proceeding the convergence of the GA the overall five best strings were output together with their cost and maintenance requirements. Using this information the most appropriate design, in terms of cost and reliability, could be identified. The flowchart depicted in figure 7.10 illustrates the procedure followed in order to perform an optimisation search using a GA.
7.8.1.1 Results and Conclusions for the Feasible Region Search

In order to support the efficiency of the GA in a reliability design problem the process was completed for ten different seeds. Table 7.8 shows the design characteristics for the best five designs resulting from each run and table 7.9 depicts the performance measures for the overall best design and the GA.

The results showed that eight out of the ten runs highlighted the best string to be the actual optimum gained from the exhaustive search, table 7.7. The two runs, runs 2 and 5, that failed to highlight the actual optimum both postulated that the best design comprised of 4 pumps of type A with an inspection interval of 11 weeks. The reliability of this design is 0.99942 and the fitness value awarded for the design was 62.0234. The difference between this design and the optimal can be shown to be $3.21 \times 10^{-4}$ in reliability terms and 0.883411 in fitness terms. From these observations it
was concluded that the difference was negligible and that the implementation of the GA in the attempt to optimise the deluge safety system was successful.

<table>
<thead>
<tr>
<th>GA Run</th>
<th>TYPE</th>
<th>No. OF PUMPS</th>
<th>0 Cost</th>
<th>Maint</th>
<th>Qi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1680</td>
<td>0.8 hr</td>
<td>0.99960</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2688</td>
<td>0.5 hr</td>
<td>0.99888</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2856</td>
<td>0.47 hr</td>
<td>0.99870</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>3528</td>
<td>0.38 hr</td>
<td>0.99774</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>4</td>
<td>2016</td>
<td>0.66 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2184</td>
<td>0.615 hr</td>
<td>0.99931</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2352</td>
<td>0.57 hr</td>
<td>0.99919</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2520</td>
<td>0.53 hr</td>
<td>0.99904</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>4368</td>
<td>0.31 hr</td>
<td>0.99588</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1512</td>
<td>0.88 hr</td>
<td>0.999679</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2016</td>
<td>0.66 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2184</td>
<td>0.615 hr</td>
<td>0.99931</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2352</td>
<td>0.57 hr</td>
<td>0.99919</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1512</td>
<td>0.88 hr</td>
<td>0.999679</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1680</td>
<td>0.8 hr</td>
<td>0.99960</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1848</td>
<td>0.72 hr</td>
<td>0.99952</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2016</td>
<td>0.66 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>4</td>
<td>2016</td>
<td>0.66 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2184</td>
<td>0.615 hr</td>
<td>0.99931</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2352</td>
<td>0.57 hr</td>
<td>0.99919</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2520</td>
<td>0.53 hr</td>
<td>0.99904</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>3360</td>
<td>0.4 hr</td>
<td>0.99802</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1512</td>
<td>0.88 hr</td>
<td>0.999679</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>2</td>
<td>672</td>
<td>9,000</td>
<td>1 hr</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>2</td>
<td>840</td>
<td>9,000</td>
<td>0.8 hr</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>2</td>
<td>1008</td>
<td>9,000</td>
<td>0.66 hr</td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1512</td>
<td>0.88 hr</td>
<td>0.999679</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2016</td>
<td>0.66 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2184</td>
<td>0.615 hr</td>
<td>0.99931</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>3024</td>
<td>0.44 hr</td>
<td>0.99850</td>
</tr>
<tr>
<td>8</td>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1680</td>
<td>0.8 hr</td>
<td>0.99960</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1848</td>
<td>0.72 hr</td>
<td>0.99952</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2016</td>
<td>0.66 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2184</td>
<td>0.615 hr</td>
<td>0.99931</td>
</tr>
<tr>
<td>9</td>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1680</td>
<td>0.8 hr</td>
<td>0.99960</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1848</td>
<td>0.72 hr</td>
<td>0.99952</td>
</tr>
<tr>
<td></td>
<td>C</td>
<td>4</td>
<td>1344</td>
<td>12,800</td>
<td>1 hr</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>2</td>
<td>1176</td>
<td>9,400</td>
<td>0.57 hr</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1680</td>
<td>0.8 hr</td>
<td>0.99960</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2016</td>
<td>0.66 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2352</td>
<td>0.53 hr</td>
<td>0.99919</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>3528</td>
<td>0.4 hr</td>
<td>0.99803</td>
</tr>
</tbody>
</table>

Table 7.8 Five Optimal Strings found for each GA run
Table 7.9 Best Design Cost and Maintenance Requirements

<table>
<thead>
<tr>
<th>GA Run</th>
<th>Terminated at Generation</th>
<th>Termination Criteria</th>
<th>Cost</th>
<th>Average Maintenance</th>
<th>Qi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gen. 65</td>
<td>Type II</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td>2</td>
<td>Gen. 37</td>
<td>Type II</td>
<td>18,000</td>
<td>0.666 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td>3</td>
<td>Gen. 30</td>
<td>Type I</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td>4</td>
<td>Gen. 50</td>
<td>Type I</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td>5</td>
<td>Gen. 20</td>
<td>Type I</td>
<td>18,000</td>
<td>0.666 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td>6</td>
<td>Gen. 20</td>
<td>Type II</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td>7</td>
<td>Gen. 30</td>
<td>Type I</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td>8</td>
<td>Gen. 35</td>
<td>Type II</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td>9</td>
<td>Gen. 11</td>
<td>Type II</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td>10</td>
<td>Gen. 20</td>
<td>Type I</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
</tbody>
</table>

The benefit of retaining the five best strings identified by the GA can be highlighted by inspection of the results gained from run 6 in table 7.8. The third, fourth and fifth best strings show a saving in cost of 9,000 units, half the allowable expenditure. By using a design with only two pumps of type A the cost is reduced by half, yet the reliability is reduced by only 3.61x10^4. By using the GA optimisation scheme a design was identified that is 99% reliable and that provides a saving of 9,000 units. The reason behind such a large saving is achieved is due to the fact that the cost of the design relates only to the cost of the pumps. Halving the number of type A pumps in a design will obviously halve the cost of the design. Nevertheless it symbolises the usefulness of retaining more than one optimal solution. In addition to the saving in cost, a reduced amount of maintenance was also highlighted in the 4th and 5th best strings for run 6. This saving was again related to the decreased number of pumps present in the designs.

To inspect the convergence of the GA the average fitness value at each generation was monitored. Table 7.10 shows the initial population average fitness value and the final populations average fitness value for each of the ten runs. Inspection of this table illustrates the conversion of the GA as all runs tend to the maximum allowable fitness value of 62.906811. In order to further support the notion of convergence the average fitness at each generation for the runs 2, 5 and 8 were plotted, as shown in figure 7.11.
In summary it can be concluded that the optimisation of the deluge pump system was successfully achieved by implementation of a GA: indeed 80% of the trial runs gave the optimal string as their optimal solution. This result shows the effectiveness of the GA as a number of strings found in the exhaustive search differed only marginally in reliability terms. The mapping scheme devised for awarding a fitness value due to a string’s reliability was therefore adequate at distinguishing between strings that differed slightly in magnitude. However, the effectiveness of the mapping scheme diminished as the difference in the reliability of a group of strings decreased. The mapping scheme was not devised to detect differences in such strings, as in reality any design that was 99% reliable would be acceptable. A modification to the mapping scheme could have been developed in order to distinguish between strings in the range
0.99-0.99741. However, such a change was deemed unnecessary due to the successfulness of the GA.

By retaining the average fitness value at each generation the convergence of the GA was monitored. For each run performed the average fitness for the initial population and the final population were compared. For each run a trend existed such that the average fitness tended towards the maximum fitness value possible, 62.906811. It can therefore be concluded from such observations that the GA converged and hence was an effective optimisation procedure to use for the deluge pump system design problem.

The effectiveness of the GA can be further supported by inspection of the number of generations performed prior to convergence, table 7.9. The average number of generations for the 10 runs was 37, with the maximum of 65 generations found in run 1. The amount of computation required to perform 65 generations, for a population of 20 strings, was in the order of minutes. The GA therefore produced an optimal result with an acceptable amount of computational effort.

From a practical viewpoint the retention of the five best strings discovered by the GA would aid the designer. By having a choice from five optimal designs the most cost effective, yet adequately reliable design, can be highlighted and implemented. Inspection of the five best strings can also be used to support the effectiveness of the GA.

7.8.2 The Penalty Function Approach

Although the initial population contained twenty viable strings it was noted that future generations could produce infeasible designs which would be rejected therefore reducing the information utilised in producing the next generation. One way of avoiding rejecting encountered infeasible solutions is to allow the search to enter the infeasible region. A penalising strategy allows consideration of the infeasible region. Any string that violates the constraints is penalised which reduces the fitness value of the string and hence its chance of reproduction. In addition to this it is realistic to
assume that a designer would be prepared to exceed the original budget if the benefits
could be demonstrated by a cost effective improvement in the system’s performance.

In order to consider designs that violate the cost and maintenance constraints a new
fitness formula had to be developed. It was decided that a penalty should be added to
represent the degree to which the constraints were violated. Small violation incur
relatively small penalties and so will be accepted if the improvements in system
performance can compensate. Large violations are heavily penalised.

The determination of the fitness of each individual string was modified to incorporate
a value due to the reliability of the design coupled with a penalty due to exceeding any
of the constraints. The value due to the reliability of the design was achieved, as in
the feasible region search, using equation (7.5). In order to obtain a value due to
exceeding either constraints penalty formulas had to be developed. The penalty forms
were based on the desired improvement in system performance required to allow the
constraint violation to be acceptable.

7.8.2.1 The Cost Constraint

The maximum cost was set to 18,000 units for which a reasonably reliable system
would be expected. A reliability in the region of 0.965 would be reasonable for any
safety system. It was assumed that an increase in cost should be accompanied by a
similar increase in reliability to warrant exceeding the cost constraint. For this reason
it was decided that a 15% increase in cost should produce an increase in reliability
from 0.965 to 0.98. Inspection of the quadratic used to determine a fitness value,
figure 7.6, indicated that an increase from 0.965 to 0.98 was equivalent to an increase
of 15 units in the fitness figure. Hence for any design which exceeded the cost
constraint the following cost penalty value was obtained:

\[ \text{PenCost} = \left( \frac{(\text{Cost of Design} - 18,000)}{2700} \right) \times 15 \]

where 2700 is equal to 15% of 18,000 and 15 is the value obtained from the quadratic
formula.
7.8.2.2 The Maintenance Constraint

As a formula had already been devised to penalise excessive system cost it was decided that the maintenance penalty formula should also be related to cost. The maximum allowable maintenance time is an average of 1 hour per week. This is equivalent to 52 hours on average a year and if it costs 100 units an hour then the total cost per year would be 5200 units. For this amount of money a system with a reliability of 0.965 is again used to gauge the performance expected. As with the cost, an increase of maintenance activity would only be tolerated if the reliability itself was significantly increased. Therefore for any design which exceeded the maintenance constraint the following penalty value would be obtained:

$$\text{Pen}_{\text{Maint}} = \frac{(\text{Average maintenance each week} - 1.0 \times 5200.00)/780) \times 15}{\text{where } 780 \text{ is equal to } 15\% \text{ of } 5,200 \text{ and } 15 \text{ is the value obtained from the quadratic formula.}}$$

7.8.2.3 The Overall Fitness Value of a String

The fitness value for each string was therefore calculated using:

$$\text{Fitness} = \text{Fitness due to reliability} - \text{Pen}_{\text{cost}} - \text{Pen}_{\text{maint}}$$

Having shown that the GA produces better results when all initial strings in a population are valid, the subroutine Valid was incorporated in order to remove all infeasible designs. Due to the penalty formulas this included only the designs where there were not enough pumps to achieve 100% capacity.

The exhaustive search was completed again with the inclusion of the penalty forms however the optimal string still remained as:

<table>
<thead>
<tr>
<th>Pump Option</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of Pumps</td>
<td>4 (2 electric, 2 diesel)</td>
</tr>
<tr>
<td>Maintenance Interval</td>
<td>8 weeks</td>
</tr>
</tbody>
</table>

200
The reasoning behind this is that the optimal produced when not including the penalty formulas has a reliability of 0.999741. Increasing the maintenance from every 8 weeks to every 7 weeks incurs an additional cost of 1,859 units but only increases the reliability by $5.2 \times 10^{-5}$, not enough to warrant the additional cost. For this particular system the penalty formulas have little effect due to the extraordinary high reliability of designs which do not violate any constraint.

7.8.2.4 Results and Conclusions for Penalty Function Approach

The program was executed for 10 different seeds, using a crossover rate of 0.75, a mutation rate of 0.005 and the maximum number of generations to be 200. As with the feasible region search, the best five strings discovered by the GA were identified. Table 7.11 shows the design characteristics for the best five designs resulting from each run and table 7.12 depicts the performance measures for the overall best design and the GA.

The results showed that five out of the ten runs highlighted the best string to be the actual optimum gained from the exhaustive search, table 7.7. The remaining five strings highlighted best designs all with reliability values greater than 0.998. It can be seen that the GA produced very near optimal designs, even though it may have not highlighted the actual optimal. For this system a number of the designs proved to be highly reliable which may have affected the performance of the GA in locating the actual optimal. However, it cannot be disputed that the GA did succeed in locating a highly reliable area of the search space.

The benefit of retaining the five best strings identified by the GA can be highlighted by inspection of the results gained from run 7 in table 7.11. The fifth best strings shows that a small increase in cost, 800 units, produces a system design with a reliability of 0.99974. In the deluge system the increase is not significant as a design already exists which produces the system reliability of 0.99974 for 800 units less. This example, nevertheless, identifies the use of penalties in the discovery of designs in the infeasible region which may yield an advantage in reliability for a small additional cost.
<table>
<thead>
<tr>
<th>GA Run</th>
<th>TYPE</th>
<th>No. PUMPS</th>
<th>0</th>
<th>Cost</th>
<th>Maint</th>
<th>Qi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1512</td>
<td>18,000</td>
<td>0.88 hr</td>
<td>0.999679</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1680</td>
<td>18,000</td>
<td>0.8 hr</td>
<td>0.99960</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1848</td>
<td>18,000</td>
<td>0.72 hr</td>
<td>0.99952</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2688</td>
<td>18,000</td>
<td>0.5 hr</td>
<td>0.99888</td>
</tr>
<tr>
<td>2</td>
<td>A</td>
<td>4</td>
<td>2016</td>
<td>18,000</td>
<td>0.66 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2352</td>
<td>18,000</td>
<td>0.57 hr</td>
<td>0.99919</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2520</td>
<td>18,000</td>
<td>0.53 hr</td>
<td>0.99904</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>4</td>
<td>2016</td>
<td>18,800</td>
<td>0.66 hr</td>
<td>0.99944</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>4</td>
<td>2520</td>
<td>18,800</td>
<td>0.53 hr</td>
<td>0.999075</td>
</tr>
<tr>
<td>3</td>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1512</td>
<td>18,000</td>
<td>0.88 hr</td>
<td>0.999679</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1680</td>
<td>18,000</td>
<td>0.8 hr</td>
<td>0.99960</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1848</td>
<td>18,000</td>
<td>0.72 hr</td>
<td>0.99952</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2352</td>
<td>18,000</td>
<td>0.57 hr</td>
<td>0.99919</td>
</tr>
<tr>
<td>4</td>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1512</td>
<td>18,000</td>
<td>0.88 hr</td>
<td>0.999679</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1680</td>
<td>18,000</td>
<td>0.8 hr</td>
<td>0.99960</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1848</td>
<td>18,000</td>
<td>0.72 hr</td>
<td>0.99952</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2352</td>
<td>18,000</td>
<td>0.57 hr</td>
<td>0.99919</td>
</tr>
<tr>
<td>5</td>
<td>A</td>
<td>4</td>
<td>2016</td>
<td>18,000</td>
<td>0.66 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2184</td>
<td>18,000</td>
<td>0.615 hr</td>
<td>0.99931</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2352</td>
<td>18,000</td>
<td>0.57 hr</td>
<td>0.99919</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2520</td>
<td>18,000</td>
<td>0.53 hr</td>
<td>0.99904</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>3360</td>
<td>18,000</td>
<td>0.4 hr</td>
<td>0.99802</td>
</tr>
<tr>
<td>6</td>
<td>A</td>
<td>4</td>
<td>1680</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.99960</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>3360</td>
<td>18,000</td>
<td>0.4 hr</td>
<td>0.99802</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>4</td>
<td>1680</td>
<td>18,800</td>
<td>0.8 hr</td>
<td>0.99962</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>2</td>
<td>3528</td>
<td>18,000</td>
<td>0.38 hr</td>
<td>0.99498</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>3696</td>
<td>18,000</td>
<td>0.36 hr</td>
<td>0.99369</td>
</tr>
<tr>
<td>7</td>
<td>A</td>
<td>4</td>
<td>2016</td>
<td>18,000</td>
<td>0.8 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2184</td>
<td>18,000</td>
<td>0.615 hr</td>
<td>0.99931</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>3024</td>
<td>18,000</td>
<td>0.44 hr</td>
<td>0.99850</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>3192</td>
<td>18,000</td>
<td>0.42 hr</td>
<td>0.99827</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>4</td>
<td>1344</td>
<td>18,800</td>
<td>1 hr</td>
<td>0.99974</td>
</tr>
<tr>
<td>8</td>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1680</td>
<td>18,000</td>
<td>0.8 hr</td>
<td>0.99960</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1848</td>
<td>18,000</td>
<td>0.72 hr</td>
<td>0.99952</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2016</td>
<td>18,000</td>
<td>0.66 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2184</td>
<td>18,000</td>
<td>0.615 hr</td>
<td>0.99931</td>
</tr>
<tr>
<td>9</td>
<td>A</td>
<td>4</td>
<td>1344</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1680</td>
<td>18,000</td>
<td>0.8 hr</td>
<td>0.99960</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>1848</td>
<td>18,000</td>
<td>0.72 hr</td>
<td>0.99952</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>4</td>
<td>1344</td>
<td>18,800</td>
<td>1 hr</td>
<td>0.99974</td>
</tr>
<tr>
<td></td>
<td>B</td>
<td>4</td>
<td>1680</td>
<td>18,800</td>
<td>0.8 hr</td>
<td>0.99962</td>
</tr>
<tr>
<td>10</td>
<td>A</td>
<td>4</td>
<td>1680</td>
<td>18,000</td>
<td>0.8 hr</td>
<td>0.99960</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2016</td>
<td>18,000</td>
<td>0.66 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>2352</td>
<td>18,000</td>
<td>0.53 hr</td>
<td>0.99919</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>3024</td>
<td>18,000</td>
<td>0.44 hr</td>
<td>0.99850</td>
</tr>
<tr>
<td></td>
<td>A</td>
<td>4</td>
<td>3360</td>
<td>18,000</td>
<td>0.4</td>
<td>0.99802</td>
</tr>
</tbody>
</table>

Table 7.11 Five Optimal Strings found for each GA run
As with the feasible region search, the convergence of the GA can be illustrated by comparison of the initial population average fitness values and the final populations average fitness values for each run. Table 7.13 shows the initial and final populations average fitness value and confirms that convergence occurs in all ten runs.

<table>
<thead>
<tr>
<th>GA Run</th>
<th>Terminated at Generation</th>
<th>Termination Criteria</th>
<th>Cost</th>
<th>Average Maintenance</th>
<th>Qi</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Gen. 30</td>
<td>Type I</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td>2</td>
<td>Gen. 21</td>
<td>Type II</td>
<td>18,000</td>
<td>0.666 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td>3</td>
<td>Gen. 30</td>
<td>Type I</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td>4</td>
<td>Gen. 60</td>
<td>Type I</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td>5</td>
<td>Gen. 30</td>
<td>Type I</td>
<td>18,000</td>
<td>0.666 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td>6</td>
<td>Gen. 12</td>
<td>Type II</td>
<td>18,000</td>
<td>0.8 hr</td>
<td>0.99960</td>
</tr>
<tr>
<td>7</td>
<td>Gen. 46</td>
<td>Type II</td>
<td>18,000</td>
<td>0.666 hr</td>
<td>0.99942</td>
</tr>
<tr>
<td>8</td>
<td>Gen. 30</td>
<td>Type II</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td>9</td>
<td>Gen. 13</td>
<td>Type II</td>
<td>18,000</td>
<td>1 hr</td>
<td>0.999741</td>
</tr>
<tr>
<td>10</td>
<td>Gen. 24</td>
<td>Type II</td>
<td>18,000</td>
<td>0.8 hr</td>
<td>0.99960</td>
</tr>
</tbody>
</table>

Table 7.12 Best Design Cost and Maintenance Requirements

In summary it can be concluded that the inclusion of the penalty forms in no way hindered the effectiveness of the GA, even though only 50% of the trial runs gave the...
optimal string as their optimal solution. The remaining 50% of the runs highlighted
designs with reliability values exceeding 0.99 and therefore for each run a highly
reliable string was identified by the GA. The reason why the optimal string was only
given in 50% of the trials can be postulated, as with the feasible region search, as the
inability of the fitness value mapping scheme to differentiate between highly reliable
strings. The reason for a reduced percentage, in comparison to the feasible region
search, of the number of trial runs that identified the optimal string is due to a larger
search space being explored by the GA. In the penalty function approach previously
infeasible strings could have been included in the initial population. The GA could
then have converged prior to the optimal string being searched. This suggestion can
be supported by the type of termination criteria used in the 5 runs where the optimal
string was not identified. For 4 out of those 5 runs the GA terminated due to all the
strings in a population being identical, convergence criteria type II. This implies that,
in comparison to the other strings in the same population, one string was significantly
fitter and eventually dominated the entire population. As there exists a large number
of highly reliable strings in the deluge system any one identified by the GA would
have a high likelihood of resulting as the GA's optimal solution.

As with the feasible region search the average fitness value of the initial population
was compared to the average fitness value of the final population. The GA for each
trial run showed a trend implying convergence which suggests that the inclusion of the
penalty formulas did not disturb the effectiveness of the GA.

The average number of generations for the 10 runs was 29, slightly less than that
observed for the feasible region search. The reason for this reduction is again a result
of a larger search space being explored. For the feasible region search there existed
122 valid design options. For the penalty function approach there existed 494. The
feasible region search, therefore, had a greater chance of the initial population
containing a large number of highly fit strings. Alternatively, the penalty function
case had a high chance of the initial population containing only a few highly fit
strings. Any population dominated, in terms of fitness, by a few strings will converge
rapidly to the area of the search space occupied by those strings, hence reducing the number of generations required.

The motivation for the inclusion of penalty formulas in the GA is to allow a larger search space to be explored in the hope of identifying a better solution to the problem and to better mimic the real decision making process. In the deluge pump system more reliable solutions were identified. However, due to the exceptionally reliable optimal string in the feasible region of the search, these were heavily penalised. Although the inclusion of a penalised fitness value did not change the result of the GA for the deluge system, it can be a powerful tool in the optimisation of a design problem.

7.9 Summary of Conclusions

1) The use of a simple Genetic Algorithm provided an effective means of determining an optimal design.

2) By using penalised formulas more reliable designs were considered, although the effect was minimal for the deluge system. For systems which do not comprise of such a high percentage of extremely reliable designs in the feasible search space the effects would have been more prominent.

3) The Genetic Algorithm approach was attached to the combined model and tested effectively and successfully on a deluge pump system.
CHAPTER 8

APPLICATION OF THE CAUSE-CONSEQUENCE DIAGRAM METHOD TO STATIC SYSTEMS

8.1 Introduction

Following the literature review on the cause-consequence diagram method an investigation into the technique and possible developments was initiated. The fundamental fault with the cause-consequence diagram method was highlighted as its non-generalised set of construction and quantification rules. An investigation was hence initiated into the development of a generalised set of construction and analysis rules for the cause-consequence diagram method.

This chapter highlights the developments made in the cause-consequence diagram method for systems containing independent failure events.

8.2 Application to Static Systems

Cause-consequence analysis is most frequently applied to dynamic systems where the system state changes with time. It is the changing state of the system during phases such as the start-up or shut down, which makes alternative methods difficult to apply. Systems which are considered as static, where the system state at an instant in time is important, are generally analysed using techniques such as FTA or Markov modelling. Prior to considering dynamic systems the capability of cause consequence analysis for static systems is investigated.

8.2.1 Symbols used for Construction

The basic and most commonly used symbols for cause-consequence diagram construction in this section are illustrated in table 8.1.
8.2.2 Rules for Construction

The cause-consequence diagram technique has been applied to a static system and shown to yield the same results as those produced by the solution of the equivalent fault tree. On the basis of this study general rules have been devised for the correct construction of the cause-consequence diagram given a static system. The use of the cause-consequence method in this manner has significant implications in terms of efficiency of the reliability analysis and can be shown to have benefits for static systems. The algorithm for static system analysis is as follows:

**Step 1 Component Failure Event Ordering**

If order of failure is irrelevant, which is the case in a static system, then the cause-consequence diagram can be initiated by considering any of the components in the system. The analysis of the cause-consequence diagram should yield identical results regardless of the component or variable ordering, however the actual diagrams may vary in size. The first step of the cause-consequence diagram construction is therefore deciding on the order in which component failure events are to be taken. To ensure a logical development of the causes of the system failure mode it was decided that the
ordering should follow the temporal action of the system, for example the systems activation for the function required.

**Step 2 Cause-Consequence Diagram Construction**

The second stage involves the actual construction of the diagram. Starting from the initiating component the functionality of each component or sub-system is investigated and the consequences of these sequences determined. If the decision box is governed by a sub-system then the probability of failure will be obtained via a fault tree diagram.

**Step 3 Reduction**

If any decision boxes are deemed irrelevant, for example the boxes attached to the NO and YES branches are identical and their outcomes and consequences are the same, then these should be removed and the diagram reduced to a minimal form. Removal of these boxes will in no way affect the end result. This is illustrated in figure 8.1 where failure (F) occurs due to either of the two paths that terminate in the failure consequence. On one path the component A works, on the other it fails proving that the state of component A represented by the decision box is irrelevant.

![Figure 8.1 Redundant Decision Box](image)

Similarly figure 8.2 illustrates another example of this situation.
When a redundant decision box is identified, reduction is achieved by removing the box and entering the next decision/consequence box encountered in its place. Each decision box is inspected and when no further redundancies exist the cause-consequence diagram is deemed minimal.

**Step 4 System Failure Quantification**

The probability of each consequence for a static system is determined by summing the probability of each set of events which lead to this particular outcome. Each sequence probability is obtained by simply multiplying the probabilities of the component events represented by the branch, as illustrated in Hickling (ref. 43). This is possible as each sequence of events is mutually exclusive (ref. 14) and the probability of component failure events are assumed independent. The 4 step procedure can be represented in a flowchart as shown in figure 8.3.
8.3 Example 1: Three Component System

The cause-consequence diagram approach for static systems is demonstrated by application to a very simple system example. In the approach it is shown why the method has potential advantages in comparison to a conventional fault tree study for larger systems. The example system contains three components A, B and C and system failure is caused by either A and B failing together or C failing alone. The system failure causes are illustrated as a fault tree structure in figure 8.4.

![Figure 8.4 Example Fault Tree](image)

The cause-consequence diagram was constructed and analysed using the algorithm developed.

**Steps 1 and 2 Component Failure Event Ordering and Cause-Consequence Diagram Construction**

The ordering chosen was that of A, B, C and the cause-consequence diagram was constructed by inspecting the failures of those components in that order (Figure 8.5).

**Step 3 Reduction**

Boxes 3 and 4 are both irrelevant and were therefore removed. This process reduced the cause-consequence diagram, the final form being illustrated in figure 8.6, and as no further redundancies existed the diagram was minimal.
Step 4 System Failure Quantification

The probability of system failure is equal to the sum of the probability of the 3 sequence paths that lead to the consequence 'F'. Therefore since the paths are mutually exclusive:

\[
\text{Probability of Failure} = P(\text{Path 1}) + P(\text{Path 2}) + P(\text{Path 4})
\]

\[
= q_A q_B + q_A(1-q_B)q_C + (1-q_A)q_C
\]

\[
= q_A q_B + q_A q_C - q_A q_B q_C + q_C - q_A q_C
\]

\[
= q_A q_B + q_C - q_A q_B q_C
\]

Figure 8.5 Cause-Consequence Diagram for three component system

Figure 8.6 Reduced Cause-Consequence Diagram
The fault tree quantification, using the exact method, calculated the top event probability to be identical to that obtained by the cause-consequence diagram approach. After studying the reduced form of the cause-consequence diagram it was noted that it was equivalent to the Binary Decision Diagram (BDD) for the fault tree in figure 8.4, with the variable ordering A<B<C (Figure 8.7). The top event probability can also be obtained directly from the BDD by multiplying the probabilities down the paths that lead to the terminal 1 node (ref. 61).

8.3.1 Repeated Events

If the four stage procedure developed to construct and analyse a cause-consequence diagram is to be considered as a generally applicable approach it must be capable of dealing with the events which occur more than once in the fault tree diagram. This section shows that the cause-consequence diagram method can deal with repeated events in a more efficient way to that used for FTA. Using the cause-consequence diagram method there exists no need to obtain the Boolean expression of the top event and then manipulate it to produce a minimal form prior to analysis. The cause-consequence method deals with sequences of events which either occur (fail) or not occur (work). The probability of a particular outcome is obtained by summation of the probability of all paths that lead to the outcome.
8.3.1.1 Example System with Repeated Events

The system investigated was taken from (ref. 43) and included a RELAY as the repeated event. The closure of a valve can either be achieved by automatic control or by an operator manually closing the valve, the Relay is common to both these subsystems. The fault tree for this sub-system is illustrated in figure 8.8.

![Fault Tree for Valve Closure](image)

Using a bottom-up approach and Boolean reduction the minimal cut sets for the top event, failure to close Valve, were found to be:

\[
\{\text{OP.DETECT}\} \\
\{\text{RELAY}\}
\]

The exact top event probability is given by:

\[
P(\text{TOP}) = q_{\text{OP-DET}} - q_{\text{REL}} - q_{\text{OP-DET}}q_{\text{REL}}
\]

The cause-consequence diagram can be drawn in a number of ways depending on the component ordering chosen, however as this system is a static system ordering was irrelevant to the analysis and the diagram was constructed and analysed using the previously outlined steps.
Step 1 Component Failure Event Ordering
The ordering decision was made by considering the temporal patterns of the system. With this system the automatic control would be activated primarily followed by the activation, if necessary, of the manual valve operation. For this reason the ordering was chosen to be Automatic control, Operator control.

Step 2 Cause-Consequence Diagram Construction
The cause-consequence diagram was constructed by considering the effect of each sub-system working or failing (Figure 8.9).

![Cause-Consequence Diagram](image)

Figure 8.9 Cause-Consequence Diagram for variable ordering Automatic Control, Operator Control

Step 3 Reduction
Decision box 3 was identified as being irrelevant as the consequences attached to both outlet branches were identical. The removal of box 3 reduced the cause-consequence diagram to its minimal form, depicted in figure 8.10.

![Reduced Cause-Consequence Diagram](image)

Figure 8.10 Reduced Cause-Consequence Diagram
Step 4 System Failure Quantification

The probability of the valve not being closed is equal to the probability of the single path that leads to the consequence 'VO'. By inspection of Ft1 and Ft2, which represent the failure of the automatic loop and the operator loop respectively, it was noted that both trees contained the failure of the Relay (Figure 8.11). Applying the traditional method of multiplication of each decision box probabilities would prove to be incorrect as the probabilities are not independent since the failure of the Relay appears in both. To overcome this problem each component could be considered separately so the ordering would become DETECTOR, RELAY, OPERATOR and the corresponding cause-consequence diagram constructed, as shown in figure 8.12. This procedure is straightforward for such a small system but would quickly prove to be highly impractical for any real life system which could potentially contain hundreds of basic events. An alternative method had to be devised in order to deal with the repeated events without considering each basic event failure separately.
As the failure of the Relay appears in both fault trees then both sub-systems would fail if the Relay itself failed. It was therefore decided to extract the dependency by considering the repeated event separately. The repeated event was extracted and placed in a decision box preceding the first decision box that contained the common failure event. The original cause-consequence diagram, figure 8.10, was then duplicated on each outlet branch stemming from this new decision box as illustrated in figure 8.13. The result of this process was that the fault trees governing the failure of the automatic loop and the operator loop had to be modified by setting the failure of the Relay to TRUE or FALSE depending on whether the NO or YES paths, respectively, were followed.

By setting the probability of failure of the Relay to 1, in the fault trees representing events below the NO outlet branch from the 'Relay functions' decision box, the top event probability of Ft1 and Ft2 became unity. Setting the probability of failure for the Relay to 0, for the fault trees representing the event below the YES outlet branch, lead to the top event probability of Ft1 and Ft2 becoming single failure fault trees, where Ft1 became qDET and Ft2 became equivalent to qOP. The decision boxes that had a probability of 1 on the NO branch were removed as failure was certain and the reduced cause-consequence diagram was developed, as shown in figure 8.14.

Figure 8.13 Extracted Common Failure Event
The probability of the valve not closing could now be calculated as all failures were independent. The probability is equal to the summation of the probability of ending in the consequence 'VO', which was achieved via two mutually exclusive paths, path 1 and path 2. Hence,

$$\text{Prob(failure to close valve) } = P(\text{Path 1}) + P(\text{Path 2})$$

$$= q_{\text{REL}} + (1-q_{\text{REL}}) \cdot q_{\text{DET}} \cdot q_{\text{OP}}$$

$$= q_{\text{REL}} + q_{\text{DET}} \cdot q_{\text{OP}} - q_{\text{DET}} \cdot q_{\text{OP}} \cdot q_{\text{REL}}$$

This is the same probability as that obtained from the FTA technique. As with the previous examples it was shown that the cause-consequence diagram with ordering automatic loop, operator loop following extraction of the common failure event, was equivalent to the BDD with variable ordering $\text{REL}<\text{DET}<\text{OP}$ (Figure 8.15).

In the circumstances where several such dependencies exist the technique needs to be applied for each event. By extracting the common failure events, duplicating the cause-consequence diagram on each outlet branch of the new decision box and using TRUE and FALSE failure logic more than one common failure event can be dealt with.
8.4 Algorithm for Restructuring Cause-Consequence Diagram With Common Failure Events

In order to be able to deal with a common failure event a way of identifying the existence of such an event within the cause-consequence diagram was required. The names of two basic event symbols used in any fault tree will be the same if and only if the events they refer to are identical. It was therefore assumed that if a basic event had the same label as another then these were deemed the same event. Given the cause-consequence diagram each fault tree structure can be scanned in turn and any identical basic events, on the same path, identified. The common failure event must exist on the same path for it to cause quantification problems, for this reason every path out of each decision box was scanned starting from the top decision box, box 1.

8.4.1 Scanning the Cause-Consequence Diagram

Initially the cause-consequence diagram is stored using three arrays, IFAIL, IWORK and Q, where IFAIL(i) contains the label of the event attached to the NO outlet branch of decision box i, IWORK contains the label of the event attached to the YES outlet branch of the decision box and Q holds the failure logic of the decision box. For identification purposes the decision boxes were assigned numbers ranging from 0 to 99 and the consequence boxes were assigned numbers from -1 to -99. A simple two box cause-consequence diagram is illustrated in figure 8.16 with corresponding fault
tree structures depicted in figure 8.17. Taking the simple cause-consequence diagram illustrated in figure 8.16 the following arrays are created:

\[
\text{IFAIL} = \begin{pmatrix} 2 \\ -1 \end{pmatrix} \quad \text{IWORK} = \begin{pmatrix} -3 \\ -2 \end{pmatrix} \quad Q = \begin{pmatrix} \text{Ft1} \\ \text{Ft2} \end{pmatrix}
\]

Figure 8.16 A Simple Cause-Consequence Diagram

Figure 8.17 Fault Tree Structures for Cause-Consequence Diagram in figure 8.16

These arrays are used to scan the cause-consequence structure by implementing an algorithm similar to that utilised for reliability block diagram minimal path identification (ref. 6). Three arrays are used to store information required for path identification, IPOINT, IPREV and BRANCH. IPOINT stores information about the next event in the path, IPREV stores information about the previous event in the path and BRANCH indicates whether the NO or YES branch has been followed. These three arrays allow a path to be traced forward through the diagram. The tracing for a particular path stops when a consequence is reached. Once a path has been identified the decision boxes encountered in the path, stored in IPOINT, are placed in the array
IPATH. Using the array BRANCH the number representing the decision box in the array IPATH is set to either a 1, if the NO branch is traced through, or a 2, if the YES branch has been traced though. Once the path has been identified the array IPATH is used to determine if any common failure events are in the cause-consequence diagram. If no common failure events are present then the next path is scanned by backtracking using IPREV and BRANCH.

Figure 8.18 represents the flowchart of the above procedure and figure 8.16 can be used to illustrate its application.
\[ IFAIL = \begin{pmatrix} 2 \\ -1 \end{pmatrix} \quad IWORK = \begin{pmatrix} -3 \\ -2 \end{pmatrix} \quad Q = \begin{pmatrix} F_{\text{f1}} \\ F_{\text{f2}} \end{pmatrix} \]

NBOX is the number of the decision box presently under investigation, initialised as 1.

**NBOX = 1**

\[ \text{BRANCH}(1) = 0, \therefore \text{BRANCH}(1) = 1 \]
\[ \text{ICOL} = \text{IFAIL}(1) = 2 \]
\[ \text{IPOINT}(1) = 2, \text{IPREV}(2) = 1, \text{NBOX} = 2 \]

**NBOX = 2**

\[ \text{BRANCH}(2) = 0, \therefore \text{BRANCH}(2) = 1 \]
\[ \text{ICOL} = \text{IFAIL}(2) = -1 \]
\[ \text{IPOINT} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \text{IPREV} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \text{BRANCH} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \]

IPOINT starts at decision box 1 and moves to decision box 2, but 2 does not connect to another box so \[\text{IPATH} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \], as both decision boxes have \(\text{BRANCH}(i) = 1\). The path is then checked for any common failure events.

The next path is checked:

**NBOX = 2**

\[ \text{BRANCH}(2) = 1, \therefore \text{BRANCH}(2) = 2 \]
\[ \text{ICOL} = \text{IWORK}(2) = -2 \]
\[ \text{IPOINT} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \text{IPREV} = \begin{pmatrix} 0 \\ 1 \end{pmatrix}, \text{BRANCH} = \begin{pmatrix} 1 \\ 2 \end{pmatrix}, \text{IPATH} = \begin{pmatrix} 1 \\ 2 \end{pmatrix} \]

Path is checked for common failure events.

**NBOX = 2 AND BRANCH(2) = 2**

**Backtracking:**

\[ \text{ITEMP} = 1, \text{IPREV}(2) = 0, \text{IPOINT}(2) = 0, \text{BRANCH}(2) = 0, \text{NBOX} = 1 \]
\[ \text{IPOINT} = \begin{pmatrix} 2 \\ 0 \end{pmatrix}, \text{IPREV} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}, \text{BRANCH} = \begin{pmatrix} 1 \\ 0 \end{pmatrix} \]

**NBOX = 1**

\[ \text{BRANCH}(1) = 1, \therefore \text{BRANCH}(1) = 2 \]
\[ \text{ICOL} = \text{IWORK}(1) = -3 \]

\[ \text{IPATH}(2) \]

Path is checked for common failure events.

**Backtracking:**

\[ \text{ITEMP} = 1, \text{IPREV}(2) = 0, \text{IPOINT}(2) = 0, \text{BRANCH}(2) = 0, \text{NBOX} = 1 \]

**NBOX = 1**  **BRANCH(1) = 2**  **STOP ALL PATHS SCANNED**
It should be noted that only one path is considered at a time since after each extraction, if a common failure event is present, the structure of the cause-consequence diagram will change.

8.4.2 Common Failure Event Identification and Extraction

Once a path has been identified each fault tree on the path is compared to determine if any common failure events are present. Any event with the same label appearing in one or more fault trees is considered a common failure event. If a common failure event is discovered then the fault trees that it is part of are stored in the array FTREE. This process is completed so that following extraction the relevant fault trees can be identified and modified. A '1' is placed in the array element that is equal to the fault tree number, for example if Ft3 contains the common failure event then FTREE(3) is set equal to 1.

Given the scenario where more than one common failure event is encountered on the same path it was decided that the primarily extracted event should be the one which is present in the largest number of decision boxes on the path. If no difference occurs in the number of decision boxes that the common failure events are present in then the event found in the highest decision box in the cause-consequence diagram will be extracted first.

Following the identification of which common failure event to tackle the event is extracted from the fault trees and set as a new decision box event at the highest point in the cause-consequence diagram which has all dependencies below it. The new decision box, NEW_BOX, is set equal to TOT_BOX +1, where TOT_BOX represents the total number of decision boxes in the cause-consequence diagram. The probability of failure of the common failure event is stored in Q(NEW_BOX). The cause-consequence diagram is then duplicated on each branch starting from the new decision box.

The decision boxes stemming from the NO branch of the new decision box retain their original numbering and IPATH is used to identify which decision boxes in the cause-
consequence diagram to use. The duplicated diagram attached to the YES branch of the new decision box, however, have all decision boxes renumbered starting from Tot-Box and IPATH is again used to aid in the development of the duplicated decision boxes and corresponding fault trees. The fault tree structures attached to the renumbered decision boxes are also renumbered starting from the last fault tree number and stored in the array Q. A '2' is placed in FTREE to symbolise that the fault trees containing the common failure event are on the path stemming from the YES branch of the new decision box. This information is used at a later stage when modifying the fault trees which contain the common failure event. The procedure of extracting a common cause event from a cause-consequence diagram can be illustrated as a flowchart (Figure 8.19). The simple cause-consequence diagram illustrated in figure 8.16 and the fault trees Ft1 and Ft2 depicted in figure 8.17 can be used to demonstrate the extraction process:

\[
\begin{align*}
\text{TOT\_BOX} &= 2, \text{LAST\_FT} = 2, \text{IPATH} = \begin{pmatrix} 1 \\ 1 \end{pmatrix} \\
\text{Any Common Failure Events Present?} \\
\text{IPATH indicates that the first path follows the NO branch of decision box 1 and the NO branch of decision box 2. The fault tree structures governing these two boxes are Ft1 and Ft2. By inspection of the Ft1 and Ft2 it can be seen that a common failure event is present, that of component A. Ft1 and Ft2 are recorded in FTREE, where a 1 in the element FTREE(i) indicates that Fti contains the common failure event and is on the path attached to the NO branch of the new decision box TOT\_BOX = 3, NEW\_BOX = 3 IPATH is traced and ICOL is set equal to the position of the first non-zero term in the array. ICOL = 1, Q(3) = QA Duplicate diagram below new decision box, \therefore IFAIL(3) = IFAIL (ICOL)=1 Renumber decision boxes on YES branch, IWORK(3) = 4 \therefore IFAIL(4) = 5, IWORK(4) = -3, Q(4) = Ft3, FTREE(3) = 2 and IFAIL(5) = -1, IWORK(5) = -2, Q(5) = Ft4, FTREE(4) =2 \\
\begin{pmatrix} 2 \\ -1 \\ 5 \\ -1 \end{pmatrix} & \begin{pmatrix} -3 \\ -2 \\ -3 \\ -2 \end{pmatrix} & \begin{pmatrix} F1 \\ F2 \\ QA \\ F3 \\ F4 \end{pmatrix} & \begin{pmatrix} 1 \\ 2 \\ 2 \end{pmatrix}
\end{align*}
\]
Set number of boxes=TOT_BOX
Number of fault trees=LAST_FT

CFE = Common Failure Event

Compare fault trees stored in IPATH
Any CFE present

Yes

Store which fault trees contain the CFE in FTREE

TOT_BOX=TOT_BOX +1
NEW_BOX=TOT_BOX

Scan IPATH for first non-zero term, ICOL=1st non-zero term

Extract CFE and place in new decision box, NEW_BOX

Attach single component failure to NO branch for CFE
Q(NEW_BOX)=Q_{det}

Duplicate the CCD on each branch. Renumber boxes stemming from YES branch starting from TOT_BOX+1

Place a 'Z' in FTREE(i), where i equals the number of the fault tree

Remember fault trees attached to decision boxes of duplicated diagram stemming from YES branch starting from LAST_FT+1

Figure 8.19 Flowchart for Extraction of Common Cause Events
The cause-consequence diagram changes structure and becomes:

![Diagram](image)

### 8.4.3 Removal of Common Failure Event from Fault Tree Structures

Having developed a single decision box for the common failure event the decision boxes that contained the event prior to extraction need updating depending on whether the common failure event functions or fails to function. The common failure event is set to 1 (TRUE) in fault trees following the NO branch from the new decision box, as this indicates failure, and 0 (FALSE) in the fault trees following the YES branch to signify the component functioning. This process is achieved by utilising the array FTREE. FTREE is scanned and whenever a '1' is encountered the common failure event in the corresponding fault tree is replaced with the Boolean variable '1'. Similarly when a '2' is encountered the common failure event is replaced with a '0' in the fault tree structure.

The structure of each fault tree that contained the common failure event prior to the extraction process will have changed. Hence after every extraction each fault tree structure, which has been modified, will be required to be reorganised in order to remove all TRUE and FALSE variables. The total number of elements in FTREE are counted in order to ensure that only the fault trees originally containing the common failure event are examined. The total number of fault trees in FTREE is stored in the variable TOT_TREE. FTREE is then scanned and the first non-zero term found.
ITREE is set equal to the position of this element and the counter, NCNT, is incremented by 1. The fault tree is inspected and, depending on the Boolean variable encountered, is dealt with accordingly. The same tree is scanned until no further Boolean variables exist and the process is completed when all fault trees containing the common cause failure event have been modified.

Therefore in order to eliminate all TRUE and FALSE variables the algorithm shown in figure 8.20 is implemented:

![Algorithm for removing Boolean Variables from Fault Tree Structures](image)

Figure 8.20 Algorithm for removing Boolean Variables from Fault Tree Structures
The cause-consequence diagram represented in figure 8.16 and previously used to demonstrate the scanning and extraction process can also be used to illustrate the removal of the common failure events from the relevant fault trees. Following extraction the cause-consequence diagram takes on the form illustrated in figure 8.21, which in matrix form is as follows:

\[
\begin{pmatrix}
2 \\ -1 \\ 1 \\ 5 \\ -1
\end{pmatrix}
\begin{pmatrix}
-3 \\ -2 \\ 4 \\ -3 \\ -2
\end{pmatrix}
= \begin{pmatrix}
F_{t1} \\
F_{t2} \\
q_A \\
F_{t3} \\
F_{t4}
\end{pmatrix}
\]

The fault trees, \(F_{t1}-F_{t4}\), are depicted in figure 8.22.

Figure 8.21 Cause-Consequence Diagram following extraction of component A

Figure 8.22 Fault Trees for Cause-Consequence Diagram in figure 8.21
NCNT = 0, TOT_TREE = 4
Scan FTREE for next non-zero term = FTREE(1)
ITREE = 1, : inspect Ft1, NCNT = 1
Boolean Number = 1 TOP = TRUE
ITREE = 1
Next non-zero term in FTREE = 2 : inspect Ft2, NCNT = 2
Boolean Number = 1 TOP = TRUE
ITREE = 2
Next non-zero term in FTREE = 3 :: inspect Ft3, NCNT = 3
Boolean Number = 0 Ft3 = 0
ITREE = 3
Next non-zero term in FTREE = 4, :: inspect Ft4, NCNT = 4
Boolean Number = 0, Ft4 = 0 ITREE = 4, NCNT = TOT_TREE :: STOP.

The cause-consequence diagram changes to the structure shown in figure 8.23.

Figure 8.23 Modified Cause-Consequence Diagram following Extraction of Component A

228
The probability of travelling down the NO branch of decision box 1 and 2 is unity and hence these boxes can be removed to produce the minimal form (Figure 8.24) of the cause-consequence diagram following the inspection of one path.

8.5 Algorithm for Construction and Analysis for a Static System

The algorithms developed thus far, figures 8.3, 8.18, 8.19 and 8.20, can be combined together to produce one algorithm which can construct, modify and analyse any static system. Having dealt with a particular path the overall structure of the cause-consequence diagram will have changed if any common failure events were present. By duplicating parts of the cause-consequence diagram onto new decision boxes the left-most path may include new common failure events. For this reason the search for additional common failure events is initiated from the top decision box of the cause-consequence diagram. Therefore given that a common failure event has been identified, extracted and the relevant fault trees modified, the search will begin again from Restart_Box, the top decision box after each loop of the algorithm. Initially Restart_Box will be set to 1 but following any decision box begin entered higher than decision box one, Restart_Box will be reset equal to the number of the new top decision box. The search is completed when all paths have been traced starting from the top decision box and no further common failure events exist. The cause-consequence diagram can then be analysed by multiplication of the probability of travelling down a particular path as all boxes will be caused by independent failures.
8.6 Application to an Industrial System

As an example, the technique has been applied to the simple high pressure protection system depicted in figure 8.25. The basic functions of the components present in the high pressure protection system are shown in table 8.2. The function of the system is to prevent the passage of a high-pressure surge. The high pressure originates from a production well and the equipment to be protected are vessels located downstream on the processing platform.

![Diagram](image)

Figure 8.25 High-Integrity Protection System (HIPS)

The first level of protection is the emergency shutdown (ESD) sub-system. This comprises of 3 pressure sensors, for which 2 out of 3 must indicate a high pressure to cause a trip. Three shutdown valves, a Master, a Wing and an ESD valve activate to trip. If a high pressure surge is detected then the ESD system acts to close the Master valve, the Wing valve and the ESD valve. To provide an additional level of protection a second sub-system is included, the high-integrity protection sub-system (HIPS).

This sub-system also comprises of 3 pressure sensors, 2 to trip, and 2 isolation valves labeled HIPSI and HIPS2. The HIPS works in an identical manner to the ESD but has independent pressure sensors. The pressure sensors for each sub-system feed information into a common computer. The fault tree structure for the HIPS system is depicted in figure 8.26.

A fault tree analysis was completed and the qualitative analysis of the HIPS system produced 1274 minimal cut sets. Using the failure data given for each component the top event probability was calculated using the minimal cut set upperbound approximation and shown to equal $2.216 \times 10^{-2}$. 

230
<table>
<thead>
<tr>
<th>Component</th>
<th>Function</th>
<th>Failure Modes</th>
<th>$\lambda$</th>
<th>Mean Repair Time</th>
<th>Maintenance Test Interval Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Master Valve</td>
<td>To stop high pressure surge passing through system</td>
<td>Valve fails open: VM</td>
<td>$1.14 \times 10^{-5}$</td>
<td>36.0</td>
<td>4360</td>
</tr>
<tr>
<td>Wing Valve</td>
<td>To stop high pressure surge passing through system</td>
<td>Valve fails open: VW</td>
<td>$1.14 \times 10^{-5}$</td>
<td>36.0</td>
<td>4360</td>
</tr>
<tr>
<td>ESD Valve</td>
<td>To stop high pressure surge passing through system</td>
<td>Valve fails open: VE</td>
<td>$5.44 \times 10^{-6}$</td>
<td>36.0</td>
<td>4360</td>
</tr>
<tr>
<td>HIPS1 Valve</td>
<td>To stop high pressure surge passing through system</td>
<td>Valve fails open: VH1</td>
<td>$5.44 \times 10^{-6}$</td>
<td>36.0</td>
<td>4360</td>
</tr>
<tr>
<td>HIPS2 Valve</td>
<td>To stop high pressure surge passing through system</td>
<td>Valve fails open: VH2</td>
<td>$5.44 \times 10^{-6}$</td>
<td>36.0</td>
<td>4360</td>
</tr>
<tr>
<td>Solenoid</td>
<td>To supply power to valves</td>
<td>Fails Energized: SM, SW, SE, SH1, SH2</td>
<td>$5.0 \times 10^{-6}$</td>
<td>36.0</td>
<td>4360</td>
</tr>
<tr>
<td>Relay Contacts</td>
<td>To supply power to solenoids</td>
<td>Fails Closed R1-R10</td>
<td>$0.23 \times 10^{-6}$</td>
<td>36.0</td>
<td>4360</td>
</tr>
<tr>
<td>Pressure Sensors</td>
<td>Indicates the level of pressure to the computer</td>
<td>Fails to record actual pressure: P1-P6</td>
<td>$1.5 \times 10^{-6}$</td>
<td>36.0</td>
<td>4360</td>
</tr>
<tr>
<td>Computer</td>
<td>Reads information sent from pressure sensors and acts to close appropriate valves</td>
<td>Fails to read or act on information: C</td>
<td>$1 \times 10^{-5}$</td>
<td>36.0</td>
<td>4360</td>
</tr>
</tbody>
</table>

Table 8.2 Component Functions for HIPS System
Figure 8.26 Fault Tree Structure for HIPS System
8.6.1 Cause-Consequences Diagram Construction and Extraction

The cause-consequence diagram was constructed and analysed following the algorithms given in figure 8.3, 8.18, 8.19 and 8.20.

Steps 1 and 2 Event Ordering and Cause-Consequence Diagram Construction

The ordering was based on the action of components which could perform the task required by the system i.e. Master Valve, Wing Valve, ESD Valve, HIPS1 Valve, HIPS2 Valve. The cause-consequence diagram was constructed by considering the functionality of each valve and their effect on the system. Following the removal of all redundant decision boxes the minimal cause-consequence structure was created (Figure 8.27). The fault trees developed for each decision box are illustrated in figure 8.28a and 8.28b.

![Cause-Consequence Diagram for HIPS System](image)

Figure 8.27 Cause-Consequence Diagram for HIPS System
Figure 8.28a Fault Trees for Cause-Consequence Diagram for ESD Sub-System
The next stage involved the scanning process, starting from decision box 1, the first path was found to be:

Decision Box 1 $\rightarrow$ Decision Box 2 $\rightarrow$ Decision Box 3 $\rightarrow$ Decision Box 4 $\rightarrow$ Decision Box 5 $\rightarrow$ HP

Having identified a path IPATH was utilised in the determination of any common failure events. TOT BOX and LAST FT were set equal to 5 and the fault trees corresponding to decision boxes 1, 2, 3, 4 and 5 were compared. By inspection of Ft1 to Ft5 it was noted that the computer failing to read or act on high pressure information was common to all five fault trees. In addition to this the failure of the ESD Pressure Sensors P1, P2, P3 (3 occurrences) and HIPS pressure sensors, P4, P5, P6 (2 occurrences) were also identified as being common failure events. However, as the failure of computer was common to more decision boxes it was extracted first.
The common failure event was extracted and placed in a new decision box preceding the first box containing the computer, decision box 1. The fault trees $F_{t1}$, $F_{t2}$, $F_{t3}$, $F_{t4}$ and $F_{t5}$ were stored in FTREE and TOT_BOX was incremented by 1, hence NEW_BOX was reset to equal 6. As the new decision box preceded the original top decision box, Restart_Box, was also reset equal to 6. The common failure event of the computer failing was placed in decision box 6 and the cause-consequence diagram, starting from decision box 1, was then duplicated on the NO and YES outlet branches. The decision boxes and relevant fault trees were renumbered following the YES outlet branch of decision box 6, and the fault tree numbers were stored in FTREE. Therefore, $\text{FTREE} = (1,1,1,1,1,2,2,2,2,2)^T$ and $\text{TOT_TREE} = 10$. The failure of the computer was set to TRUE following the NO branch (Figure 8.29a) and FALSE following the YES branch (Figure 8.29b).

The first non-zero term in FTREE was found in FTREE(1) and hence $F_{t1}$ was stored in the variable ITREE and inspected for any Boolean numbers. Figure 8.29a shows that the top event, Master Valve Fails open, is certain. This was due to $F_{t1}$ containing a ‘1’ attached to an OR gate and no AND gate in the remainder of the tree structure.

![Diagram of fault trees](image)

Figure 8.29a Fault Trees $F_{t1}$, $F_{t2}$ and $F_{t3}$ following the NO branch of the common failure event decision box
The procedure was repeated until all relevant fault trees contained only gates and basic events and the cause-consequence diagram changed to that described in the following diagram:
The cause-consequence diagram was then reduced by removing all redundant decision boxes, boxes 1-5 (Figure 8.30).

![Figure 8.30 Reduced Cause-Consequence Diagram following Computer Extraction](image)

The cycle was then re-started, with

IFAIL = (2,3,4,5,-1,-1,8,9,10,11,-1)\(^T\)

IWORK = (-2,-2,-2,-2,7,-2,-2,-2,-2,-2,-2)\(^T\)

Q=(Ft1,Ft2,Ft3,Ft4,Ft5,qc,Ft6,Ft7,Ft8,Ft9,Ft10)\(^T\)

scanning began from Restart_Box, i.e. Decision Box 6:

<table>
<thead>
<tr>
<th>Restart_Box</th>
<th>IPATH(^T)</th>
<th>TOT_BOX</th>
<th>LAST_FT</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>(0,0,0,0,0,2,1,1,1,1,1)(^T)</td>
<td>11</td>
<td>10</td>
</tr>
</tbody>
</table>

Comparison of fault trees 7,8,9,10 and 11 found the Common failure events P1, P2, P3, P4, P5 and P6. P1 was extracted as it was common to the highest decision box and found in 3 decision boxes on the path.

FTREE = (0,0,0,0,0,1,1,1,1,1,1)\(^T\)

TOT_BOX =12, NEW_BOX = 12

First non-zero term in IPATH = IPATH(7), therefore ICOL = 7 Q(12) = Qp1

FTREE = (0,0,0,0,0,1,1,1,1,1,2,2,2,2,2)\(^T\), TOT_TREE = 15
The relevant fault tree structures were modified and following the second extraction of a common failure event the cause-consequence diagram changed as shown in figure 8.31.

The cycle was again re-started with

$$ IFAIL = (2,3,4,5,-1,18,9,10,11,-1,7,14,15,16,17,-1)^T $$

$$ IWORK = (-2,-2,-2,-2,7,-2,-2,-2,-2,13,-2,-2,-2,-2,-2)^T $$

$$ Q=(Ft1,Ft2,Ft3,Ft4,Ft5,qC,Ft6,Ft7,Ft8,Ft9,Ft10,qp1,Ft11,Ft12,Ft13,Ft14, Ft15)^T $$

The scanning process started from decision box 6 and following completion of one full cycle of the algorithm the cause-consequence diagram had changed, as depicted in figure 8.32.
Figure 8.32 Cause-Consequence Diagram following the Extraction of a third Common Failure Event
The algorithm was repeated until all paths leading from the top decision box had been scanned and no further common failure events were present. The final cause-consequence diagram was developed via the extraction of seven individual common failure events, C, P1, P2, P3, P4, P5 and P6. The fault trees corresponding to the Master, Wing, ESD and HIPS valves were reduced to the form depicted in figure 8.33 and the final cause-consequence diagram is given in figure 8.34.

![Figure 8.33 Reduced Fault Tree Structures for the Master, Wing, ESD and HIPS Valves](image-url)
Figure 8.34 Minimal Cause-Consequence Diagram for the HIPS System
8.6.2 Analysis

The probability of a high pressure surge could now be obtained by summing the probabilities of ending in the consequence HP, which was reached via 37 independent paths. Therefore

\[ \text{Probability (High Pressure)} = \sum_{i=1}^{n} P(\text{Path } i) \]

Component failures on the safety system are unrevealed and tested and repaired on scheduled maintenance. Their failure probabilities are given by equation (7.2). The system unavailability was calculated to equal \(2.216 \times 10^{-2}\). The figure is identical to that produce by the FTA method. This result does not reflect poorly on the cause-consequence diagram method, it merely emphasizes the fact that this particular system can be failed by a single component, the computer. The remaining minimal cut sets are of order 4 or more and therefore have little effect on the overall system unavailability. For a system that contained a large number of small order minimal cut sets it can be stated that the cause-consequence diagram method would yield a more accurate result than that obtained via FTA.

8.7 Comparison to the Binary Decision Diagram

By inspection of the final cause-consequence diagram produced for the HIPS system it was noted that the structure was identical to the BDD structure of the system with the variable ordering COMMENTER<P1<P2<P3<MASTER VALVE<WING VALVE<ESD VALVE<P4<P5<P6<HIPS1 VALVE<HIPS2 VALVE (Figure 8.35).

It can be noted that some of the nodes actually represent independent sub-systems, such as the ESD valve and Master valve, which implies that modularization has occurred. Modularization is used to aid in the efficiency of quantifying large fault trees (ref. 63) and involves identifying and solving independent submodules in the fault tree structure. The cause-consequence diagram method completes partial modularization firstly due to the fault trees governing the decision boxes and secondly due to extraction of any repeated events which results in the fault trees attached to the decision boxes becoming independent sub-modules.
The top event probability was obtained directly from the BDD and was identical to that produced by the cause-consequence analysis.
8.8 Advantages of New Algorithm

The conventional technique of FTA has been shown to be inefficient when the tree itself is complex (ref. 47). If the fault tree has many minimal cut sets approximations are utilised, as determining the top event probability using equation (2.23) would require extensive calculations. The approximations used, however, rely on the basic events having a small likelihood of occurrence and when this condition is not satisfied numerical errors can be significant. One way of avoiding such errors is to use Shannon’s theorem (ref. 62). The structure function for the top event, \( \phi(x) \), is pivoted with respect to the most repeated variable using

\[
\phi(x) = X_i \phi(1_i, x) + X_i \phi(0_i, x)
\]

where \( X_i \) is the most repeated variable and

\[
\phi(1_i, x) = \phi(X_1, \ldots X_{i-1}, 1, X_{i+1}, \ldots X_n)
\]

\[
\phi(0_i, x) = \phi(X_1, \ldots X_{i-1}, 0, X_{i+1}, \ldots X_n)
\]

This process is repeated until all repeated events have been manipulated. The expectation of the structure function can then be taken yielding the top event probability.

The most attractive feature of the BDD method is that it is created from the fault tree structure into a format which encodes Shannon’s decomposition. Each variable is taken and the effect of it failing and functioning on the structure function is obtained. Shannon’s formula can be written as \( X_1 \phi_1 + X_\bar{1} \phi_2 \), if \( \phi \) is a Boolean function which is identical to the \( \text{ite} \) structure \( \text{ite}(X_1, \phi_1, \phi_2) \) where \( \phi_1 \) and \( \phi_2 \) are Boolean functions with \( X_1 = 1 \) and \( X_\bar{1} = 0 \) respectively. The exact failure probability can be determined directly from the BDD in a very efficient calculation procedure involving the summation of the probability of all paths that terminate in a 1.

One of the disadvantages with the BDD, as outlined in chapter 3, is that it cannot be obtained directly from the system description and is obtained via the fault tree
structure. During the conversion process the BDD loses all the causality information that is represented in the fault tree structure. As all the cause-consequence diagrams developed are identical to the BDD with the same variable ordering then there is no need to convert the fault tree of the system to the BDD structure, hence reducing the effort required with the conventional approach. As the BDD is a more efficient analysis tool than the fault tree method then so is the cause-consequence diagram method. By applying the cause-consequence technique directly to the system description the failure causality is not lost, making the method attractive.

The algorithm developed outlines clear rules for construction and analysis of any static system. By inspection of the algorithm created it was noted that it performs a process similar to Shannon's decomposition. The extraction process and duplication of the relevant part of the cause-consequence diagram is similar to the pivoting process used in Shannon decomposition. The difference, however, lies in the fact that only the boxes containing the common failure event are manipulated and not the whole structure function, which is the case with the fault tree method. Following the extraction of any repeated event the fault trees governing the decision boxes are independent which implies that a form of modularization is conducted.

8.9 Algorithm to Extract Independent Submodules

In the case of the HIPS system it may have proved beneficial to extract P1, P2 and P3 as a whole structure rather than extracting individual components. On such a small system the difference in efficiency would be minimal but for a larger system with several repeated submodules the savings could be large. An algorithm was therefore required which could deal with repeated independent submodules and their extraction.

8.9.1 Identification of a Common Failure Submodule

Analysis of large fault trees has been shown to be computationally expensive and many attempts have been made to develop techniques that reduce the size of the structure. One of these techniques is MODULARIZATION, which involves identification of independent subtrees that are then solved separately and the results placed back into the fault tree structure as new basic events. A module of a fault tree can be defined as a subtree whose terminal events, i.e. basic events, do not occur
elsewhere in the tree (ref. 48). It was decided that the identification process of these modules could be used in order to highlight independent submodules in each fault tree structure attached to any decision box. Following the identification of a sequence path on the cause-consequence diagram each submodule associated with the relevant fault tree structures on the path could be compared to establish if any identical submodules existed. The comparison should begin with the smallest submodules and then increase in size until no identical submodules can be found. By following this procedure the largest common submodule would be identified and extracted. For example Ft1, Ft2 and Ft3 below have a common subtree starting from gate G1. G2 and G3 are also common subtrees but not the largest independent module therefore the search would continue until G1 had been identified as being the largest common module.

8.9.2 Modularization of a Fault Tree

It was decided that the modularization algorithm described in section 6.4.1, and used to find independent subtrees in a fault tree structure containing independent and dependent failure events, would be used to identify common submodules in the cause-consequence diagram.
8.9.3 Identification and Extraction of Common Submodule

Initially the cause-consequence diagram is stored in the arrays IFAIL, IWORK and Q, as described in section 8.4.1. The failure logic for each decision box is held in Q and it is this array which is used to aid in the identification of any common submodules. Using algorithm 8.18 the cause-consequence diagram is traced and various paths identified. Following the identification of a path, IPATH is used to indicate which fault tree structures are present in the path. Each individual fault tree is then modularized using the algorithm outlined in section 6.4.1 and each submodule is compared to all other submodules in the same path. As soon as the largest submodule structure is discovered to be identical with another submodule in a different tree the comparison process stops. The submodule is then stored in the 1-d array SUB, where the i\textsuperscript{th} entry holds the i\textsuperscript{th} basic event in the independent submodule. Following the identification of a common submodule the fault trees that contain the module are stored in FTREE so that following extraction the relevant fault trees can be modified.

Following the identification of a common submodule the module is extracted from the fault tree structure and set as a new decision box at the highest point in the cause-consequence diagram which has all dependencies below it. The procedure is then identical to that developed in 8.4.2 for a single common failure event. If a path is scanned and no common submodule exists then each individual event is compared to determine whether there exists any common failure events. If a common failure event is identified then the extraction process outlined in 8.4.3 is followed, otherwise the next sequence path is scanned and each fault tree modularized and compared. The process continues until all paths have been traced and no dependencies remain in the diagram.

8.9.4 Application to the HIPS System

The common submodule algorithm can be applied to the HIPS system to illustrate two points. Firstly that the modularization technique is efficient and secondly that the technique reduces the size of the cause-consequence diagram required for analysis.
8.9.4.1 Cause-Consequence Diagram Construction and Minimisation

The cause-consequence diagram for the HIPS system is replicated in figure 8.36 with the relevant fault trees depicted in figure 8.37a and 8.37b.
Initially the cause-consequence diagram was stored as:

\[
\text{IFAIL} = \begin{pmatrix} 2 \\ 3 \\ 4 \\ 5 \\ -1 \end{pmatrix}, \quad \text{IWORK} = \begin{pmatrix} -2 \\ -2 \\ -2 \end{pmatrix}, \quad \text{Q} = \begin{pmatrix} \text{Ft1} \\ \text{Ft2} \\ \text{Ft3} \\ \text{Ft4} \\ \text{Ft5} \end{pmatrix}
\]

Starting the scanning process from decision box 1 the first path identified was:

Decision Box 1 \(\longrightarrow\) Decision Box 2 \(\longrightarrow\) Decision Box 3 \(\longrightarrow\) Decision Box 4
\(\longrightarrow\) Decision Box 5 \(\longrightarrow\) HP

Having identified a path, IPATH was used to determine which fault trees to modularize. Ft1, Ft2, Ft3, Ft4 and Ft5 were all modularized. The modularization of Ft1 is depicted in Figure 8.38 and shows that all gate nodes were found to be independent modules.
Following the identification of all the submodules they were compared and the first common submodule was highlighted to be G5 of Ft1. Gate G5 comprised of the pressure switches P1, P2 and P3 which were common to decision boxes 1, 2 and 3. The submodule was extracted and the diagram duplicated on both NO and YES outlet branches. The extracted submodule was set equal to unity in the fault trees it was present in following the NO outlet branch and zero in the fault tree structures attached to the YES outlet branch (figure 8.39). The redundant boxes were removed, Boxes 1-3, and the cause-consequence diagram reduced to the form shown in figure 8.40 with the fault trees Ft6-Ft11 depicted in figure 8.41.
Figure 8.39 Extraction of Common Pressure Sensor Submodule
Figure 8.40 Reduced Cause-Consequence Diagram for HIPS System
The cycle was then re-started with Restart _Box equal to 1 using the arrays:

\[
\text{IFAIL} = (2,3,4,5,-1,4,8,9,10,11,-1)^T
\]

\[
\text{IWORK} = (-2,-2,-2,-2,2,7,-2,-2,-2,-2)^T
\]

\[
\text{Q}=(Ft1,Ft2,Ft3,Ft4,Ft5,Ft6,Ft7,Ft8,Ft9,Ft10,Ft11)^T
\]

\[
\text{IPATH} = (0,0,0,1,1,1,0,0,0,0,0)^T
\]

IPATH was identified as IPATH = (0,0,0,1,1,1,0,0,0,0,0)^T and therefore fault trees 4,5 and 6 were modularized and compared for any common submodules. A common submodule was found in Ft4 and Ft5 and comprised of the three pressure sensors P4,P5,P6 and the computer logic C. The common submodule was extracted and the relevant part of the cause-consequence diagram duplicated on each outlet branch. Following the removal of all redundant decision boxes the cause-consequence diagram took on the form illustrated in figure 8.42.
Figure 8.42 Cause-Consequence Diagram following the Extraction of a Common Submodule

The scanning procedure was repeated and following the extraction, on the right-hand side of the diagram, of a common submodule containing P4, P5, P6 and the extraction of the single common failure event C (the computer failure), the cause-consequence diagram was reduced to the minimal and independent form which is depicted in figure 8.43.
Figure 8.43 Final Cause-Consequence Diagram for the HIPS System

8.9.4.2 Analysis

The probability of a high pressure surge could now be obtained by summing the probabilities of all independent paths which terminate in the consequence HP. The cause-consequence diagram for the HIPS system using the algorithm for extracting individual events only consisted of 37 such paths, section 8.6.2, but by using the modularization approach the number of paths to sum was reduced dramatically to only 5. The system unavailability was calculated to equal 2.216x10^{-2}, which was identical to that produced by the BDD for the HIPS system and the cause-consequence diagram analysed previously with the 37 failure sequences.
The effort that is required to complete the modularization process has been shown to be minimal (ref. 48) and results in extensive savings in both the size of the diagram and computation required to analyse it.

8.10 Summary and Conclusion

An algorithm has been developed that given any static system will produce the correct cause-consequence diagram and calculate the exact system failure probability. This is achieved without having to construct the fault tree of the system yet retains the failure logic of the system. The cause-consequence diagram is reduced to a minimal form by firstly removing any redundant decision boxes and secondly by manipulating any common failure events which exist on the same path. The common failure events can be extracted as common submodules or individual events. The extraction of common submodules has been shown to be computationally more advantageous than removing each event in the submodule separately.

The minimised cause-consequence diagram is then analysed using a BDD analysis procedure. The advantages of the cause-consequence diagram are two-fold: firstly the diagram can be constructed directly from the system description and secondly that any dependencies are dealt with on an individual basis. The effort that is required to convert a fault tree into a BDD is avoided.
CHAPTER 9

APPLICATION OF THE CAUSE-CONSEQUENCE DIAGRAM METHOD TO A MULTI-PHASE SYSTEM

9.1 Introduction

Cause-consequence analysis is most frequently applied to model systems during phases where the state of the system is changing with time. Systems that feature start-up and shut-down phases can be included in this category and are often difficult to analyse using traditional methods. In order to generalise the cause-consequence diagram method further, such that it can be applied to many types of systems, an investigation into the use of the technique with a multi-phased system was initiated.

9.2 System Description

The system chosen to be investigated, which contained a start-up and a shut-down sequence in addition to its operational phase, was a pressure tank system (ref. 64). The function of the system is to control the operation of a pump which transports fluid from a large reservoir into a tank. The system configuration is illustrated in figure 9.1 and the components individual functions and failure modes are represented in table 9.1.

Initially the system is considered to be in a dormant state and therefore de-energized. The switch S1, the relay contacts K1 and K2 are all open when in the dormant state and the timer and pressure switch contacts are closed. Depressing the switch S1 provides power to the coil of K1 which results in the closure of the K1 contacts. K1 self latches when S1 opens when released and power is also supplied to K2 resulting in K2 contacts closing which starts the pump motor. It is assumed that the tank takes 30 minutes to fill and once the pressure threshold is reached the pressure switch contacts open, de-energizing K2 which results in the removal of power from the pump motor. After a period of time the tank becomes empty and the pressure switch closes which energizes K2. The pump restarts and the filling process commence again. The
tank is filled twice daily and the system is inspected at 6 monthly intervals for dormant failures.

In the event of the pressure switch failing to open a safety feature is included in the form of the Timer relay. Power is applied to the timer relay following the closure of the K1 contacts which initiates a clock. If the clock registers 30 minutes of continuous pumping then the timer relay contacts are opened which results in a break in the circuit to K1 and system shutdown.

Figure 9.1 Pressure Tank System

For quantification purposes the probability of component failure was obtained via equation (9.1) for a revealed failure and equation (9.2) for an unrevealed failure.

\[ Q = 1 - e^{-\lambda t} \]  
\[ Q_{AV} = \lambda \left( \frac{\theta}{2} + \tau \right) \]
### Component Functions and Failure modes

<table>
<thead>
<tr>
<th>Component</th>
<th>Function</th>
<th>Failure Modes</th>
<th>Effect on System</th>
<th>Failure Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch, S1</td>
<td>To apply power to coil of K1 relay</td>
<td>S1C: Switch failed closed</td>
<td>Circuit remains energized but can be broken by K2</td>
<td>Unrevealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>S1O: Switch failed open</td>
<td>No Power to energize circuit</td>
<td>Revealed</td>
</tr>
<tr>
<td>Relay K1</td>
<td>Electrically self-latched applying power to relay of K2</td>
<td>K1D: Relay fails de-energized</td>
<td>No Power to Circuit</td>
<td>Revealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K1CC: Contact fails closed</td>
<td>Circuit remains energized but can be broken by K2</td>
<td>Unrevealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K1CO: Contact fails open</td>
<td>No Power to Circuit</td>
<td>Revealed</td>
</tr>
<tr>
<td>Relay K2</td>
<td>Delivers power to the Motor</td>
<td>K2D: Relay fails de-energized</td>
<td>No power to Motor</td>
<td>Revealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K2CC: Contact fails closed</td>
<td>Continuous power to motor</td>
<td>Revealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>K2CO: Contact fails open</td>
<td>No power to Motor</td>
<td>Revealed</td>
</tr>
<tr>
<td>Timer Relay (TIM)</td>
<td>Provides emergency shut-down in event of pressure switch failing</td>
<td>TIMCC: Timer contact fails closed</td>
<td>Circuit energized but PRSW can open</td>
<td>Unrevealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>TIMCO: Timer contact fails open</td>
<td>No power to motor</td>
<td>Revealed</td>
</tr>
<tr>
<td>Pressure Switch (PRSW)</td>
<td>De-energizes coil of K2 when tank is full</td>
<td>PSWC: Fails closed</td>
<td>Continuous power to motor</td>
<td>Revealed</td>
</tr>
<tr>
<td></td>
<td></td>
<td>PSWO: Fails Open</td>
<td>No power to motor</td>
<td>Revealed</td>
</tr>
<tr>
<td>Fuse</td>
<td>To prevent power surge</td>
<td>F: Fails Broken</td>
<td>No power to motor</td>
<td>Revealed</td>
</tr>
<tr>
<td>Power Supplies 1 &amp; 2</td>
<td>Supplies Power to Relays and Motor</td>
<td>PS1, PS2: No Power</td>
<td>No power to Motor</td>
<td>Revealed</td>
</tr>
<tr>
<td>Motor</td>
<td>Pumps fluid into tank</td>
<td>M: Fails Broken</td>
<td>No power to pump</td>
<td>Revealed</td>
</tr>
</tbody>
</table>

#### Table 9.1 Component Functions and Failure modes

### 9.3 Cause-Consequence Diagram Construction

The algorithm developed for the construction of a cause-consequence diagram for a system containing independent failures, outlined in chapter 8, was used to construct the cause-consequence diagram for the pressure tank system.

#### Step 1 Component Failure Event Ordering

The ordering of the components for the construction of the cause-consequence diagram was selected by considering the temporal patterns of the system. For the
pressure tank system the switch, S1, is depressed followed by it opening. K1 energizes and powers K2 which powers the pump. Following 30 minutes of operation the pressure switch should open. In the event that the pressure switch fails to open the timer should time out and the timer contacts opened. Given the pressure switch opens K2 contacts should de-energize, removing power from the pump. In the instance where the timer is required to break the circuit containing K1, K1 contacts should de-energize removing power from K2 which results in the removal of the power supply to the pump. The ordering was therefore chosen to be:

S1, K1, K2, PRESSURE SWITCH, TIMER RELAY, K1, K2

It can be seen that the components K1 and K2 both occur twice in the ordering sequence. This is the result of the system containing two different phases and hence some components perform different actions in each different phase. The components K1 and K2 are both required to close in the start-up sequence and open in the shut-down sequence.

Step 2 and 3 Cause-Consequence Diagram Construction and Reduction
The cause-consequence diagram was constructed by considering the affect of each component, in the chosen order, on system performance. In order to illustrate the control of the pressure switch on the system a feedback loop could have been attached to the end of the normal filling sequence. In order to highlight the features under inspection in this chapter, i.e. different system phases, only one filling sequence was investigated, the cause-consequence diagram of which is shown in figure 9.2. The corresponding fault trees are illustrated in figure 9.3.
Figure 9.2 Cause-Consequence Diagram for Pressure Tank System

Figure 9.3 Fault Trees for Pressure Tank Cause-Consequence Diagram

262
Step 4 System Failure Quantification

The probability of the tank becoming overpressurised is equal to the sum of the probabilities of the six paths that lead to the consequence labelled 'O'. By inspection of the cause-consequence diagram it was noted that dependencies existed that rendered simple techniques, which are used to obtain the product of each event probability to yield the path probability, inappropriate. Such dependency can be seen in the form of the common failure event 'Power supply 1' (PS1), which is present as a cause in both fault tree 7 (Ft7) and 8 (Ft8). This event was extracted following the previously developed algorithm, presented in section 8.4, and the relevant changes made to the cause-consequence diagram. The updated cause-consequence diagram is depicted in figure 9.4 and the modified fault trees in figure 9.5.

Figure 9.4 Cause-Consequence Diagram following the extraction of PS1
Further inspection of the cause-consequence diagram indicated that a new dependency was present. Components S1, K1, K2, PRSW and TIM in certain operational sequences are required to perform two different functions which if successfully accomplished results in the components residing in different states at different times. For example initially K2 contacts were required to close and then open. As the system is not in continuous use it is possible that the K2 contacts could fail closed between operations and hence be the cause of K2 being in the closed state at the beginning of the operational sequence. In this event the tank would overpressurise as a continuous supply of power would be supplied to the pump. This type of failure would not be incorporated in the cause-consequence diagram analysis outlined thus far. An alternative method therefore had to be developed in order to deal with such 'inconsistent failure events'. To illustrate the development of the method the simple cause-consequence diagram section shown in figure 9.6 can be utilised with the relevant fault trees depicted in figure 9.7. This diagram represents the extracted relevant dependency features from the main cause-consequence diagram.

Figure 9.6 Example Cause-Consequence Diagram

Figure 9.5 Modified Fault trees following PS1 extraction
The problem with this cause-consequence diagram is that the component K2 is required to perform two different functions, firstly to close, decision box 1, and then later in the sequence to open, decision box 3.

For decision box 1, 'K2 contacts close', the event of travelling down the YES outlet branch can be governed by two independent situations. Firstly K2 contacts can close due to the correct functioning of the system, i.e. fault tree 1 (Ft1) does not occur. This implies that K2 contacts do not fail open AND the event P1 does not occur. This situation results in a time implication for travelling down the NO outlet branch of decision box 3. The NO outlet branch of decision box 3 represents the situation where K2 contacts have failed closed and as K2 contacts functioned correctly at decision box 1, it implies that they could only have failed closed in the time it took the system to travel to decision box 3.

The second situation that can occur to cause the YES outlet branch to be followed from decision box 1 is that K2 contacts could have failed closed between two operations. This situation results in the fault tree representing the causes of decision box 3 to occur with certainty, arriving at the outcome system failure, F. The two situations described above and their respective affect on decision box 1 and 3 in the cause-consequence diagram illustrated in figure 9.6, is shown in figure 9.8.
To identify this potential problem it can be seen by inspection of the fault trees $F_{t1}$ and $F_{t3}$, which are two of the three trees identifying causes of the cause-consequence diagram paths from decision box 1-3, that they contain $K2$ failing open ($K2CO$) and $K2$ failing closed ($K2CC$) respectively. These are inconsistent failure modes and can be used to identify the situation described above. For the remainder of this section it will be assumed that the first failure event represents the decision box containing the first failure mode and the second failure event represents the decision box containing the second failure mode. For the example in figure 9.6, then, the first failure event is decision box 1, the first failure mode is $K2CO$, the second failure event is decision box 3 and the second failure mode is $K2CC$. The problem that evolves is that causes of the second failure event can be inconsistent with the causes of the first failure event. As only one failure mode is represented by each decision box fault tree, the problem is to determine how to represent a situation of the type described above.

An investigation was initiated into the identification of which failure mode, for the component with two different functions, caused the occurrence of the fault tree corresponding to the first failure event. It was discovered that depending on whether the second failure mode was a revealed or unrevealed failure the structure of the cause-consequence diagram was different.
For the second failure mode, which is a **revealed failure**, it was discovered that the YES branch of the first failure event could not be governed by the second failure mode, i.e. the second failure mode cannot occur between two operations. A revealed failure represents a component that when failed will cause an event which is immediately noticeable. For example, in the pressure tank system K2 contacts failing closed is a revealed failure as the tank will overpressurise due to a continuously operating pump. Therefore if, in the case of inconsistent failure modes, the second failure mode is a revealed failure then the action of the decision box containing the first failure mode cannot be caused by the second failure mode as this would be immediately noticeable. The decision box containing the first failure mode is therefore governed by the fault tree corresponding to the NO outlet branch of the decision box. For K2 then in figure 9.6, it was assumed that at decision box 1 if the event represented by the YES branch occurred then K2 contacts were considered to be functioning correctly, i.e. work closed, and the probability associated with the path was equal to 1-P(Ft1). This assumption resulted in the time implications discussed for the decision box containing the second failure mode later in the sequence, decision box 3, where the second failure event must have occurred in the time it took the system to perform the action represented by the event sequence leading to decision box 3. If on the otherhand the event represented by the NO branch of decision box 3 was caused by the component P3 the time to fail would not be the time it took the system to reach decision box 3 but the time associated with the failure of P3, which would be dependent on whether P3 was unrevealed or revealed and calculated using equation (9.2) and (9.1) respectively.

For the second failure mode which is an **unrevealed failure event**, a failure which is only revealed when a demand is made on the system (ref. 6) or the component inspected, the decision box containing the first failure mode could either be caused by the occurrence of the first failure mode, represented by the NO exit branch, or the occurrence of the second failure mode, represented by the YES exit branch. For example, taking the cause-consequence diagram in figure 9.6 and assuming that the event K2CC is an unrevealed failure event, decision box 1 could potentially have two different probabilities associated with the NO and YES branches depending solely on
the state of the component K2. If K2 contacts were functioning correctly then the event represented by the YES outlet branch for decision box 1 would be dependent on the fault tree originally associated with the first failure event, Ft1, and would be equal to 1-P(Ft1). If, on the other hand, K2 contacts had failed closed between system operations then the event represented by the YES outlet branch for decision box 1 would be equal to unity, and the probability of entering the start of the sequence would be QK2CC, indicating that K2 contacts had failed between operations (Figure 9.8). In order to inspect both eventualities in the cause-consequence diagram a new type of decision box was created, the EXISTENCE DECISION BOX (Figure 9.9).

The existence decision box contains the second failure mode and asks whether the component i exists in a particular state or not, generally in a failed state. Similar to the traditional decision box, the existence decision box has both YES and NO outlet branches and for distinction purposes the YES option was placed on the left-hand side of the oval shaped box.

![Figure 9.9 New Existence Decision Box](image)

The process of developing a new decision box to represent an event present in fault trees lower down in the cause-consequence structure was proceeded, as with the extraction process discussed in section 8.4, by duplication of the cause-consequence diagram on both outlet branches. The probability attached to the YES outlet branch of the existence box containing component i is equal to Q, and the probability of the NO outlet branch is equal to 1-Q.

Following the YES outlet branch stemming from the existence decision box indicated that the second failure mode existed and therefore the first failure mode probability could not exist and would be set equal to zero. In addition to this as the second failure
mode existed the probability of its occurrence found anywhere in the remainder of the sequence following the YES outlet branch of the existence box would be set equal to unity. Therefore for figure 9.6, assuming K2CC is an unrevealed failure event, the cause-consequence diagram illustrated in figure 9.10 would be created and reduced to the form shown in figure 9.11, where \( F_4 = F_1, F_5 = F_2 \) and \( F_6 = F_3 \).

![Figure 9.10 Modified Cause-Consequence Diagram for Inconsistent Failure Modes](image)

![Figure 9.11 Reduced Cause-Consequence Diagram for Inconsistent Failure Modes](image)

269
Following the NO outlet branch stemming from the existence decision box results in the same scenario as if the failure had in fact been a revealed failure. The decision box containing the first failure mode is governed by the original fault tree, which includes the first failure event. The same implications regarding the time to failure for the second failure mode used for a revealed failure event are employed. For K2CC in figure 9.11 the time to fail for the second failure mode, following the YES outlet branch from decision box 5, is set equal to the time it takes the system to travel from decision box 5 to decision box 7. Generally equation (9.2) is used to obtain the probability of failure for an unrevealed event. However as the component is functioning correctly at the beginning of the sequence, due to the NO outlet branch from the existence decision box being traced, the component can only fail in a short time period $t$ and is calculated by:

$$Q(t) = 1 - e^{-at}$$

A summary of the cause-consequence diagram reconstruction process given an inconsistent failure event is depicted in table 9.2.

<table>
<thead>
<tr>
<th>Failure type for Second failure event</th>
<th>Revealed</th>
<th>Unrevealed</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES Branch of existence decision box:</td>
<td>Second failure mode in second failure event occurs in the time it takes the system to travel from the first failure event to the second failure event.</td>
<td>Decision box containing first failure mode is governed by the failure of second failure mode. Second failure mode probability is set to '1' in all decision boxes beneath the existence box. First failure mode probability is set to '0'.</td>
</tr>
<tr>
<td>NO Branch of existence decision box:</td>
<td>Second failure event occurs in the time, $t$, which is equal to the time it takes the system to travel from the first failure event to the second failure event. The probability of failure is given by:</td>
<td>$Q = 1 - e^{-at}$</td>
</tr>
</tbody>
</table>

Table 9.2 Summary of procedure given an unrevealed or revealed second failure event

9.4 Algorithm for Reconstructing a Cause-Consequence Diagram which contains Inconsistent Failure Events.

An algorithm was developed whereby given a cause-consequence diagram containing an inconsistent failure event the event would be identified and the diagram modified accordingly to produce a structure which could be analysed simply.
9.4.1 Identification and Manipulation of Inconsistent Failure Events

Initially the cause-consequence diagram is stored using the arrays IFAIL, IWORK, Q, COMP, T_FAIL and F_TYPE. IFAIL, IWORK and Q contain the same data as defined for the algorithm developed in chapter 8, however for any existence decision box IFAIL represents the YES outlet branch and IWORK the NO outlet branch. COMP(i,j) is a 2 dimensional array which represents the individual components found in the fault tree structures where i represents the fault tree number and j the jth component in the fault tree structure. Depending on whether the failure of component i is revealed or unrevealed T_FAIL(i) represents the time governing the failure, t, or the maintenance test interval, θ, respectively for component i. F_TYPE represents the type of failure corresponding to each component in COMP, i.e. revealed, R, or unrevealed, U. For figure 9.6 the initial arrays were:

\[
\begin{align*}
\text{IFAIL} &= \begin{pmatrix} -1 \\ -1 \\ -2 \end{pmatrix} \\
\text{IWORK} &= \begin{pmatrix} 2 \\ 3 \\ -3 \end{pmatrix} \\
\text{Q} &= \begin{pmatrix} \text{Ft1} \\ \text{Ft2} \\ \text{Ft3} \end{pmatrix} \\
\text{COMP} &= \begin{pmatrix} \text{K2CO} & \text{P1} & 0 \\ \text{M} & \text{P3} & 0 \\ \text{K2CC} & \text{P2} & 0 \end{pmatrix} \\
\text{T\_FAIL} &= \begin{pmatrix} \theta/t \\ \theta/t \\ \theta/t \end{pmatrix} \\
\text{FTYPE} &= \begin{pmatrix} \text{U/R} & \text{U/R} \\ \text{U/R} & \text{U/R} \\ \text{U/R} & \text{U/R} \end{pmatrix}
\end{align*}
\]

For figure 9.6 component K2 is required to close and then open at decision box 3. The probability of the event 'K2 failing to close' is obtained via Ft1 and the probability of the event 'K2 failing to open' is stored in Ft3 (Figure 9.7). Comparison of these respective fault trees showed that the basic events K2CO and K2CC were identical except for the last letter. It was therefore defined that any two events with the same label excluding the last letter were deemed the same component with inconsistent failure modes. The last variable of the basic event label must be a character to warrant it being an inconsistent failure event, as it is common practice to label the same type of component with the same name and a different number attached to the end, e.g. R1, R2 and R3, symbolising Relays 1,2 and 3.
The identification of such events can be achieved alongside the identification of common failure events. Each path is scanned using the algorithm depicted in figure 8.18 and IPATH created, the relevant fault tree structures representing the decision boxes in IPATH are compared and any common failure events or inconsistent failure events identified. For the cause-consequence diagram in figure 9.6 the first path leads to the outcome No Start, NS, following the NO outlet branch stemming from decision box 1. As there exists only one decision box in the path there were no trees to compare and the next path was identified. The second path was identified as IPATH = (2,1)\(^T\), which indicated that the path consisted of following the YES branch from decision box 1 and the NO branch of decision box 2. The basic events of the fault trees Ft1 and Ft2 were compared and no dependencies were found. The third path was identified and IPATH = (2,2,1)\(^T\), representing that the YES outlet branch from decision box 1 and 2 were followed and the NO outlet branch from decision box 3. The basic events of the fault trees Ft1, Ft2 and Ft3 were compared and an inconsistent failure event was apparent in Ft1 and Ft3. Following the identification of an inconsistent failure mode, the two failure modes are stored in the array FMODE. The first failure mode is stored in FMODE(1) and the second failure mode in FMODE(2). For figure 9.6 then FMODE = (K2CO, K2CC)\(^T\). In addition the two fault tree structures containing the different failure modes are also stored in FTREE and for figure 9.6 FTREE = (Ft1, Ft3)\(^T\).

Having identified that an inconsistent failure event is present in the cause-consequence diagram the next decision to be made is dependent on the type of failure of the second failure mode. FMODE(2) and FTREE(2) are used to scan COMP and the element that the second failure mode is stored in found, IELEMENT. The array F_TYPE(FTREE(2),IELEMENT) is then inspected and the type of failure identified. The algorithm outlining this procedure is shown in figure 9.12.
Depending on whether the failure type is revealed, 'R', or unrevealed, 'U', the cause-consequence diagram reconstruction process is different. If F_TYPE shows that the failure type of the second failure mode is an 'R' then the time corresponding to the second failure mode, in the relevant fault tree, requires modification. Each basic event in each fault tree structure, in the sequence under inspection, is examined. When the second failure mode is identified the corresponding element in T_FAIL is changed to equal the time it takes the system to arrive at the decision box containing the second failure mode. This time will be predicted by the analyst. The modification process is shown as a flowchart in figure 9.13.

If F_TYPE shows that the failure type of the second failure mode is an unrevealed failure then the second failure mode is placed in a new existence box and the cause-consequence diagram duplicated on both outlet branches. The duplication process and the renumbering of the decision boxes and fault trees proceeding the NO branch of the new existence decision box is achieved in an identical manner to that for the common
failure events outlined in chapter 8. Following the YES path stemming from the existence decision box indicates that the second failure event exists. The decision boxes that are present on the YES path stemming from the existence decision box are then scanned, using an identical algorithm as previously implemented, and the various paths beneath the new existence box identified. IPATH is then used and each fault tree structure on a path inspected for the presence of the first and second failure mode, which when found is replaced with the Boolean variable '0' and '1' respectively in the appropriate element of COMP. The NO path is also scanned using an identical approach, the only difference being when the second failure mode is found in the relevant fault tree structure the time to failure for that particular component is changed in T_FAIL to equal the time it took to complete the operation and the type of failure for that particular component reset to revealed, 'R'. The above process is shown as a flowchart in figure 9.14.

Figure 9.13 Flowchart for Revealed Failure Type
Using the algorithm given in figure 8.20 the fault tree structures are scanned and all Boolean variables removed and fault trees minimised.
9.5 Application to Pressure Tank System

The pressure tank system was fully solved by applying the algorithm developed to deal with inconsistent failure modes. Following the extraction of the only common failure event, PS1, the cause-consequence diagram took the form illustrated in figure 9.4 and the relevant arrays were modified. Each path was scanned using the algorithm in figure 8.18 and following the modification of time to failure for the components K2, S1, K1, PRSW, TIM and the inclusion of three existence decision boxes for the components S1, K1 and TIM the final cause-consequence diagram was produced (Figure 9.15). The corresponding fault trees are shown in figure 9.16.

The probability of the system entering an overpressurised state was obtained by using F_TYPE, T_FAIL, COMP and the component failure data shown in table 9.3.

<table>
<thead>
<tr>
<th>Component</th>
<th>Failure Rate</th>
<th>Inspection Interval, θ</th>
<th>Mean Time to Repair, τ</th>
</tr>
</thead>
<tbody>
<tr>
<td>Switch, S1</td>
<td>S1FC: 1x10^{-6}</td>
<td>4368.0</td>
<td>36.0</td>
</tr>
<tr>
<td></td>
<td>S1FO: 8.698x10^{-4}</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Relay K1</td>
<td>K1D: 0.23x10^{-6}</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>K1CC: 0.23x10^{-6}</td>
<td>4368.0</td>
<td>36.0</td>
</tr>
<tr>
<td></td>
<td>K1CO: 0.23x10^{-6}</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Relay K2</td>
<td>K2D: 0.23x10^{-6}</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>K2CC: 0.23x10^{-6}</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>K2CO: 0.23x10^{-6}</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Timer Relay</td>
<td>TCC: 1x10^{-4}</td>
<td>4368.0</td>
<td>36.0</td>
</tr>
<tr>
<td></td>
<td>TCO: 1x10^{-4}</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Pressure Switch</td>
<td>PSWC: 1x10^{-4}</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>PSWO: 1x10^{-4}</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Fuse</td>
<td>F: 1x10^{-5}</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Power Supplies 1 &amp; 2</td>
<td>PS1: 1x10^{-6}</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td></td>
<td>PS2: 1x10^{-6}</td>
<td>NA</td>
<td>NA</td>
</tr>
<tr>
<td>Motor</td>
<td>M: 1x10^{-6}</td>
<td>NA</td>
<td>NA</td>
</tr>
</tbody>
</table>

Table 9.3 Failure Data for Pressure Tank System
The system functions twice daily and therefore the time between operations is 12 hours. The probability of failure for revealed failures between operations was hence obtained using equation (9.1) with $t=12$ hours. For unrevealed failures the probability of failure was obtained using $\theta$ and $\tau$, given in table 9.3, and equation (9.2). The probability of each fault tree was calculated using the inclusion-exclusion method, given in chapter 2, and the probability of overpressure was obtained by summing the probabilities of any sequence that terminated in the consequence '0'. There existed 12 such paths, the details of which are given in Appendix III. The probability of overpressure was calculated to equal $1.12 \times 10^{-5}$.

In addition to obtaining the probability of overpressure, the probability of the tank being empty, a safe operation and a normal operation can also be calculated and shown to equal:

\[
\begin{align*}
P(\text{Normal Operation}) &= 0.7666 \\
P(\text{Safe Operation}) &= 0.2213 \\
P(\text{Empty Tank}) &= 1.21 \times 10^{-2}
\end{align*}
\]

9.6 Summary and Conclusion

The problem to be solved was given a component with different failure modes how should the cause-consequence diagram be restructured to yield the numerically correct diagram. An algorithm has been developed that given any system that contains different operations in the same sequence of events will produce the correct cause-consequence diagram and calculate the exact failure probability. The cause-consequence diagram is reduced to a mathematically exact form by dealing with the type of failure for the different failure modes. Unrevealed and revealed failures are considered and used to determine the structure of the cause-consequence diagram.

The structurally correct cause-consequence diagram is then analysed using the BDD procedure which yields the exact probability for system failure. The main advantages of the newly developed algorithm is that a system with a start-up and shut-down sequence can be analysed without having to unnecessarily use a more complicated method, such as simulation.
Figure 9.15a First page of Final Cause-Consequence Diagram for the Pressure Tank System
Figure 9.15b Second page of Final Cause-Consequence Diagram for the Pressure Tank System
Figure 9.16 Fault Tree Structures for figure 9.15

where
Ft27=Ft3
Ft28=Ft34=Ft43=Ft52=Ft4
Ft29=Ft35=Ft44=Ft53=Ft5
Ft30=Ft39=Ft45=Ft56=Ft57=Ft6
Ft42=Ft51=Ft33
CHAPTER 10

CAUSE-CONSEQUENCE ANALYSIS OF DEPENDENT SYSTEMS

10.1 Introduction

The main disadvantage with the most commonly used reliability tool, FTA, has been highlighted as its inability to accurately analysis systems containing dependent failures. The development of the combined model in chapter 6 served as a solution to this problem. It has, however, been shown that the Cause-Consequence Diagram technique also has the capability of representing a system containing dependent features and several authors have used this modelling tool to analysis such systems.

One of the features of the cause-consequence diagram method which separates it from other consequence identification tools is the time delay symbol given in table 8.1. In order to generalise the cause-consequence diagram method, such that it can be applied to dependent types of systems, a further investigation into the use of the technique with the inclusion of the time delay symbol was initiated.

10.2 Cause-Consequence Diagram development using the Time Delay Symbol

Part of an offshore platform was investigated in order to highlight the analysis difficulties associated with systems containing time-dependent failures. The configuration of the Isolation-Blowdown system is depicted in figure 10.1 and is of a simplified form in order to highlight the required features of the system. Figure 10.1 contains three separate sections which the gas supply flows through, each of which contains a compressor and processor. Only one of these sections will be considered for the analysis as all three are identical in composition. The safety system is sequentially operated and given a gas leak of a certain size detection should occur and various safety systems activated. The detection system activates the isolation/blowdown and deluge subsystems.

In the event of a gas leak the detection system, via gas detectors, detects the leak and channels the information to a computer which de-energises a solenoid. Following the identification of a leak the isolation system is activated by the detection system, via the solenoid, and the isolation valves are closed which isolates the gas supply.
Simultaneously the blow-down system is activated which involves depressurizing each section by sending the gas supply in the section to ‘flare’. The deluge system may also be activated which uses a water supply to fight a fire, if an ignition is immediate, or mitigate an explosion by reducing overpressures if the ignition is delayed. The deluge system is assumed here to be a simplified version of that analysed in chapter 6, containing only one pump and one power supply (Figure 10.2).

Figure 10.1 Configuration of the Isolation/Blowdown System

Figure 10.2 Deluge Pump System
Activation of each system takes time and if during these time intervals an ignition source occurs then various consequences can result. The detection system takes 1 minute to detect a leak, which is the time it takes the leak to cause the module of gas to reach the detectable level, and 30 seconds to close the isolation valves and open the blowdown valves. If an ignition source is present prior to activation of the detection system then a fire will result. However, if the detection system fails prior to an ignition source occurring then an unmitigated explosion will be the outcome, due to the deluge system not being activated. An unmitigated explosion can be described as a severe explosion where limited alleviation of the overpressures has occurred due to the activation of the isolation/blowdown sub-systems. Depending on how many of the three subsystems function correctly the consequences of the leak varies. For example if all three subsystems function correctly then the leak takes 10 minutes to clear. If within this 10 minute period an ignition source occurs then a mitigated explosion will result. A mitigated explosion can be described as an explosion which is less severe than an unmitigated explosion as the overpressures have been alleviated due to the activation of the deluge system. The various clearance times and type of explosion, if an ignition source is present between these times, are tabulated in table 10.1

<table>
<thead>
<tr>
<th>Systems Function Correctly</th>
<th>Time to Clear Leak</th>
<th>Type of Explosion following an ignition source prior to clearance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Isolation, Blow-Down and Deluge</td>
<td>10 minutes</td>
<td>Mitigated</td>
</tr>
<tr>
<td>Isolation and Blow-Down</td>
<td>10 minutes</td>
<td>Unmitigated</td>
</tr>
<tr>
<td>Isolation and Deluge</td>
<td>20 minutes</td>
<td>Mitigated</td>
</tr>
<tr>
<td>Blow-Down and Deluge</td>
<td>20 minutes</td>
<td>Mitigated</td>
</tr>
<tr>
<td>Isolation</td>
<td>20 minutes</td>
<td>Unmitigated</td>
</tr>
<tr>
<td>Blow-Down</td>
<td>20 minutes</td>
<td>Unmitigated</td>
</tr>
<tr>
<td>Deluge</td>
<td>3 hours</td>
<td>Mitigated</td>
</tr>
</tbody>
</table>

Table 10.1 Clearance Times for a Gas Leak
10.2.1 Cause-Consequence Diagram Construction

The cause-consequence diagram was constructed following the three step construction algorithm outlined in section 8.2.2. The temporal sequence for the isolation/blowdown system given a leak is:

Detection System < Isolation System < Blowdown System < Deluge System

The cause-consequence diagram, following the removal of redundant decision boxes, is depicted in figure 10.3, with the corresponding fault trees shown in figure 10.4a and 10.4b.

![Figure 10.3 Cause-Consequence Diagram for figure 10.1](attachment:image.png)
The likelihood of a leak occurring was represented on the cause-consequence diagram using the initiator triangle symbol, see table 8.1. The frequency of the leak is given by $\lambda_L$, and for the purpose of this analysis was set equal to $2 \times 10^{-3} \text{ hr}^{-1}$. In order to model the system correctly the time delay symbol was incorporated in the cause-consequence diagram. Prior to each decision box representing an ignition source a time delay symbol was inserted on the sequence path which denoted the relevant time interval.
For example immediately following a leak the time delay is set equal to 1 minute, due to the fact that if an ignition source is present at any point in the time interval 0 to 1 then a fire will result. Similarly, if the detection system functions correctly but the isolation, blowdown and deluge systems all fail then the time interval given for an ignition source is set equal to 180 minutes, as if an ignition source is present within 180 minutes then an unmitigated explosion will result.

Inspection of fault trees, Ft1 and Ft2, showed that a common failure event was present. The failure of the solenoid contacts is common to both the isolation and blowdown subsystems. In order to produce the minimal cause-consequence diagram the solenoid required extraction which was achieved using the algorithm developed in section 8.4. Following the extraction of the solenoid the cause-consequence diagram changed (Figure 10.5). The corresponding fault trees are given in figure 10.6.

![Figure 10.5 Minimal Cause-Consequence Diagram for figure 10.1](image-url)
In order to perform the analysis of a cause-consequence diagram containing time delay symbols a new quantification technique required derivation. For the offshore platform modelled in this chapter the time delay symbols are used to represent a critical time interval for an ignition source. The likelihood of an ignition source being present is represented by $\lambda_i$. For each time interval then the probability of an ignition source being present is given by:

$$P(\text{Ignition in interval } t_0 \text{ to } t_1) = \int_{t_0}^{t_1} \lambda_i e^{-\lambda_i t} dt$$

A quantification complication arises when more than one ignition source is considered in any particular sequence of the cause-consequence diagram, as at one point no ignition may be present and later on in the sequence an ignition source may appear. For example, consider that in the time interval 0 to 1 minute no ignition source is present, the detection system fails to function and an ignition source is present within the next 12 hours. The ignition source is inspected twice, yet is only present during the second inspection. Equation (10.1) represents this situation.

$$\lambda_i \cdot Q_D \left[ \int_{1 \text{ mins}}^{721 \text{ mins}} \lambda_i e^{-\lambda_i t} dt \right]$$  \hspace{1cm} (10.1)

From expression (10.1) it can be seen that the time interval for the ignition source ranges from the accumulation of the times in each time delay symbol preceding the last time delay in the sequence to the accumulation of all times in every time delay symbol in the sequence. Therefore if a time delay symbol is present in a cause-consequence diagram then the integral governing the failure of the event which is dependent on the time delay is given by equation (10.2), where $t_0$ is equal to the
summation of time present in every time delay symbol preceding the last time delay symbol in the sequence and \( t_1 \) is equal to the summation of all time delays in the sequence.

\[
P(\text{event i which is dependent on time delay}) = \int_{t_0}^{t_1} \lambda e^{-\lambda t} dt \quad (10.2)
\]

Having developed a quantification technique incorporating the time delay symbol the offshore platform could be analysed. As with all cause-consequence diagrams investigated, for each type of explosion the frequency of occurrence was obtained by summation of the frequency of all paths entering each consequence.

10.3.1 Consequence 1: FIRE

The frequency of a fire occurring, given that a leak of a certain size is present, was obtained via a single path, Path 1. Path 1 represents the situation where a leak is present and an ignition source is apparent within 1 minute of the leak. The frequency was calculated to be equal to:

\[
\lambda_{P1} = \lambda_1 \cdot \int_{0}^{1} e^{-\lambda t} dt
\]

10.3.2 Consequence 2: Unmitigated Explosion

There are seven independent sequences which result in an unmitigated explosion. The frequency of an unmitigated explosion was therefore calculated via the summation of the frequency of occurrence for each of the seven paths.

Path 1 = A leak is present but no ignition source occurs in the first minute. After a minute of leakage the detection system fails and an ignition source occurs within the next 12 hours.

\[
\lambda_{P1} = \lambda_1 \cdot Q_D \left[ \int_{1\text{mins}}^{721\text{mins}} \lambda e^{-\lambda t} dt \right]
\]

Path 2 = A leak is present but no ignition source is apparent in the first minute. After a minute of leakage the detection system detects a leak and attempts to activate the safety systems, but an ignition source occurs within 30 seconds of the detection systems activation.
\[
\lambda_{p2} = \lambda_4 \cdot (1 - Q_D) \cdot \left[ \int_{1 \text{ mins}}^{1.5 \text{ mins}} \lambda_4 e^{-\lambda t_2} dt_2 \right]
\]

**Path 3:** A leak is present but no ignition source is apparent during the first minute. After a minute of leakage the detection system detects and attempts to activate the safety systems, an ignition source does not occur within 30 seconds of the detection systems activation. The common solenoid and deluge subsystem fail and an ignition occurs within 180 minutes.

\[
\lambda_{p3} = \lambda_4 \cdot (1 - Q_D) \cdot Q_{sol} \cdot Q_{f3} \cdot \lambda e^{-\lambda t_3} dt_3
\]

**Path 4:** A leak is present but no ignition source is present during the first minute. After a minute of leakage the detection system detects and attempts to activate the safety systems, an ignition source does not occur within 30 seconds of the detection system's activation. The common solenoid functions correctly and the isolation and blowdown subsystems fail. The deluge subsystem fails and an ignition occurs within 180 minutes.

\[
\lambda_{p4} = \lambda_4 \cdot (1 - Q_D) \cdot (1 - Q_{sol}) \cdot Q_{f4} \cdot Q_{f5} \cdot Q_{f10} \cdot Q_{f3} \cdot \lambda e^{-\lambda t_4} dt_4
\]

**Path 5:** A leak is present but no ignition source is present in the first minute. After a minute of leakage the detection system detects and attempts to activate the safety systems, an ignition source does not occur within 30 seconds of the detection system's activation. The common solenoid functions correctly and the isolation subsystem fails. The blow-down subsystem functions correctly but the deluge subsystem fails and an ignition occurs within 20 minutes.

\[
\lambda_{p5} = \lambda_4 \cdot (1 - Q_D) \cdot (1 - Q_{sol}) \cdot Q_{f4} \cdot (1 - Q_{f10}) \cdot Q_{f3} \cdot Q_{f3} \cdot \lambda e^{-\lambda t_5} dt_5
\]

**Path 6:** A leak is present but no ignition source is apparent during the first minute. After a minute of leakage the detection system detects and attempts to activate the safety systems, an ignition source does not occur within 30 seconds of the detection
system's activation. The common solenoid and the isolation subsystem function correctly. The blow-down subsystem and the deluge subsystem fail and an ignition occurs within 20 minutes.

\[ \lambda_{P6} = \lambda_d \cdot (1 - Q_d) \cdot (1 - Q_{solv}) \cdot Q_{f4} \cdot Q_{f5} \cdot Q_{f6} \cdot \int_{15 \text{ min}}^{21.5 \text{ min}} \lambda_i e^{-\lambda_i t} \, dt \]

**Path 7:** A leak is present but no ignition source is present during the first minute. After a minute of leakage the detection system detects and attempts to activate the safety systems, an ignition source does not occur within 30 seconds of the detection system's activation. The common solenoid, the isolation and blowdown subsystems function correctly. The deluge subsystem fails and an ignition occurs within 10 minutes.

\[ \lambda_{P7} = \lambda_d \cdot (1 - Q_d) \cdot (1 - Q_{solv}) \cdot (1 - Q_{f4}) \cdot Q_{f5} \cdot Q_{f6} \cdot \int_{15 \text{ min}}^{11.5 \text{ min}} \lambda_i e^{-\lambda_i t} \, dt \]

**10.3.3 Consequence 3: Mitigated Explosion**

There are five independent sequences which result in a mitigated explosion.

**Path 1** = A leak is present but no ignition source is apparent in the first minute. After a minute of leakage the detection system detects a leak and attempts to activate the safety systems, an ignition source does not occur within the next 30 seconds. The solenoid fails, the deluge subsystem works and an ignition occurs within 180 minutes.

\[ \lambda_{P1} = \lambda_d \cdot (1 - Q_d) \cdot Q_{solv} \cdot (1 - Q_{f4}) \cdot \int_{1.5 \text{ min}}^{181.5 \text{ min}} \lambda_i e^{-\lambda_i t} \, dt \]

**Path 2:** = A leak is present but no ignition source is apparent in the first minute. After a minute of leakage the detection system detects a leak and attempts to activate the safety systems, an ignition source does not occur within the next 30 seconds. The solenoid functions correctly while the isolation and blow-down subsystems fail. The deluge subsystem functions and an ignition occurs within 180 minutes.
\[ \lambda_{P2} = \lambda_L \cdot (1 - Q_D) \cdot (1 - Q_{solv}) \cdot Q_{sv} \cdot Q_{sc} \cdot (1 - Q_{ro}) \cdot \int_{15 \text{ min}}^{215 \text{ min}} \lambda_r e^{-\lambda t} dt \]

**Path 3:** A leak is present but no ignition source is apparent in the first minute. After a minute of leakage the detection system detects a leak and attempts to activate the safety systems, an ignition source does not occur within the next 30 seconds. The solenoid functions while the isolation subsystem fails. The blow-down subsystem and the deluge subsystem function correctly and an ignition occurs within 20 minutes.

\[ \lambda_{P3} = \lambda_L \cdot (1 - Q_D) \cdot (1 - Q_{solv}) \cdot Q_{sv} \cdot (1 - Q_{ro}) \cdot (1 - Q_{ro}) \cdot \int_{15 \text{ min}}^{215 \text{ min}} \lambda_r e^{-\lambda t} dt \]

**Path 4:** A leak is present but no ignition source is apparent in the first minute. After a minute of leakage the detection system detects a leak and attempts to activate the safety systems, an ignition source does not occur within the next 30 seconds. The solenoid and isolation subsystem function correctly. The blow-down subsystem fails while the deluge subsystem function correctly and an ignition occurs within 20 minutes.

\[ \lambda_{P4} = \lambda_L \cdot (1 - Q_D) \cdot (1 - Q_{solv}) \cdot (1 - Q_{sv}) \cdot Q_{sc} \cdot (1 - Q_{ro}) \cdot \int_{15 \text{ min}}^{215 \text{ min}} \lambda_r e^{-\lambda t} dt \]

**Path 5:** A leak is present but no ignition source is apparent in the first minute. After a minute of leakage the detection system detects a leak and attempts to activate the safety systems, an ignition source does not occur within the next 30 seconds. The solenoid, isolation subsystem, blow-down subsystem and the deluge subsystem function correctly and an ignition occurs within 10 minutes.

\[ \lambda_{P5} = \lambda_L \cdot (1 - Q_D) \cdot (1 - Q_{solv}) \cdot (1 - Q_{sv}) \cdot (1 - Q_{ro}) \cdot (1 - Q_{sc}) \cdot \int_{15 \text{ min}}^{115 \text{ min}} \lambda_r e^{-\lambda t} dt \]

Using the component failure data given in table 10.3 the frequency of each outcome for the offshore platform, given a leak of a certain size, were calculated and are illustrated in table 10.4.
Component Probability of failure (Q) or Frequency of Occurrence (λ)

<table>
<thead>
<tr>
<th>Component</th>
<th>Probability of failure (Q) or Frequency of Occurrence (λ)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Detection System</td>
<td>Q = 0.92</td>
</tr>
<tr>
<td>Isolation Value</td>
<td>Q = 1.26x10^{-4}</td>
</tr>
<tr>
<td>Pressure Relief Valve</td>
<td>Q = 1.1x10^{-3}</td>
</tr>
<tr>
<td>Power Supply</td>
<td>Q = 2.0x10^{-3}</td>
</tr>
<tr>
<td>Filter</td>
<td>Q = 2.37x10^{-3}</td>
</tr>
<tr>
<td>Pump</td>
<td>Q = 9.5x10^{-3}</td>
</tr>
<tr>
<td>Solenoid</td>
<td>Q = 1.8x10^{-4}</td>
</tr>
<tr>
<td>Blowdown Valve</td>
<td>Q = 2.0x10^{-4}</td>
</tr>
<tr>
<td>Leak</td>
<td>λ_t = 2x10^{-3}</td>
</tr>
<tr>
<td>Ignition Source</td>
<td>λ_A = 1x10^{-3}</td>
</tr>
</tbody>
</table>

Table 10.3 Component Failure Data for the Offshore Platform

<table>
<thead>
<tr>
<th>CONSEQUENCE</th>
<th>Frequency of Consequence Occurrence</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRE</td>
<td>0.333x10^{-7}</td>
</tr>
<tr>
<td>UNMITIGATED EXPLOSION</td>
<td>0.193x10^{-5}</td>
</tr>
<tr>
<td>MITIGATED EXPLOSION</td>
<td>0.3032x10^{-6}</td>
</tr>
</tbody>
</table>

Table 10.4 Results of Cause-Consequence Analysis for Offshore Platform

The results obtained are not typical as one would except a mitigated explosion to occur more frequently than an unmitigated explosion. The reason, in this case, that this trend is not apparent is due to the sequence:

\[
\lambda_{F1} = \lambda_t \cdot Q \left[ \int_{1min}^{721min} \lambda_t e^{-\lambda_t \, dt} \right]
\]

which leads to the first unmitigated explosion consequence in figure 10.5. This path has a frequency of occurrence equal to 0.19085x10^{-5} hr^{-1}, as the detection system fails with a probability of 0.08 and an ignition source has a frequency of occurrence equal to 0.0119 in a 12 hour period. The remaining sequences that lead to an unmitigated explosion have a low frequency of occurrence.

10.4 Conclusion

The problem encountered when analysing the simplified detection-isolation-blowdown system discussed in this chapter was that the system outcome is dependent on the timing of an ignition source. In order to include this feature a time delay symbol was included in the cause-consequence diagram of the system and the various consequences of a leak on an offshore platform were identified. Following the
construction of the cause-consequence diagram, using previously developed construction algorithms, a new analysis procedure was devised to incorporate the use of the time delay symbol and the system was quantified.

Conventional methods would not have been capable of modelling this particular system, as they do not possess the ability to represent the time dependency of the ignition source. It can therefore be concluded that the cause-consequence diagram method is superior to such methods for systems that contain an event which can occur at different time intervals resulting in different system outcomes.
CHAPTER 11

CONCLUSIONS AND FUTURE WORK

11.1 Summary of Work

Following an extensive critical literature review on the available reliability assessment techniques for dependency modelling, the development of a new assessment method comprising of existing reliability modelling techniques was considered worthy. Initially five main methods were investigated, namely FTA, Markov analysis, Petri net theory, event tree analysis and cause-consequence analysis. The FTA method was shown to be suitable for analysing systems containing independent failure events and to a limited degree sequential failures. The main advantage of the FTA method was found to be its representation of the system's failure logic. The Markov analysis method was shown to be capable of accurately analysing both sequential and standby failure events. However the representation of the system on the Markov state-transition diagram held no textual description of the system's failure logic and the construction process became complicated as the number of the components in the system increased. The Petri net method was initially investigated due to its ability to represent dynamic systems. The construction procedure, however, was confusing and generally the diagram is analysed using simulation. For these reasons the Petri net method was not further investigated. The event tree analysis method was shown to be capable of analysing independent and dependent failures as was the cause-consequence diagram method. The cause-consequence diagram method was chosen for further investigation, instead of the event tree analysis method, due to its ability to model low level events with different failure modes in the same sequence of events.

The development of a 'Combined Model' was completed in order to create a model that could analyse systems containing independent and dependent failure events whilst retaining the most advantageous parts of the FTA and Markov methods. The combined model approach uses a combination of FTA and Markov analysis and has been successfully demonstrated by its application to industrial systems. A computer
program has been developed which implements the analysis of systems containing sequential failure events, standby failure events and independent failure events. The program requires an input file, the fault tree structure including the PAND gate and new standby gates, that contains information about the system and how each component in the system is related. If both a qualitative and quantitative assessment of the system is required then the program needs an additional input file which contains the reliability data for each component in the fault tree. The analysis of the system is performed using the data files inputted, transparently to the user. The quantification of a system, using the combined model, is achieved through identification of independent subtrees of the fault tree. These subtrees are then analysed using either FTA or Markov analysis depending on whether the subtrees contain independent or dependent failure events respectively. Once solved the subtrees are replaced in the fault tree structure as new basic events. This process is continued until only one independent subtree requires solution.

The combined model was tested against the most appropriate conventional modelling technique for a deluge pump system and shown to be more efficient in terms of speed of computation and memory requirements. The increase in efficiency is due to analysing small sections using the most appropriate technique, rather than large sections using an inappropriate method for the majority of the section, which occurs when analysing dependent failure events using traditional methods. The combined model was used to obtain quantitative information about a system. This included the calculation of system failure, system unconditional failure intensity, the failure rate of the system and the repair rate of the system. The calculation of these parameters using the combined model proved to be superior to the approximations obtained using the FTA method. Therefore the combined model produced more accurate results, in a shorter time, when compared to the FTA procedure.

The use of the combined model was further developed by embedding it within an optimisation scheme. As the combined model retained the fault tree structure, house events were used to represent all different design options on a single fault tree. This fault tree was then subjected to a Genetic Algorithm optimisation scheme, in order to
highlight the 'best design'. A program was developed to incorporate the optimisation scheme and a dependent industrial system was successfully optimised.

During the literature review it was noted that the cause-consequence diagram method had interesting features. It was observed that the method had the ability of modelling both independent and dependent failure events on a diagram which retained both the failure and success logic of the system. As with the FTA method, the diagram is logically constructed and would therefore be appealing to engineers. During the literature review it was highlighted that the modelling technique had been used to analyse dependent failure events yet no generalised set of rules for construction or quantification had been developed. It was therefore decided that an investigation into the development of such rules should be initiated. An investigation into the cause-consequence diagram method for a system containing independent failure events highlighted that the cause-consequence diagram method was an alternative representation of a BDD, and therefore more efficient than FTA. A further advantage noted was that the cause-consequence diagram method retained the failure logic description, which is lost when a fault tree is converted to a BDD. A general set of construction and quantification rules were developed for the cause-consequence diagram method and implemented successfully on an industrial system.

As the main focus of the thesis was the analysis of dependent failure events the cause-consequence diagram method was developed further by investigating its uses in analysing dependent systems. In order to quantify dependent systems new developments in analysing the cause-consequence diagram were achieved. The development of an algorithm to identify and modify components that were required to operate in different modes, made it possible to model multi-phase systems. In addition to this the use of a time delay symbol meant that modelling systems, where the timing of event failures is relevant, was also possible. All new developments were successfully implemented in the quantification of real-life industrial systems.
11.2 Conclusions

The Combined Model

1. The combined model overcomes some of the disadvantages of conventional FTA to produce an accurate result for systems containing dependent failure events. FTA assumes independence, due to Kinetic Tree Theory, and therefore cannot model "heavy" dependent failure events accurately. The combined model produces an accurate analysis of dependent systems by analysing the dependencies using Markov analysis and returning the solution of the analysis as a new basic event in the fault tree structure.

2. The combined model overcomes some of the disadvantages of the Markov analysis method. Firstly the combined model contains a textual description of the systems behaviour, which is not apparent on the Markov diagram, and secondly the combined model reduces the number of components that are required to be analysed using Markov analysis, therefore reducing the size of the Markov diagram and the computational effort required to solve it. The reduction in the size of the Markov diagram is due to the combined model identifying independent subtrees which are either independent or dependent in nature. The identification of such subtrees means that the entire system is not required to be solved using a single Markov analysis.

3. The combined model is capable of evaluating fault trees containing a combination of independent and dependent failure events and can obtain a full range of system reliability parameters. Due to the modularization of the fault tree and the analysis of subtrees containing dependent failure events using Markov analysis, and independent subtrees using FTA, the evaluation of the system is performed quickly with a minimal amount of memory requirement.

4. The application of the combined model to an industrial system has shown that this new assessment method improves the efficiency and accuracy of the quantification of systems containing dependent failure events.
5. The combined model was embedded within an optimisation scheme and successfully used to highlight the best design for an industrial system which contained dependent failure events, hence increasing its useability.

**The Cause-Consequence Diagram Method**

1. A generalised set of rules for the construction and quantification of a cause-consequence diagram for independent systems have been established and justified. It was shown that the cause-consequence diagram method produces an alternative BDD representation. This is achieved by considering the affect of component success and failure on the system's functionality. As the cause-consequence diagram method is equivalent to a BDD it produces the exact system failure probability efficiently, while retaining the textual description of the system failure logic. The cause-consequence diagram construction procedure results in a certain degree of modularization, as the causes of the NO outlet branch of each decision box can be governed by a fault tree. In any particular sequence through the cause-consequence diagram the fault trees attached to each decision box may contain the same failure events. An algorithm has been developed that extracts such common cause events in order to accurately quantify the diagram. A further modularization algorithm has been developed which extracts common cause modules from fault tree structures in the same sequence. Inclusion of this algorithm has been shown to further increase the efficiency of the cause-consequence diagram method.

2. A generalised set of quantification rules have been established for a cause-consequence diagram representing certain dependent failure events. Components that are involved in different operation modes or have different outcomes depending on what time they occur, can be modelled using the cause-consequence diagram method developed. Components that perform different functions during different phases of system operation are modelled depending on whether their failure events are revealed or unrevealed. An algorithm has been developed that identifies dependent failure modes in the same sequence and modifies the cause-
consequence diagram accordingly depending on whether the failure events are revealed or unrevealed. Systems that result in different consequences due to the specific timing of events are modelled on a cause-consequence diagram using a time delay symbol. An algorithm has been developed that quantifies such time delays accurately using integration.

3. The combined model can be used to accurately model sequential and standby failure events. The cause-consequence diagram method can be used to model sequential failures and standby failure events. The cause-consequence diagram method, however, is best utilised in systems where repair during operation is not possible as inclusion of repair results in an infinite diagram which has yet to be accurately solved.

11.3 Future Work

11.3.1 Modelling of Different Dependent Failures

The dependent failures investigated in this thesis, sequential and standby failures, are two of the most commonly witnessed in safety systems. However it is evident that other dependent failure events do exist, such as secondary failure events. Therefore these dependencies could be incorporated in the combined model, which would increase its usability. In addition to this the switching operator, found on many standby systems, could be considered as an event that could also fail and hence effect the standby system. In this thesis the switching operator was considered to be perfectly reliable.

11.3.2 Modelling Repair on a Cause-Consequence Diagram

The developments achieved with the cause-consequence diagram method have resulted in a new and exciting modelling technique which is highly applicable to independent and dependent systems. One of the main disadvantages with this method is that inclusion of component repair into the diagram is complex. The result of including repair can be an infinite diagram, the quantification of which has yet to be successfully achieved. The development of new rules to construct and quantify a
cause-consequence diagram which includes repair would increase the usability of the method even further and would make the method more applicable than existing conventional reliability modelling tools.

11.3.3 Time Operating Systems

It was suggested that feedback loops could be incorporated into the cause-consequence diagram in order to represent systems that operate at set time intervals. The quantification method used to include feedback loops was however inconclusive. An investigation could therefore be initiated into 'How best to quantify a system using the cause-consequence diagram method and feedback loops'. One way could be to restart each sequence with the probability calculated at the end of the last time sequence. If a quantification technique could be developed to include such a feature then the cause-consequence diagram could be applied to an even wider field of systems.
APPENDIX I

DATA FILES FOR DELUGE PUMP SYSTEM

*.ATS FILE FOR DELUGE PUMP SYSTEM

<table>
<thead>
<tr>
<th>GTOP WSBY</th>
<th>3 4 4 0</th>
<th>G1 OR 0 0 1 1</th>
<th>G2 tvl</th>
</tr>
</thead>
<tbody>
<tr>
<td>G1 OR 0 0 1 1</td>
<td>G3 isol11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G2 OR 0 0 1 1</td>
<td>G4 prv1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G3 OR 0 0 1 1</td>
<td>G5 G6 ep 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G4 OR 0 0 2 1</td>
<td>G7 tv2 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G5 OR 0 0 0 2</td>
<td>G8 isol12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G6 OR 0 0 0 2</td>
<td>G9 prv2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G7 OR 0 0 1 1</td>
<td>G10 ep 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G8 OR 0 0 1 1</td>
<td>G11 $EP1FR 1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G9 OR 0 0 1 1</td>
<td>G12 ep 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G10 OR 0 0 2 1</td>
<td>G13 tv3 3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G11 OR 0 0 0 2</td>
<td>G14 isol3 4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G12 OR 0 0 0 2</td>
<td>G15 prv3 5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G13 OR 0 0 1 1</td>
<td>G16 dp 6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G14 OR 0 0 1 1</td>
<td>G17 $DP1FR 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G15 OR 0 0 1 1</td>
<td>G18 isol23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G16 OR 0 0 2 1</td>
<td>G19 tv4 10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G17 OR 0 0 0 2</td>
<td>G20 isol14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G18 OR 0 0 0 2</td>
<td>G21 prv4 11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G19 OR 0 0 1 1</td>
<td>G22 dp 12</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G20 OR 0 0 1 1</td>
<td>G23 $DP2FR 13</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G21 OR 0 0 1 1</td>
<td>G24 isol24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G22 OR 0 0 2 1</td>
<td>G24 isol24</td>
<td></td>
<td></td>
</tr>
<tr>
<td>G23 OR 0 0 2 1</td>
<td>G24 isol24</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**ADQ FILE FOR DELUGE PUMP SYSTEM**

<table>
<thead>
<tr>
<th>NAME</th>
<th>CODE</th>
<th>( Q_i )</th>
<th>( \lambda_i )</th>
<th>( \mu_i )</th>
<th>( \theta )</th>
<th>( \tau )</th>
<th>( \lambda_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>tv1</td>
<td>1</td>
<td>0</td>
<td>0.0000055</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>isol11</td>
<td>1</td>
<td>0</td>
<td>0.0000034</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>prv1</td>
<td>1</td>
<td>0</td>
<td>0.00003</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>ep</td>
<td>5</td>
<td>0</td>
<td>0.002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>isol21</td>
<td>1</td>
<td>0</td>
<td>0.0000034</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>f1</td>
<td>1</td>
<td>0</td>
<td>0.0000064</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>$EP1FR</td>
<td>5</td>
<td>0</td>
<td>0.0007</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00035</td>
</tr>
<tr>
<td>EP1FS</td>
<td>0</td>
<td>0.0095</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>tv2</td>
<td>1</td>
<td>0</td>
<td>0.0000055</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>isol12</td>
<td>1</td>
<td>0</td>
<td>0.0000034</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>prv2</td>
<td>1</td>
<td>0</td>
<td>0.00003</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>ep</td>
<td>5</td>
<td>0</td>
<td>0.002</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>isol22</td>
<td>1</td>
<td>0</td>
<td>0.0000034</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>f2</td>
<td>1</td>
<td>0</td>
<td>0.0000064</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>$EP2FR</td>
<td>5</td>
<td>0</td>
<td>0.0007</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00035</td>
</tr>
<tr>
<td>EP2FS</td>
<td>0</td>
<td>0.0095</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>tv3</td>
<td>1</td>
<td>0</td>
<td>0.0000055</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>isol13</td>
<td>1</td>
<td>0</td>
<td>0.0000034</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>prv3</td>
<td>1</td>
<td>0</td>
<td>0.00003</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>dp</td>
<td>5</td>
<td>0</td>
<td>0.0001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>isol23</td>
<td>1</td>
<td>0</td>
<td>0.0000034</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>f3</td>
<td>1</td>
<td>0</td>
<td>0.0000064</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>$DP1FR</td>
<td>5</td>
<td>0</td>
<td>0.0007</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00035</td>
</tr>
<tr>
<td>DP1FS</td>
<td>0</td>
<td>0.0095</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>tv4</td>
<td>1</td>
<td>0</td>
<td>0.0000055</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>isol14</td>
<td>1</td>
<td>0</td>
<td>0.0000034</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>prv4</td>
<td>1</td>
<td>0</td>
<td>0.00003</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>dp</td>
<td>5</td>
<td>0</td>
<td>0.0001</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>isol24</td>
<td>1</td>
<td>0</td>
<td>0.0000034</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>f4</td>
<td>1</td>
<td>0</td>
<td>0.0000064</td>
<td>0</td>
<td>26</td>
<td>24</td>
<td>0</td>
</tr>
<tr>
<td>$DP2FR</td>
<td>5</td>
<td>0</td>
<td>0.0007</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.00035</td>
</tr>
<tr>
<td>DP2FS</td>
<td>0</td>
<td>0.0095</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
APPENDIX II

FAULT TREE REPRESENTATION FOR THE DELUGE PUMP SYSTEM
DIESEL PUMP STREAM 1 fails

Diesel Pump 1 fails

Pump Stream 1 fails

Pump Stream 5 failed

NE < 1
HPS = 0

No Pressure at outlet

Test valve fails

Pressure relief valve fails

Isol 15

Pump Type fails

No Pressure at isol 2 outlet

Diesel Power fails

Prv 5

Filter blocked

Isol 2 valve fails closed

Isol 25
Diesel Pump Stream 3 fails

Pump Stream 2 fails

NE > 3
HP7 = 0

No Pressure at inlet

No Pressure at sol outlet

Pressure relief valve fails

Pump Type fails

No Pressure at sol 2 outlet

Diesel Power fails

Flow

Isol 17

Water blocked

Isol 2 valve fails closed

Isol 2

Pv7

dp
APPENDIX III

FAILURE SEQUENCE PATHS FOR PRESSURE TANK SYSTEM

Path 1 = \{Decision Box 24,3,4,5,6\}

\[ P_1 = \{F_{t24}, (1-P(F_{t3})), (1-P(F_{t4})), P(F_{t5}) \} \]
\[ = Q_{SIFC}. [1-(1-Q_{PS1}).(1-Q_{K2D}).(1-Q_{K2CO}).(1-Q_{PSWO})] \]
\[ .[1-(1-Q_F)(1-Q_M)(1-Q_{PS2})].Q_{PSWC}^{t=30\text{mins}} \]

Path 2 = S1 failed closed and K2 contacts close, Motor starts and tank fills, the pressure switch opens but the K2 contacts have failed closed (K2CC could only happen in a short time period so t=30 minutes)

\[ P_2 = \{F_{t24}, (1-P(F_{t3})), (1-P(F_{t4})), (1-P(F_{t5})) \} \]
\[ = Q_{SIFC}. [1-(1-Q_{PS1}).(1-Q_{K2D}).(1-Q_{K2CO}).(1-Q_{PSWO})] \]
\[ .[1-(1-Q_F)(1-Q_M)(1-Q_{PS2})].Q_{PSWC}^{t=30\text{mins}} .Q_{K2CC}^{t=30\text{min}} \]

Path 3 = S1 has not failed closed but closes when depressed. The switch fails to open indicated that it has failed closed (therefore S1 has failed in a very small time period which is set equal to 10 seconds). K2 contacts close, Motor starts and tank fills but the pressure switch fails to open(PSWC could only happen in a short time period so t=30 minutes).

\[ P_3 = (1-P(F_{t24})).(1-P(F_{t25})).(1-P(F_{t26})).(1-P(F_{t27})).(1-P(F_{t28})).P(F_{t29}) \]
\[ = (1-Q_{SIFC}).[(1-Q_{PSWO})]Q_{SIFC}^{t=10s}. [1-(1-Q_{PS1}).(1-Q_{K2D}).(1-Q_{K2CO}).(1-Q_{PSWO})] \]
\[ .[1-(1-F)(1-M)(1-PS2)].Q_{PSWC}^{t=30\text{mins}} \]
Path 4 = S1 has not failed closed but closes when depressed. The switch fails to open indicated that it has failed closed (therefore S1 has failed in a very small time period which is set equal to 10 seconds). K2 contacts close, Motor starts and tank fills the pressure switch opens but the K2 contacts have failed closed (K2CC could only happen in a short time period so $t=30$ minutes)

$$= (1-P(Ft24)).(1-P(Ft25)).(P(Ft26)).(1-P(Ft27)).(1-P(Ft28)).(1-P(Ft29)).P(Ft30)$$

$$= (1-Q_{S1FC}).[(1-Q_{S1FO}).Q_{S1FC}^{t=10s}].(1-(1-(1-Q_{PS1}).(1-Q_{K2D}).(1-Q_{K2CO}).(1-Q_{PSWO}))].$$

$$[1-(1-(1-F). (1-M). (1-PS2))].(1-Q_{PSW}^{t=30mins}).Q_{K2CC}^{t=30min}$$

Path 5 = S1 has not failed closed but closes when depressed and then opens. The power supply, P1 is present but K1 exists failed closed. K2 contacts close, Motor starts and the tank fills but the pressure switch fails to open(PSWC could only happen in a short time period so $t=30$ minutes).

$$= (1-P(Ft24)).(1-P(Ft25)).(1-P(Ft26)).(1-P(Ft31)) .(P(Ft40)).(1-P(Ft33)).$$

$$P(1-Ft34).P(Ft35)$$

$$= (1-Q_{S1FC}).[(1-Q_{S1FO}).(1-Q_{S1FC}^{t=10s}).(1-Q_{PS1}).Q_{K1FC}^{t=10s}].1-(1-[1-Q_{K2D}].$$

$$(1-Q_{K2CO}).(1-Q_{PSWO})).[1-(1-(1-F). (1-M). (1-PS2))].Q_{PSWC}^{t=30mins}$$

Path 6 = S1 has not failed closed but closes when depressed and then opens. The power supply, P1 is present but K1 exists failed closed. K2 contacts close, Motor starts and the tank fills. The pressure switch opens but the K2 contacts have failed closed (K2CC could only happen in a short time period so $t=30$ minutes)

$$= (1-P(Ft24)).(1-P(Ft25)).(1-P(Ft26)).(1-P(Ft31)) .(P(Ft40)).(1-P(Ft33)).$$

$$(1-P(Ft34)).(1-P(Ft35)).$$

$$= (1-Q_{S1FC}).[(1-Q_{S1FO}).(1-Q_{S1FC}^{t=10s})).(1-Q_{PS1}).Q_{K1FC}^{t=10s}].1-(1-[1-Q_{K2D}].$$

$$(1-Q_{K2CO}).(1-Q_{PSWO})).[1-(1-(1-F). (1-M). (1-PS2))].Q_{PSWC}^{t=30mins})Q_{K2CC}^{t=30min}$$
Path 7 = S1 has not failed closed but closes when depressed and then opens. The power supply, P1 is present and K1 does not exists failed closed. The timer exists failed closed. K2 contacts close and the Motor starts which fills the tank and the pressure switch fails closed(PSWC could only happen in a short time period so t=30 minutes)

\[
= (1-P(Ft24))(1-P(Ft25)) (1-P(Ft26))(1-P(Ft31))(1-P(Ft40)) \cdot P(Ft49).
\]

\[
(1-P(Ft42))(1-P(Ft43)) \cdot P(Ft44)
\]

\[
= (1-Q_{SIFC})(1-(1-Q_{SIFO}))(1-Q_{SIFC}^{t=10s})(1-Q_{PS1})(1-Q_{KIFC}) \cdot Q_{TIMCC} \cdot [1-(1-[1-(1-Q_{K2D})(1-Q_{K2CO})(1-Q_{PSWO})])][1-(1-(1-F)(1-M)(1-PS2))] \cdot Q_{PSWC}^{t=30mins}
\]

Path 8 = S1 has not failed closed but closes when depressed and then opens. The power supply, P1 is present and K1 does not exists failed closed. The timer exists failed closed. K2 contacts close and the Motor starts which fills the tank. The pressure switch opens but K2 contacts fail closed(K2CC could only happen in a short time period so t=30 minutes)

\[
= (1-P(Ft24))(1-P(Ft25))(1-P(Ft26))(1-P(Ft31))(1-P(Ft40)) \cdot P(Ft49).
\]

\[
(1-P(Ft42))(1-P(Ft43))(1-P(Ft44)) \cdot P(Ft48)
\]

\[
= (1-Q_{SIFC})(1-(1-Q_{SIFO}))(1-Q_{SIFC}^{t=10s})(1-Q_{PS1})(1-Q_{KIFC}) \cdot Q_{TIMCC} \cdot [1-(1-[1-(1-Q_{K2D})
\]

\[
-(1-Q_{K2CO})(1-Q_{PSWO}))][1-(1-(1-F)(1-M)(1-PS2))] \cdot (1-Q_{PSWC}^{t=30mins}) \cdot Q_{K2CC}^{t=30mins}
\]
Path 9 = S1 has not failed closed but closes when depressed and then opens. The power supply, P1 is present and K1 does not exists failed closed. The timer does not exists failed closed and K1 contacts close. K2 contacts close and the Motor starts which fills the tank. The pressure switch fails closed (PSWC could only happen in a short time period so t=30 minutes) and the timer fails to open (TIMCC can only occur in small time interval so t=30 minutes).

\[= (1-P(Ft24))\cdot(1-P(Ft25))\cdot(1-P(Ft26))\cdot(1-P(Ft31))\cdot(1-P(Ft40))\cdot(1-P(Ft49))\cdot(1-P(Ft50))\cdot(1-P(Ft51))\cdot(1-P(Ft52))\cdot P(Ft53)\cdot P(Ft54)\]

\[= (1-Q_{SIFC})\cdot(1-Q_{SIFO})\cdot(1-Q_{SIFC}^{t=10s})\cdot(1-Q_{PS1})\cdot(1-Q_{K1FC})\cdot(1-Q_{TIMCC})\cdot [1-(1-[(1-Q_{K1D})\cdot(1-Q_{K1CO})\cdot(1-Q_{TIMCC})])]\cdot[1-(1-[(1-Q_{K2D})\cdot(1-Q_{K2CO})\cdot(1-Q_{PSWO})])]\cdot[1-(1-[(1-F)\cdot(1-M)\cdot(1-PS2)])]\cdot(Q_{PSWC}^{t=30mins})\cdot Q_{TIMCC}^{t=30mins}\]

Path 10 = S1 has not failed closed but closes when depressed and then opens. The power supply, P1 is present and K1 does not exists failed closed. The timer does not exists failed closed and K1 contacts close. K2 contacts close and the Motor starts which fills the tank. The pressure switch fails closed (PSWC could only happen in a short time period so t=30 minutes). The timer opens but K1 contacts failed closed (K1CC can only fail in small time interval, t=30 minutes)

\[= (1-P(Ft24))\cdot(1-P(Ft25))\cdot(1-P(Ft26))\cdot(1-P(Ft31))\cdot(1-P(Ft40))\cdot(1-P(Ft49))\cdot(1-P(Ft50))\cdot(1-P(Ft51))\cdot(1-P(Ft52))\cdot P(Ft53)\cdot P(Ft54)\cdot P(Ft55)\]

\[= (1-Q_{SIFC})\cdot(1-Q_{SIFO})\cdot(1-Q_{SIFC}^{t=10s})\cdot(1-Q_{PS1})\cdot(1-Q_{K1FC})\cdot(1-Q_{TIMCC})\cdot [1-(1-[(1-Q_{K1D})\cdot(1-Q_{K1CO})\cdot(1-Q_{TIMCC})])]\cdot[1-(1-[(1-Q_{K2D})\cdot(1-Q_{K2CO})\cdot(1-Q_{PSWO})])]\cdot[1-(1-[(1-F)\cdot(1-M)\cdot(1-PS2)])]\cdot(Q_{PSWC}^{t=30mins})\cdot (1-Q_{TIMCC}^{t=30mins})\cdot Q_{K1CC}^{t=30mins}\]
Path 11: S1 has not failed closed but closes when depressed and then opens. The power supply, P1 is present and K1 does not exist fails closed. The timer does not exist and K1 contacts close. K2 contacts close and the Motor starts which fills the tank. The pressure switch fails closed (PSWC could only happen in a short time period so t=30 minutes). The timer opens, K1 contacts open but K2 contacts failed closed (K2CC can only fail in small time interval, t=30 minutes)

\[= (1-P(Ft24)).(1-P(Ft25)).(1-P(Ft26)).(1-P(Ft31)) .(1-P(Ft40)).(1-P(Ft49)).
(1-P(Ft50)).(1-P(Ft51)).(1-P(Ft52)).(P(Ft53)).(1-P(Ft54)).(1-P(Ft55)).(P(Ft56))\]

\[= (1-QSIFC) \cdot (1-QSIFO) \cdot (1-QK1FC) \cdot (1-QTIMCC) \cdot [1-(1-\{1-QK1D\} \cdot (1-QK1CO) \cdot (1-QTIMCO))], [1-(1-\{1-QK2D\} \cdot (1-QK2CO) \cdot (1-QPSWO))], [1-(1-\{1-F\} \cdot (1-M) \cdot (1-PS2))], (QPSWC \cdot 30mins) \cdot (QTIMCC \cdot 30mins) \cdot (QK1CC \cdot 30mins) \cdot (QK2CC \cdot 30mins)\]

Path 12: S1 has not failed closed but closes when depressed and then opens. The power supply, P1 is present and K1 does not exist fails closed. The timer does not exist and K1 contacts close. K2 contacts close and the Motor starts which fills the tank. The pressure switch opens but K2 contacts fail closed (K2CC could only happen in a short time period so t=30 minutes)

\[= (1-P(Ft24)).(1-P(Ft25)).(1-P(Ft26)).(1-P(Ft31)) .(1-P(Ft40)).(1-P(Ft49)).
(1-P(Ft50)).(1-P(Ft51)).(1-P(Ft52)).(1-P(Ft53)).(P(Ft54)).\]

\[= (1-QSIFC) \cdot (1-QSIFO) \cdot (1-QK1FC) \cdot (1-QTIMCC) \cdot [1-(1-\{1-QK1D\} \cdot (1-QK1CO) \cdot (1-QTIMCO))], [1-(1-\{1-QK2D\} \cdot (1-QK2CO) \cdot (1-QPSWO))], [1-(1-\{1-F\} \cdot (1-M) \cdot (1-PS2))], (QPSWC \cdot 30mins) \cdot (QK2CC \cdot 30mins)\]
REFERENCES


9) J.L. Peterson, “Petri Nets” Computing Surveys, Vol. 9, No. 3 September 1977

11) T.S Liu, S.B. Chiou, "The application of Petri Nets to Failure Analysis"  

12) A Villemeur, "Reliability, Availability, Maintainability and Safety Assessment",  
Wiley, Chichester, 1991

13) J. Adamson, G.E. Jones, "An Event Tree Analysis of a Modulating Package  
Burner Control System", Project M6401, British Gas plc, March 1988

14) D.S. Nielsen, "The Cause/Consequence Diagram Method as a Basis for  
Quantitative Accident Analysis", Danish Atomic Energy Commission, RISO-M-  
1374, May 1971

15) H.A. Watson & Bell Telephone Laboratories, "Launch Control Safety Study ",  
Bell Telephone Lab, Murray Hill, NJ, USA, 1961

Symposium 1965


19) E.G. Frankel, "Systems Reliability and Risk Analysis", 1984; The Hague;  
Lancaster; Nijhoff.

20) J.A. Rice, "Mathematical Statistics and Data Analysis", 1995, Belmont,  
California, Duxbury Press.

21) C.E. Shannon, "A Symbolic Analysis of Relay and Switching Circuits", AIEE  
Transactions, Vol. 57, pp 713, 1938
22) W.G. Schneeweiss, “Fault Tree Analysis using a Binary Decision Tree”, IEEE Transactions on Reliability, Vol 34, No. 5, 1985 December


62) W.G Schneeweiss, "Boolean Functions with Engineering Applications and

63) A. Anand, A.K. Somani, "Hierarchical Analysis of Fault Tree with
Dependencies, using Decomposition", Annual Reliability & Maintainability
Symposium, 1998

64) D F Hassl, N H Roberts, W E Vesely, F F Goldberg, “Fault Tree Handbook”,
US Nuclear Regulatory Commission, 1981, NUREG-0492